Introduction
Let’s begin

• Goal
  – Teach you how to program effectively

• Skills and information to be acquired
  – Mental model of computer and network behavior
  – Problem solving
  – Object-oriented design
  – Java
Computer Organization

• Computer advertisement specification
  – Intel® Pentium 4 Processor at 3.06GHz with 512K cache
  – 512MB DDR SDRAM
  – 200GB ATA-100 Hard Drive (7200 RPM, 9.0 ms seek time)
  – 17” LCD Monitor
  – 64MB NVIDIA GeForce4 MX Graphics Card®
  – 16x Max DVD-ROM Drive
  – 48x/24x/48x CD-RW Drive
  – 56K PCI Telephony Modem
  – Windows XP Home Edition SP2 ®
  – 10/100 Fast Ethernet Network Card
Computer Organization

- Input Devices
- Central Processing Unit
- Memory
- Output Devices

The diagram shows the relationship between these components in a computer system.
Computer Organization

- Computer advertisement specification
  - Intel® Pentium 4 Processor
    at 3.06GHz with 512K cache
  - 512MB DDR SDRAM
  - 200GB ATA-100 Hard Drive
    (7200 RPM, 9.0 ms seek time)
  - 17” LCD Monitor
  - 64MB NVIDIA GeForce4
    MX Graphics Card®
  - 16x Max DVD-ROM Drive
  - 48x/24x/48x CD-RW Drive
  - 56K PCI Telephony Modem
  - Windows XP Home Edition SP2 ®
  - 10/100 Fast Ethernet Network Card

3.06 billion operations per second
Computer Organization

- Computer advertisement specification
  - Intel® Pentium 4 Processor at 3.06GHz with 512K cache
  - 512MB DDR SDRAM
  - 200GB ATA-100 Hard Drive (7200 RPM, 9.0 ms seek time)
  - 17” LCD Monitor
  - 64MB NVIDIA GeForce4 MX Graphics Card®
  - 16x Max DVD-ROM Drive
  - 48x/24x/48x CD-RW Drive
  - 56K PCI Telephony Modem
  - Windows XP Home Edition SP2 ®
  - 10/100 Fast Ethernet Network Card

512 million bytes of memory that can be transferred at double the normal rate

A byte is 8 bits

A bit is a 0 or a 1
Computer Organization

- Intel® Pentium 4 Processor at 3.06GHz with 512K cache
- 512MB DDR SDRAM
- 200GB ATA-100 Hard Drive (7200 RPM, 9.0 ms seek time)
- 17” LCD Monitor
- 64MB NVIDIA GeForce4 MX Graphics Card®
- 16x Max DVD-ROM Drive
- 48x/24x/48x CD-RW Drive
- 56K PCI Telephony Modem
- Windows XP Home Edition SP2®
- 10/100 Fast Ethernet Network Card

Stores 200 billion bytes of data. You want high RPM and low seek time. 0.009 seconds is average.
Computer Organization

- Computer advertisement specification
  - Intel® Pentium 4 Processor at 3.06GHz with 512K cache
  - 512MB DDR SDRAM
  - 200GB ATA-100 Hard Drive (7200 RPM, 9.0 ms seek time)
  - 17” LCD Monitor
  - 64MB NVIDIA GeForce4 MX Graphics Card®
  - 16x Max DVD-ROM Drive
  - 48x/24x/48x CD-RW Drive
  - 56K PCI Telephony Modem
  - Windows XP Home Edition SP2 ®
  - 10/100 Fast Ethernet Network Card

17” on the diagonal. Resolution up to 1,280 by 1,024 pixels
Computer Organization

- Computer advertisement specification
  - Intel® Pentium 4 Processor at 3.06GHz with 512K cache
  - 512MB DDR SDRAM
  - 200GB ATA-100 Hard Drive (7200 RPM, 9.0 ms seek time)
  - 17” LCD Monitor
  - 64MB NVIDIA GeForce4 MX Graphics Card®
  - 16x Max DVD-ROM Drive
  - 48x/24x/48x CD-RW Drive
  - 56K PCI Telephony Modem
  - Windows XP Home Edition SP2®
  - 10/100 Fast Ethernet Network Card

Microprocessor for displaying images with 64 million bytes of memory. More memory supports more colors and higher resolution.
Computer Organization

- Computer advertisement specification
  - Intel® Pentium 4 Processor at 3.06GHz with 512K cache
  - 512MB DDR SDRAM
  - 200GB ATA-100 Hard Drive (7200 RPM, 9.0 ms seek time)
  - 17” LCD Monitor
  - 64MB NVIDIA GeForce4 MX Graphics Card®
  - 16x Max DVD-ROM Drive
  - 48x/24x/48x CD-RW Drive
  - 56K PCI Telephony Modem
  - Windows XP Home Edition SP2 ®
  - 10/100 Fast Ethernet Network Card

Reads DVDs 16 times faster than a basic DVD drive. Can hold up to 8 billion bytes of data.
Computer Organization

- Computer advertisement specification
  - Intel® Pentium 4 Processor at 3.06GHz with 512K cache
  - 512MB DDR SDRAM
  - 200GB ATA-100 Hard Drive (7200 RPM, 9.0 ms seek time)
  - 17” LCD Monitor
  - 64MB NVIDIA GeForce4 MX Graphics Card®
  - 16x Max DVD-ROM Drive
  - 48x/24x/48x CD-RW Drive
  - 56K PCI Telephony Modem
  - Windows XP Home Edition SP2 ®
  - 10/100 Fast Ethernet Network Card

Can read and write CDs. Can hold 650 million bytes of data. Reads at 48 times faster and writes 24 times faster than a basic drive.
Computer Organization

- Computer advertisement specification
  - Intel® Pentium 4 Processor at 3.06GHz with 512K cache
  - 512MB DDR SDRAM
  - 200GB ATA-100 Hard Drive (7200 RPM, 9.0 ms seek time)
  - 17” LCD Monitor
  - 64MB NVIDIA GeForce4 MX Graphics Card®
  - 16x Max DVD-ROM Drive
  - 48x/24x/48x CD-RW Drive
  - 56K PCI Telephony Modem
  - Windows XP Home Edition SP2 ®
  - 10/100 Fast Ethernet Network Card

Can send or receive up to 56 thousand bits per second
Computer Organization

- Computer advertisement specification
  - Intel® Pentium 4 Processor at 3.06GHz with 512K cache
  - 512MB DDR SDRAM
  - 200GB ATA-100 Hard Drive (7200 RPM, 9.0 ms seek time)
  - 17” LCD Monitor
  - 64MB NVIDIA GeForce4 MX Graphics Card®
  - 16x Max DVD-ROM Drive
  - 48x/24x/48x CD-RW Drive
  - 56K PCI Telephony Modem
  - Windows XP Home Edition SP2 ®
  - 10/100 Fast Ethernet Network Card

Computer operating system using a graphical interface
Computer Organization

- Computer advertisement specification
  - Intel® Pentium 4 Processor at 3.06GHz with 512K cache
  - 512MB DDR SDRAM
  - 200GB ATA-100 Hard Drive (7200 RPM, 9.0 ms seek time)
  - 17” LCD Monitor
  - 64MB NVIDIA GeForce4 MX Graphics Card®
  - 16x Max DVD-ROM Drive
  - 48x/24x/48x CD-RW Drive
  - 56K PCI Telephony Modem
  - Windows XP Home Edition SP2 ®
  - 10/100 Fast Ethernet Network Card

Can send or receive data at two rates – 10 or 100 million bytes per second
Network communication

- Communication protocol
  - Set of rules that govern how data is sent and received

- TCP/IP
  - Exchanging packets of information over the Internet

- FTP
  - Exchanging files between computers

- SMTP
  - Exchanging email over the Internet

- POP
  - Exchanging email between mail reader and the ISP

- HTTP
  - Exchanging files over the WWW

- SSL
  - How information is to be encrypted
Software

• Program
  – Sequence of instruction that tells a computer what to do

• Execution
  – Performing the instruction sequence

• Programming language
  – Language for writing instructions to a computer

• Major flavors
  – Machine language or object code
  – Assembly language
  – High-level
Software

- **Program**
  - Sequence of instruction that tells a computer what to do

- **Execution**
  - Performing the instruction sequence

- **Programming language**
  - Language for writing instructions to a computer

- **Major flavors**
  - Machine language or object code
  - Assembly language
  - High-level

Program to which computer can respond directly. Each instruction is a binary code that corresponds to a native instruction.
Software

• Program
  – Sequence of instruction that tells a computer what to do

• Execution
  – Performing the instruction sequence

• Programming language
  – Language for writing instructions to a computer

• Major flavors
  – Machine language or object code
  – Assembly language
  – High-level

  Symbolic language for coding machine language instructions
Software

- **Program**
  - Sequence of instruction that tells a computer what to do

- **Execution**
  - Performing the instruction sequence

- **Programming language**
  - Language for writing instructions to a computer

- **Major flavors**
  - Machine language or object code
  - Assembly language
  - High-level

Detailed knowledge of the machine is not required. Uses a vocabulary and structure closer to the problem being solved.
Software

• Program
  – Sequence of instruction that tells a computer what to do

• Execution
  – Performing the instruction sequence

• Programming language
  – Language for writing instructions to a computer

• Major flavors
  – Machine language or object code
  – Assembly language
  – High-level
    
    Java is a high-level programming language
Software

- Program
  - Sequence of instruction that tells a computer what to do

- Execution
  - Performing the instruction sequence

- Programming language
  - Language for writing instructions to a computer

- Major flavors
  - Machine language or object code
  - Assembly language
  - High-level

For program to be executed it must be translated
Translation

- Translator
  - Accepts a program written in a source language and translates it to a program in a target language

- Compiler
  - Standard name for a translator whose source language is a high-level language

- Interpreter
  - A translator that both translates and executes a source program
Java translation

- Two-step process

- First step
  - Translation from Java to bytecodes
    - Bytecodes are architecturally neutral object code
    - Bytecodes are stored in a file with extension .class

- Second step
  - An interpreter translates the bytecodes into machine instructions and executes them
    - Interpreter is known a Java Virtual Machine or JVM
Task

- Display the forecast

I think there is a world market for maybe five computers.
   Thomas Watson, IBM, 1943.
Sample output

```
cmd: javac DisplayForecast.java

I think there is a world market for maybe five computers.
     Thomas Watson, IBM, 1943.

cmd: java DisplayForecast
I think there is a world market for maybe five computers.
     Thomas Watson, IBM, 1943.

cmd: java DisplayForecast
I think there is a world market for maybe five computers.
     Thomas Watson, IBM, 1943.

cmd:

cmd:
```
DisplayForecast.java

// Authors: J. P. Cohoon and J. W. Davidson
// Purpose: display a quotation in a console window

public class DisplayForecast {

    // method main(): application entry point
    public static void main(String[] args) {
        System.out.print("I think there is a world market for");
        System.out.println(" maybe five computers.");
        System.out.println(" Thomas Watson, IBM, 1943.");
    }
}
public class DisplayForecast {

    // method main(): application entry point
    public static void main(String[] args) {
        System.out.print("I think there is a world market for");
        System.out.println(" maybe five computers.");
        System.out.println(" Thomas Watson, IBM, 1943.");
    }
}

Three statements make up the action of method main()

Method main() is part of class DisplayForecast
public class DisplayForecast {

    // method main(): application entry point
    public static void main(String[] args) {
        System.out.print("I think there is a world market for");
        System.out.println(" maybe five computers.");
        System.out.println(" Thomas Watson, IBM, 1943.");
    }
}

A method is a named piece of code that performs some action or implements a behavior
DisplayForecast.java

// Authors: J. P. Cohoon and J. W. Davidson
// Purpose: display a quotation in a console window

public class DisplayForecast {

    // method main(): application entry point
    public static void main(String[] args) {
        System.out.print("I think there is a world market for");
        System.out.println(" maybe five computers.");
        System.out.println(" Thomas Watson, IBM, 1943.");
    }
}

An application program is required to have a public static void method named main().
Java and the Internet

Your machine

DisplayForecast.java

Java Compiler

DisplayForecast.class

Your friend's machine

I think ...

JVM

DisplayForecast.class

Modem

Internet

Modem
Engineering software

- Complexity of software grows as attempts are made to make it easier to use
  - Rise of wizards

[Graph showing the relationship between complexity and expertise needed]
Software engineering

• Goal
  – Production of software that is effective and reliable, understandable, cost effective, adaptable, and reusable
Software engineering

• Goal
  – Production of software that is effective and reliable, understandable, cost effective, adaptable, and reusable

• Work correctly and not fail
Software engineering

• Goal
  – Production of software that is effective and reliable, understandable, cost effective, adaptable, and reusable

• Because of the long lifetime many people will be involved
  – Creation
  – Debugging
  – Maintenance
  – Enhancement

• Two-thirds of the cost is typically beyond creation
Software engineering

• Goal
  – Production of software that is effective and reliable, understandable, cost effective, adaptable, and reusable

• Cost to develop and maintain should not exceed expected benefit
Software engineering

- **Goal**
  - Production of software that is effective and reliable, understandable, cost effective, adaptable, and reusable
  - Design software so that new features and capabilities can be added
Software engineering

• Goal
  – Production of software that is effective and reliable, understandable, cost effective, adaptable, and reusable

• Makes sense due to the great costs involved to have flexible components that can be used in other software
Principles

- Abstraction
- Encapsulation
- Modularity
- Hierarchy
Principles

- Abstraction
- Encapsulation
- Modularity
- Hierarchy

Determine the relevant properties and features while ignoring nonessential details
Principles

- Abstraction
- Encapsulation
- Modularity
- Hierarchy

Separate components into external and internal aspects
Principles

- Abstraction
- Encapsulation
- Modularity
- Hierarchy

Construct a system from components and packages
Principles

- Abstraction
- Encapsulation
- Modularity
- Hierarchy

Ranking or ordering of objects
Object-oriented design

- **Purpose**
  - Promote thinking about software in a way that models the way we think and interact with the physical world
  - Including specialization

- **Object**
  - Properties or attributes
  - Behaviors
Programming

- **Class**
  - Term for a type of software object

- **Object**
  - An instance of a class with
  - specific properties and attributes
Programming

- Problem solving through the use of a computer system

- Maxim
  - You cannot make a computer do something if you do not know how to do it yourself
Problem Solving

• Why do you care?
  – We are all assigned tasks to do
    • At work
    • At home
    • At school

• Why not do them
  – Right
  – Efficiently
Problem Solving

- Why care about computer-based problem solving (i.e., programming)?
  - Neat
  - Frontier of science
  - Profitable
  - Necessary
  - Quality of life
Problem Solving

• Remember
  – The goal is not a clever solution but a correct solution
Problem Solving

• Accept
  – The process is iterative
  • In solving the problem increased understanding might require restarting
Problem Solving

• Solutions
  – Often require both concrete and abstract thinking
• Teamwork
Problem Solving Process

- What is it?
Problem Solving Process

• What is it?
  – Analysis
  – Design
  – Implementation
  – Testing
Problem Solving Process

• What is it?
  – Analysis
  – Design
  – Implementation
  – Testing

Determine the inputs, outputs, and other components of the problem

Description should be sufficiently specific to allow you to solve the problem
Problem Solving Process

- What is it?
  - Analysis
  - Design
  - Implementation
  - Testing

Describe the components and associated processes for solving the problem

- Straightforward and flexible
- Method – process
- Object – component and associated methods
Problem Solving Process

- What is it?
  - Analysis
  - Design
  - Implementation
  - Testing

Develop solutions for the components and use those components to produce an overall solution

  Straightforward and flexible
Problem Solving Process

• What is it?
  – Analysis
  – Design
  – Implementation
  – Testing

Test the components individually and collectively
Problem Solving Process

Analysis

Design

Implementation

Testing

Determine problem features

Describe objects and methods

Produce the classes and code

Examine for correctness

Rethink as appropriate
Problem Solving Methodologies

• How to do it?
  – Depends upon your mode of thinking

• Bricolage approach

• Planned approach
Problem Solving Methodologies

- How to do it?
  - Depends upon your mode of thinking

- Bricolage approach

- Planned approach

Problem features and aspects are repeatedly tried and manipulated according to your *personal* way of organizing information

A mistake is not an error, but a correction waiting to be made in the natural course of solving the problem
Problem Solving Methodologies

- How to do it?
  - Depends upon your mode of thinking

- Bricolage approach

- Planned approach

Uses logic, formalism, and engineering coupled with a structured methodology

Inherent structure of the planned approach offers makes it easier to show correctness of solution

Dominant method in terms of teaching
Tips

- Find out as much as you can
- Reuse what has been done before
- Expect future reuse
- Break complex problems into subproblems
Tips

- Find out as much as you can
- Reuse what has been done before
- Expect future reuse
- Break complex problems into subproblems

Find out what is known about the problem

  Talk to the presenter

  Determine what attempts have succeeded and what attempts have failed
Tips

• Find out as much as you can

• Reuse what has been done before

• Expect future reuse

• Break complex problems into subproblems

Research can require significant time and generate questions

The effort is worthwhile because the result is a better understanding

True understanding of the problem makes it easier to solve
Tips

• Find out as much as you can
• Reuse what has been done before
• Expect future reuse
• Break complex problems into subproblems

Consider
  Sketching a solution and then repeatedly refine its components until the entire process is specified
Tips

- Find out as much as you can
- Reuse what has been done before
- Expect future reuse
- Break complex problems into subproblems

Your time is valuable
Correctness is probably even more valuable
Use existing infrastructure that is known to work
Tips

- Find out as much as you can
- Reuse what has been done before
- Expect future reuse
- Break complex problems into subproblems

Be open to *indirect* use of existing materials
Tips

• Find out as much as you can
• Reuse what has been done before
• Expect future reuse
• Break complex problems into subproblems

Make as few assumptions as necessary

Maximizes the likelihood that your effort can be used in future situations
Tips

- Find out as much as you can
- Reuse what has been done before
- Expect future reuse
- Break complex problems into subproblems

Divide-and-conquer

Solve subproblems and combine into an overall solution
Java basics
Programming

- Problem solving through the use of a computer system

- Maxim
  - You cannot make a computer do something if you do not know how to do it yourself
Software

- Program
  - Sequence of instruction that tells a computer what to do

- Execution
  - Performing the instruction sequence

- Programming language
  - Language for writing instructions to a computer

- Major flavors
  - Machine language or object code
  - Assembly language
  - High-level

  Program to which computer can respond directly. Each instruction is a binary code that corresponds to a native instruction.
Software

- Program
  - Sequence of instruction that tells a computer what to do

- Execution
  - Performing the instruction sequence

- Programming language
  - Language for writing instructions to a computer

- Major flavors
  - Machine language or object code
  - Assembly language
  - High-level

  Symbolic language for coding machine language instructions
Software

- Program
  - Sequence of instruction that tells a computer what to do

- Execution
  - Performing the instruction sequence

- Programming language
  - Language for writing instructions to a computer

- Major flavors
  - Machine language or object code
  - Assembly language
  - High-level

  Detailed knowledge of the machine is not required. Uses a vocabulary and structure closer to the problem being solved
Software

- Program
  - Sequence of instruction that tells a computer what to do

- Execution
  - Performing the instruction sequence

- Programming language
  - Language for writing instructions to a computer

- Major flavors
  - Machine language or object code
  - Assembly language
  - High-level

Java is a high-level programming language
Software

• Program
  – Sequence of instruction that tells a computer what to do

• Execution
  – Performing the instruction sequence

• Programming language
  – Language for writing instructions to a computer

• Major flavors
  – Machine language or object code
  – Assembly language
  – High-level

For program to be executed it must be translated
Translation

- Translator
  - Accepts a program written in a source language and translates it to a program in a target language

- Compiler
  - Standard name for a translator whose source language is a high-level language

- Interpreter
  - A translator that both translates and executes a source program
Java translation

- Two-step process

- First step
  - Translation from Java to bytecodes
    - Bytecodes are architecturally neutral object code
    - Bytecodes are stored in a file with extension .class

- Second step
  - An interpreter translates the bytecodes into machine instructions and executes them
    - Interpreter is known a Java Virtual Machine or JVM
Task

- Display the supposed forecast

  I think there is a world market for maybe five computers.
  Thomas Watson, IBM, 1943.
Sample output

```
Sample output

```

```
Sample output

```

```
Sample output

```

```
Sample output

```

```
Sample output

```

```
Sample output

```

```
Sample output

```

```
Sample output

```

```
Sample output

```

```
Sample output

```

```
Sample output

```

```
Sample output

```

```
Sample output

```

```
Sample output

```

```
public class DisplayForecast {

    // method main(): application entry point
    public static void main(String[] args) {
        System.out.println("I think there is a world market for");
        System.out.println("maybe five computers.");
        System.out.println("Thomas Watson, IBM, 1943.");
    }

    // Authors: J. P. Cohoon and J. W. Davidson
    // Purpose: display a quotation in a console window
}

DisplayForecast.java
public class DisplayForecast {

   // method main(): application entry point
   public static void main(String[] args) {
      System.out.print("I think there is a world market for");
      System.out.println(" maybe five computers.");
      System.out.println("   Thomas Watson, IBM, 1943.");
   }
}

Three statements make up the action of method main()
Method main() is part of class DisplayForecast
public class DisplayForecast {

    // method main(): application entry point
    public static void main(String[] args) {
        System.out.println("I think there is a world market for");
        System.out.println(" maybe five computers.");
        System.out.println(" Thomas Watson, IBM, 1943.");
    }
}

A method is a named piece of code that performs some action or implements a behavior
public class DisplayForecast {

   // method main(): application entry point
   public static void main(String[] args) {
      System.out.print("I think there is a world market for");
      System.out.println(" maybe five computers.");
      System.out.println(" Thomas Watson, IBM, 1943.");
   }
}

An application program is required to have a public static void method named main().
public class DisplayForecast {

    // method main(): application entry point
    public static void main(String[] args) {
        System.out.print("I think there is a world market for");
        System.out.println(" maybe five computers.");
        System.out.println(" Thomas Watson, IBM, 1943.");
    }
}

public, static, and void are keywords. They cannot be used as names

public means the method is shareable
DisplayForecast.java

// Authors: J. P. Cohoon and J. W. Davidson
// Purpose: display a quotation in a console window

public class DisplayForecast {

    // method main(): application entry point
    public static void main(String[] args) {
        System.out.print("I think there is a world market for");
        System.out.println(" maybe five computers.");
        System.out.println(" Thomas Watson, IBM, 1943.");
    }
}

Consider static and void later
DisplayForecast.java

// Authors: J. P. Cohoon and J. W. Davidson
// Purpose: display a quotation in a console window

public class DisplayForecast {

    // method main(): application entry point
    public static void main(String[] args) {
        System.out.print("I think there is a world market for");
        System.out.println(" maybe five computers.");
        System.out.println(" Thomas Watson, IBM, 1943.");
    }
}

Java allows a statement to be made up of multiple lines of text
Semicolons delimit one statement from the next
DisplayForecast.java

// Authors: J. P. Cohoon and J. W. Davidson
// Purpose: display a quotation in a console window

public class DisplayForecast {

    // method main(): application entry point
    public static void main(String[] args) {
        System.out.print("I think there is a world market for");
        System.out.println(" maybe five computers.");
        System.out.println(" Thomas Watson, IBM, 1943.");
    }
}

A class defines an object form. An object can have methods and attributes

Keyword class indicates a class definition follows
DisplayForecast.java

// Authors: J. P. Cohoon and J. W. Davidson
// Purpose: display a quotation in a console window

public class DisplayForecast {

    // method main(): application entry point
    public static void main(String[] args) {
        System.out.print("I think there is a world market for");
        System.out.println(" maybe five computers.");
        System.out.println("   Thomas Watson, IBM, 1943.");
    }
}

The class has a name
DisplayForecast.java

// Authors: J. P. Cohoon and J. W. Davidson
// Purpose: display a quotation in a console window

public class DisplayForecast {

    // method main(): application entry point
    public static void main(String[] args) {
        System.out.print("I think there is a world market for");
        System.out.println(" maybe five computers.");
        System.out.println(" Thomas Watson, IBM, 1943.");
    }
}

Programs are read by people – make sure they are readable.

Use whitespace, comments, and indentation to aid understanding
DisplayForecast.java

// Authors: J. P. Cohoon and J. W. Davidson
// Purpose: display a quotation in a console window

public class DisplayForecast {

    // method main(): application entry point
    public static void main(String[] args) {
        System.out.print("I think there is a world market for");
        System.out.println(" maybe five computers.");
        System.out.println("    Thomas Watson, IBM, 1943.");
    }
}

Whitespace separates program elements

Whitespace between program elements is ignored by Java
DisplayForecast.java

// Authors: J. P. Cohoon and J. W. Davidson
// Purpose: display a quotation in a console window

public class DisplayForecast {
    // method main(): application entry point
    public static void main(String[] args) {
        System.out.print("I think there is a world market for");
        System.out.println(" maybe five computers.");
        System.out.println(" Thomas Watson, IBM, 1943.");
    }
}

// indicates rest of the line is a comment

Comments are used to document authors, purpose, and program elements
public class DisplayForecast {

    // method main(): application entry point
    public static void main(String[] args) {
        System.out.print("I think there is a world market for");
        System.out.println(" maybe five computers.");
        System.out.println(" Thomas Watson, IBM, 1943.");
    }
}

Method main() is part of DisplayForecast
Statements are part of method main()

Indentation indicates subcomponents
Method main()

```java
public static void main(String[] args) {
    System.out.print("I think there is a world market for");
    System.out.println(" maybe five computers.");
    System.out.println(" Thomas Watson, IBM, 1943.");
}
```

- Class `System` supplies objects that can print and read values
- `System` variable `out` references the standard printing object
  - Known as the standard output stream
- Variable `out` provides access to printing methods
  - `print()`: displays a value
  - `println()`: displays a value and moves cursor to the next line
System.out

Variable System.out gives access to an output stream of type PrintStream.

The printing destination attribute for this PrintStream object is the console window.

The behaviors of a PrintStream object support a high-level view of printing.

System.out: PrintStream
- destination = ...
+ println(String s): void
+ print(String s): void
+ ...
Selection

The period indicates that we want to select an individual class member of System

The period want to select class member.
Method main()

public static void main(String[] args) {
    System.out.print("I think there is a world market for");
    System.out.println(" maybe five computers.");
    System.out.println(" Thomas Watson, IBM, 1943.");
}

- Method print() and println() both take a string parameter
  - The parameter specifies the value that is to be used in the invocation
Method main()

```java
public static void main(String[] args) {
    System.out.print("I think there is a world market for");
    System.out.println(" maybe five computers.");
    System.out.println(" Thomas Watson, IBM, 1943.");
}
```

- The `print()` statement starts the program output

I think there is a world market for
public static void main(String[] args) {
    System.out.print("I think there is a world market for");
    System.out.println(" maybe five computers.");
    System.out.println(" Thomas Watson, IBM, 1943.");
}

• The first println() statement completes the first line of output

I think there is a world market for maybe five computers
Method main()

```java
public static void main(String[] args) {
    System.out.print("I think there is a world market for");
    System.out.println(" maybe five computers.");
    System.out.println(" Thomas Watson, IBM, 1943.");
}
```

- The second println() statement starts and completes the second line of output

I think there is a world market for maybe five computers
Thomas Watson, IBM, 1943
public static void main(String[] args) {
    System.out.print("The real problem is not whether ");
    System.out.print("machines think but whether people ");
    System.out.println("do");
    System.out.println("-- B.F. Skinner (paraphrased)" );
}

• What does this method main() output?
Computation

• Programmers frequently write small programs for computing useful things

• Example – body mass index (BMI)
  – Measure of fitness
    • Ratio of person’s weight to the square of the person’s height
      – Weight in is kilograms, height is in meters
    • Person of interest is 4.5 feet and weighs 75.5 pounds

• Metric conversions
  – Kilograms per pound 0.454
  – Meters per foot 0.3046
Common program elements

- **Type**
  - Set of values along with operators that can manipulate and create values from the set

- **Primitive types support numeric, character, logical values**
  - double and float
    - Values with decimals
  - byte, short, int, long
    - Integers
  - char
    - Characters (considered numeric)
  - boolean
    - Logical values

- **Basic operators**
  - + addition
  - - subtraction
  - * multiplication
  - / division
Common program elements

- **Constant**
  - Symbolic name for memory location whose value does not change
    - KILOGRAMS_PER_POUND

- **Variable**
  - Symbolic name for memory location whose value can change
    - weightInPounds
Program outline for BMI.java

// Purpose: Compute BMI for given weight and height

public class BMI {

    // main(): application entry point
    public static void main(String[] args) {
        // define constants

        // set up person's characteristics

        // convert to metric equivalents

        // perform bmi calculation

        // display result
    }
}
public static void main(String[] args) {
    // define constants
    final double KILOGRAMS_PER_POUND = 0.454;
    final double METERS_PER_FOOT = 0.3046;

    // set up person's characteristics
    double weightInPounds = 75.5;  // our person's weight
    double heightInFeet = 4.5;    // our person's height

    // convert to metric equivalents
    double metricWeight = weightInPounds * KILOGRAMS_PER_POUND;
    double metricHeight = heightInFeet * METERS_PER_FOOT;

    // perform bmi calculation
    double bmi = metricWeight / (metricHeight * metricHeight);

    // display result
    System.out.println("A person with");
    System.out.println(" weight " + weightInPounds + " lbs");
    System.out.println(" height " + heightInFeet + " feet");
    System.out.println("has a BMI of " + Math.round(bmi));
}
public static void main(String[] args) {
    // define constants
    final double KILOGRAMS_PER_POUND = 0.454;
    final double METERS_PER_FOOT = 0.3046;

    // set up person's characteristics
    double weightInPounds = 75.5;  // our person’s weight
    double heightInFeet = 4.5;      // our person’s height

    // convert to metric equivalents
    double metricWeight = weightInPounds * KILOGRAMS_PER_POUND;
    double metricHeight = heightInFeet * METERS_PER_FOOT;

    // perform bmi calculation
    double bmi = metricWeight / (metricHeight * metricHeight);

    // display result
    System.out.println("A person with");
    System.out.println(" weight " + weightInPounds + " lbs");
    System.out.println(" height " + heightInFeet + " feet");
    System.out.println("has a BMI of " + Math.round(bmi));
}
public static void main(String[] args) {
    // define constants
    final double KILOGRAMS_PER_POUND = 0.454;
    final double METERS_PER_FOOT = 0.3046;

    // set up person's characteristics
    double weightInPounds = 75.5;  // our person’s weight
    double heightInFeet = 4.5;    // our person’s height

    // convert to metric equivalents
    double metricWeight = weightInPounds * KILOGRAMS_PER_POUND;
    double metricHeight = heightInFeet * METERS_PER_FOOT;

    // perform bmi calculation
    double bmi = metricWeight / (metricHeight * metricHeight);

    // display result
    System.out.println("A person with");
    System.out.println("  weight "+weightInPounds + " lbs");
    System.out.println("  height "+heightInFeet + " feet");
    System.out.println("has a BMI of "+Math.round(bmi));
}
public static void main(String[] args) {
    // define constants
    final double KILOGRAMS_PER_POUND = 0.454;
    final double METERS_PER_FOOT = 0.3046;

    // set up person's characteristics
    double weightInPounds = 75.5;  // our person’s weight
    double heightInFeet = 4.5;  // our person’s height

    // convert to metric equivalents
    double metricWeight = weightInPounds * KILOGRAMS_PER_POUND;
    double metricHeight = heightInFeet * METERS_PER_FOOT;

    // perform bmi calculation
    double bmi = metricWeight / (metricHeight * metricHeight);

    // display result
    System.out.println("A person with");
    System.out.println("  weight " + weightInPounds + " lbs");
    System.out.println("  height " + heightInFeet + " feet");
    System.out.println("has a BMI of " + Math.round(bmi));
}
public static void main(String[] args) {
    // define constants
    final double KILOGRAMS_PER_POUND = 0.454;
    final double METERS_PER_FOOT = 0.3046;

    // set up person's characteristics
    double weightInPounds = 75.5; // our person's weight
    double heightInFeet = 4.5; // our person's height

    // convert to metric equivalents
    double metricWeight = weightInPounds * KILOGRAMS_PER_POUND;
    double metricHeight = heightInFeet * METERS_PER_FOOT;

    // perform bmi calculation
    double bmi = metricWeight / (metricHeight * metricHeight);

    // display result
    System.out.println("A person with");
    System.out.println("  weight "+weightInPounds+" lbs");
    System.out.println("  height "+heightInFeet+" feet");
    System.out.println("has a BMI of "+Math.round(bmi));
}
public static void main(String[] args) {
    // define constants
    final double KILOGRAMS_PER_POUND = 0.454;
    final double METERS_PER_FOOT = 0.3046;

    // set up person's characteristics
    double weightInPounds = 75.5; // our person's weight
    double heightInFeet = 4.5; // our person's height

    // convert to metric equivalents
    double metricWeight = weightInPounds * KILOGRAMS_PER_POUND;
    double metricHeight = heightInFeet * METERS_PER_FOOT;

    // perform bmi calculation
    double bmi = metricWeight / (metricHeight * metricHeight);

    // display result
    System.out.println("A person with");
    System.out.println(" weight " + weightInPounds + " lbs");
    System.out.println(" height " + heightInFeet + " feet");
    System.out.println("has a BMI of " + Math.round(bmi));
}
public static void main(String[] args) {
    // define constants
    final double KILOGRAMS_PER_POUND = 0.454;
    final double METERS_PER_FOOT = 0.3046;

    // set up person's characteristics
    double weightInPounds = 75.5; // our person's weight
    double heightInFeet = 4.5; // our person's height

    // convert to metric equivalents
    double metricWeight = weightInPounds * KILOGRAMS_PER_POUND;
    double metricHeight = heightInFeet * METERS_PER_FOOT;

    // perform bmi calculation
    double bmi = metricWeight / (metricHeight * metricHeight);

    // display result
    System.out.println("A person with");
    System.out.println(" weight " + weightInPounds + " lbs");
    System.out.println(" height " + heightInFeet + " feet");
    System.out.println("has a BMI of " + Math.round(bmi));
}
public static void main(String[] args) {
    // define constants
    final double KILOGRAMS_PER_POUND = 0.454;
    final double METERS_PER_FOOT = 0.3046;

    // set up person's characteristics
    double weightInPounds = 75.5;  // our person's weight
    double heightInFeet = 4.5;     // our person's height

    // convert to metric equivalents
    double metricWeight = weightInPounds * KILOGRAMS_PER_POUND;
    double metricHeight = heightInFeet * METERS_PER_FOOT;

    // perform bmi calculation
    double bmi = metricWeight / (metricHeight * metricHeight);

    // display result
    System.out.println("A person with");
    System.out.println("  weight "+weightInPounds+" lbs");
    System.out.println("  height "+heightInFeet+" feet");
    System.out.println("has a BMI of "+ Math.round(bmi));
}
public static void main(String[] args) {
    // define constants
    final double KILOGRAMS_PER_POUND = 0.454;
    final double METERS_PER FOOT = 0.3046;
    // set up person's characteristics
    double weightInPounds = 75.5; // our person's weight
    double heightInFeet = 4.5; // our person's height
    // convert to metric equivalents
    double metricWeight = weightInPounds * KILOGRAMS_PER_POUND;
    double metricHeight = heightInFeet * METERS_PER FOOT;
    // perform bmi calculation
    double bmi = metricWeight / (metricHeight * metricHeight);
    // display result
    System.out.println("A person with");
    System.out.println(" weight " + weightInPounds + " lbs");
    System.out.println(" height " + heightInFeet + " feet");
    System.out.println("has a BMI of " + Math.round(bmi));
}
public static void main(String[] args) {
    // define constants
    final double KILOGRAMS_PER_POUND = 0.454;
    final double METERS_PER_FOOT = 0.3046;

    // set up person's characteristics
    double weightInPounds = 75.5;  // our person's weight
    double heightInFeet = 4.5;    // our person's height

    // convert to metric equivalents
    double metricWeight = weightInPounds * KILOGRAMS_PER_POUND;
    double metricHeight = heightInFeet * METERS_PER_FOOT;

    // perform bmi calculation
    double bmi = metricWeight / (metricHeight * metricHeight);

    // display result
    System.out.println("A person with");
    System.out.println("  weight " + weightInPounds + " lbs");
    System.out.println("  height " + heightInFeet + " feet");
    System.out.println("has a BMI of " + Math.round(bmi));
}
// Purpose: Convert a Celsius temperature to Fahrenheit

public class CelsiusToFahrenheit {

    // main(): application entry point
    public static void main(String[] args) {
        // set Celsius temperature of interest
        int celsius = 28;

        // convert to Fahrenheit equivalent
        int fahrenheit = 32 + ((9 * celsius) / 5);

        // display result
        System.out.println("Celsius temperature");
        System.out.println("   " + celsius);
        System.out.println("equals Fahrenheit temperature");
        System.out.println("   " + fahrenheit);
    }
}

// Purpose: Demonstrate char arithmetic

public class LowerToUpper {

    // main(): application entry point
    public static void main(String[] args) {
        // set lower case character of interest
        char lowerCaseLetter = 'c';

        // convert to uppercase equivalent
        char upperCaseLetter = 'A' + (lowerCaseLetter - 'a');

        // display result
        System.out.println("Uppercase equivalent of ");
        System.out.println(" " + lowerCaseLetter);
        System.out.println("is ");
        System.out.println(" " + upperCaseLetter);
    }
}

Expressions

- What is the value used to initialize expression
  ```java
  int expression = 4 + 2 * 5;
  ```

- What value is displayed
  ```java
  System.out.println(5 / 2.0);
  ```

- Java rules in a nutshell
  - Each operator has a precedence level and an associativity
    - Operators with higher precedence are done first
      - * and / have higher precedence than + and -
    - Associativity indicates how to handle ties
  - When floating-point is used the result is floating point
Question

- Does the following statement compute the average of double variables a, b, and c? Why

  double average = a + b + c / 3.0;
Interactive programs

- Programs that interact with their users through statements performing input and output
- BMI.java
  - Not interactive – weight and height are fixed
Support for interactive console programs

- Variable System.in
  - Associated with the standard input stream – the keyboard
- Class Scanner
  - Supports extraction of an input as a numbers, characters, and strings

Scanner stdin = Scanner.create(System.in);

Variable stdin gives access to an input stream that supports the extraction (reading) of input as values.
The input destination attribute for this Scanner object is stream associated with standard input—System.in.
The behaviors of a Scanner object support a high-level view of extracting values.
Accessing the standard input stream

- Set up

  ```java
  Scanner stdin = Scanner.create(System.in);
  ```

  The method returns a reference to a new Scanner object. This object is built using out of the standard input stream.
• Interactive program for bmi

Program outline

Purpose: Compute BMI for user-specified weight and height

public static void main(String[] args) throws IOException {
    final double KILOGRAMS_PER_POUND = 0.454;
    final double METERS_PER_FOOT = 0.3046;

    System.out.println("BMI Calculator\n");
    Scanner stdin = Scanner.create(System.in);
    System.out.print("Enter weight (lbs): ");
    double weight = stdin.nextDouble();
    System.out.print("Enter height (feet): ");
    double height = stdin.nextDouble();

    double metricWeight = weight * KILOGRAMS_PER_POUND;
    double metricHeight = height * METERS_PER_FOOT;

    double bmi = metricWeight / (metricHeight * metricHeight);

    System.out.println("A person with");
    System.out.println(" weight " + weight + " (lbs)" );
    System.out.println(" height " + height + " (feet)" );
    System.out.println("has a BMI of " + bmi);
}
Accessing the standard input stream

- Extraction
  
  ```java
  System.out.print("Enter weight (lbs): ");
  double weight = stdin.nextDouble();

  System.out.print("Enter height (feet): ");
  double height = stdin.nextDouble();
  ```
Primitive variable assignment

- Assignment operator =
  - Allows the memory location for a variable to be updated

```
target = expression;
```

- Consider
  ```
  int j = 11;
  j = 1985;
  ```
Primitive variable assignment

- Assignment operator =
  - Allows the memory location for a variable to be updated

\[ \text{target} = \text{expression}; \]

- Consider
  \[
  \text{int } j = 11; \\
  j = 1985;
  \]
Primitive variable assignment

- Consider
  ```java
  int a = 1;
  int aSquared = a * a;
  a = 5;
  aSquared = a * a;
  ```

- Consider
  ```java
  int i = 0;
  i = i + 1;
  ```

- Consider
  ```java
  int asaRating;
  asaRating = 400;
  ```
Primitive variable assignment

- Consider
  
  ```
  int a = 1;
  int aSquared = a * a;
  a = 5;
  aSquared = a * a;
  ```

- Consider
  
  ```
  int i = 0;
  i = i + 1;
  ```

- Consider
  
  ```
  int asaRating;
  asaRating = 400;
  ```
Primitive variable assignment

• Consider
  ```
  int a = 1;
  int aSquared = a * a;
  a = 5;
  aSquared = a * a;
  ```

• Consider
  ```
  int i = 0;
  i = i + 1;
  ```

• Consider
  ```
  int asaRating;
  asaRating = 400;
  ```
Primitive variable assignment

- Consider
  
  ```java
  int a = 1;
  int aSquared = a * a;
  a = 5;
  aSquared = a * a;
  ```

- Consider
  
  ```java
  int i = 0;
  i = i + 1;
  ```

- Consider
  
  ```java
  int asaRating;
  asaRating = 400;
  ```
Primitive variable assignment

• Consider
  ```
  int a = 1;
  int aSquared = a * a;
  a = 5;
  aSquared = a * a;
  ```

• Consider
  ```
  int i = 0;
  i = i + 1;
  ```

• Consider
  ```
  int asaRating;
  asaRating = 400;
  ```
Primitive variable assignment

- Consider
  ```java
  int a = 1;
  int aSquared = a * a;
  a = 5;
  aSquared = a * a;
  ```

- Consider
  ```java
  int i = 0;
  i = i + 1;
  ```

- Consider
  ```java
  int asaRating;
  asaRating = 400;
  ```
Primitive variable assignment

- Consider
  ```java
  int a = 1;
  int aSquared = a * a;
  a = 5;
  aSquared = a * a;
  ```

- Consider
  ```java
  int i = 0;
  i = i + 1;
  ```

- Consider
  ```java
  int asaRating;
  asaRating = 400;
  ```
Primitive variable assignment

- Consider
  
  ```java
double x = 5.12;
double y = 19.28;
double rememberX = x;
x = y;
y = rememberX;
```
Primitive variable assignment

Consider

```java
double x = 5.12;
double y = 19.28;
double rememberX = x;
x = y;
y = rememberX;
```

\[ \begin{align*}
\text{x} & : 5.12 \\
\text{y} & : 19.28
\end{align*} \]
Primitive variable assignment

- Consider
  ```java
  double x = 5.12;
  double y = 19.28;
  double rememberX = x;
  x = y;
  y = rememberX;
  ```

<table>
<thead>
<tr>
<th>x</th>
<th>y</th>
<th>rememberX</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.12</td>
<td>19.28</td>
<td>5.12</td>
</tr>
</tbody>
</table>
Primitive variable assignment

- Consider
  
  ```java
double x = 5.12;
double y = 19.28;
double rememberX = x;
x = y;
y = rememberX;
```

```plaintext
<table>
<thead>
<tr>
<th>x</th>
<th>19.28</th>
</tr>
</thead>
<tbody>
<tr>
<td>y</td>
<td>19.28</td>
</tr>
<tr>
<td>rememberX</td>
<td>5.12</td>
</tr>
</tbody>
</table>
```
Primitive variable assignment

- Consider
  ```
  double x = 5.12;
  double y = 19.28;
  double rememberX = x;
  x = y;
  y = rememberX;
  ```

  ![Table showing variable assignments](image)
Increment and decrement operators

- `++`
  - Increments a number variable by 1
- `--`
  - Decrements a numeric variable by 1

Consider
```
int i = 4;
++i;
System.out.println(i);
System.out.println(++i);
System.out.println(i++);
System.out.println(i);
```
Increment and decrement operators

- **++**
  - Increments a number variable by 1

- **--**
  - Decrements a numeric variable by 1

Consider

```java
int i = 4;               // define
++i;
++i;
System.out.println(i);
System.out.println(;++i);
System.out.println(i++);
System.out.println(i);
```
Increment and decrement operators

- `++` 
  - Increments a number variable by 1
- `--` 
  - Decrements a numeric variable by 1

Consider

```java
int i = 4;
++i; // increment
System.out.println(i);
System.out.println(++i);
System.out.println(i++);
System.out.println(i);
```
Increment and decrement operators

- ++
  - Increments a number variable by 1
- --
  - Decrements a numeric variable by 1

Consider

```java
int i = 4;
++i;
System.out.println(i);  // display
System.out.print(++i);
System.out.println(i++);
System.out.println(i);
```
Increment and decrement operators

- **++**
  - Increments a number variable by 1
- **--**
  - Decrements a numeric variable by 1

Consider

```java
int i = 4;
++i;
System.out.println(i);
System.out.print(++i);  // update then display
System.out.println(i++);
System.out.println(i);
```

```java
6
```
Increment and decrement operators

- `++`
  - Increments a number variable by 1
- `--`
  - Decrements a numeric variable by 1

Consider

```java
int i = 4;
++i;
System.out.println(i);
System.out.println(++i);
System.out.println(i++); // display then update
System.out.println(i);
```
Increment and decrement operators

- `++` - Increments a number variable by 1
- `--` - Decrements a numeric variable by 1

Consider

```java
int i = 4;
++i;
System.out.println(i);
System.out.println(++i);
System.out.println(i++);
System.out.println(i);  // display
```
Escape sequences

- Java provides escape sequences for printing special characters
  - \b backspace
  - \n newline
  - \t tab
  - \r carriage return
  - \ \ backslash
  - \" double quote
  - \' single quote
Escape sequences

• What do these statements output?

System.out.println("Person\tHeight\tShoe size");
System.out.println("=========================");
System.out.println("Hannah\t5'1"\t7");
System.out.println("Jenna\t5'10"\t9");
System.out.println("JJ\t6'1"\t14");

• Output

Person   Height   Shoe size
========================
Hannah   5‘1"     7
Jenna    5'10"    9
JJ        6'1"    14
Objects
Getting classy

- Your current job
  - Gain experience creating and manipulating objects from the standard Java types

- Why
  - Prepares you for defining your own classes and creating and manipulating the objects of those classes
Values versus objects

- Numbers
  - Have values but they do not have behaviors

- Objects
  - Have attributes and behaviors

- System.in
  - References an InputStream
    - Attribute: keyboard
    - Behaviors: reading

- System.out
  - References an OutputStream
    - Attribute: monitor
    - Behaviors: printing
Some other Java object types

- Scanner
- String
- Rectangle
- Color
- JFrame
Consider

• Statements
  
  ```java
  int peasPerPod = 8;
  String message = "Don't look behind the door!"
  ```

• How show we represent these definitions according to the notions of Java?
The value of String variable message is a reference to a String object representing the character string "Don't look behind the door!"

The value of primitive int variable peasPerPod is 8

<table>
<thead>
<tr>
<th>String</th>
</tr>
</thead>
<tbody>
<tr>
<td>- text = &quot;Don't look behind the door!&quot;</td>
</tr>
<tr>
<td>- length = 27</td>
</tr>
<tr>
<td>- ...</td>
</tr>
<tr>
<td>+ length() : int</td>
</tr>
<tr>
<td>+ charAt(int i) : char</td>
</tr>
<tr>
<td>+ subString(int m, int n) String</td>
</tr>
<tr>
<td>+ indexOf(String s, int m) : int</td>
</tr>
<tr>
<td>+ ...</td>
</tr>
</tbody>
</table>
Shorthand representation

peasPerPod 8
message "Don't look behind the door!"
Examples

• Consider
  
  String a = "excellence";
  String b = a;

• What is the representation?
Examples

• Consider
  
  ```java
  String a = "excellence";
  String b = a;
  ```

• What is the representation?
Uninitialized versus null

• Consider
  
  String dayOfWeek;
  Scanner inStream;

• What is the representation?
Uninitialized versus null

- Consider
  ```java
  String dayOfWeek;
  Scanner inStream;
  ```

- What is the representation?

  ![Uninitialized objects](null)

  ```java
  dayOfWeek = -
  inStream = -
  ```
Uninitialized versus null

- Consider
  
  ```java
  String fontName = null;
  Scanner fileStream = null;
  ```

- What is the representation?
Uninitialized versus null

• Consider
  
  ```java
  String fontName = null;
  Scanner fileStream = null;
  ```

• What is the representation?

<table>
<thead>
<tr>
<th>fontName</th>
<th>null</th>
</tr>
</thead>
<tbody>
<tr>
<td>fileStream</td>
<td>null</td>
</tr>
</tbody>
</table>
Assignment

- Consider
  String word1 = "luminous";
  String word2 = "graceful";
  Word1 = word2;

- Initial representation
Assignment

- Consider
  ```java
  String word1 = "luminous";
  String word2 = "graceful";
  Word1 = word2;
  ```

- After assignment

   ![Diagram showing assignment of word2 to word1]
Using objects

- Consider

```java
Scanner stdin = Scanner.create(System.in);

System.out.print("Enter your account name: ");
String response = stdin.nextLine();
```
Using objects

- Consider

```java
Scanner stdin = Scanner.create(System.in);

System.out.print("Enter your account name: ");
String response = stdin.nextLine();
```
Using objects

• Consider
  
  ```java
  Scanner stdin = Scanner.create(System.in);
  System.out.print("Enter your account name: ");
  String response = stdin.nextLine();
  ```

• Suppose the user interaction is
  
  Enter your account name: artiste
String representation

- Consider
  - String alphabet = "abcdefghijklmnopqrstuvwxyz";

- Standard shorthand representation

- Truer representation
String representation

- Consider
  - String alphabet = "abcdefghijklmnopqrstuvwxyz";
  - char c1 = alphabet.charAt(9);
  - char c2 = alphabet.charAt(15);
  - char c3 = alphabet.charAt(2);

- What are the values of c1, c2, and c3? Why?

<table>
<thead>
<tr>
<th>c1</th>
<th>c2</th>
<th>c3</th>
</tr>
</thead>
<tbody>
<tr>
<td>'j'</td>
<td>'p'</td>
<td>'c'</td>
</tr>
</tbody>
</table>
public class WordLength {
    public static void main(String[] args) {

        Scanner stdin = Scanner.create(System.in);

        System.out.print("Enter a word: ");
        String word = stdin.readLine();
        int wordLength = word.length();
        System.out.println("Word " + word + " has length "+ wordLength + ".");
    }
}
More String methods

• Consider
  ```java
  String weddingDate = "August 21, 1976";
  String month = weddingDate.substring(0, 6);
  System.out.println("Month is " + month + ".");
  ```

• What is the output?
More String methods

• Consider
  
  String weddingDate = "August 21, 1976";
  String month = weddingDate.substring(0, 6);
  System.out.println("Month is " + month + ".");

• What is the output?
  Month is August.
More String methods

• Consider

    String fruit = "banana";
    String searchString = "an";
    int n1 = fruit.indexOf(searchString, 0);
    int n2 = fruit.indexOf(searchString, n1 + 1);
    int n3 = fruit.indexOf(searchString, n2 + 1);

    System.out.println("First search: " + n1);
    System.out.println("Second search: " + n2);
    System.out.println("Third search: " + n3);

• What is the output?
More String methods

- Consider
  ```java
  String fruit = "banana";
  String searchString = "an";
  int n1 = fruit.indexOf(searchString, 0);
  int n2 = fruit.indexOf(searchString, n1 + 1);
  int n3 = fruit.indexOf(searchString, n2 + 1);
  System.out.println("First search: " + n1);
  System.out.println("Second search: " + n2);
  System.out.println("Third search: " + n3);
  ```

- What is the output?
  First search: 1
  Second search: 3
  Third search: -1
More String methods

- Consider
  ```java
  int v1 = -12;
  double v2 = 3.14;
  char v3 = 'a';
  String s1 = String.valueOf(v1);
  String s2 = String.valueOf(v2);
  String s3 = String.valueOf(v3);
  ```
Final variables

- Consider
  ```java
  final String POEM_TITLE = "Appearance of Brown";
  final String WARNING = "Weather ball is black";
  ```

- What is the representation?

The locks indicate the memory locations hold constants.
Final variables

In general, these attributes can be modified through member methods.

The reference cannot be modified once it is established.
Rectangle

```java
int x = 3;
int y = 4;
int width = 5;
int height = 2;
Rectangle r = new Rectangle(x,
```

The first two parameters of the Rectangle constructor specify the position of the upper-left-hand corner of the new Rectangle.

The third and fourth parameters of the Rectangle constructor specify the dimensions of the new Rectangle.
Rectangle

The first two parameters of the Rectangle constructor specify the position of the upper-left-hand corner of the new Rectangle.

The third and fourth parameters of the Rectangle constructor specify the dimensions of the new Rectangle.

```java
int x = 3;
int y = 4;
int width = 5;
int height = 2;
Rectangle r = new Rectangle(x, y, width, height);
```
Consider:

```java
final Rectangle BLOCK = new Rectangle(6, 9, 4, 2);
BLOCK.setLocation(1, 4);
BLOCK.resize(8, 3);
```
Consider

```java
final Rectangle BLOCK = new Rectangle(6, 9, 4, 2);
BLOCK.setLocation(1, 4);
BLOCK.resize(8, 3);
```
Final variables

- Consider

```java
final String LANGUAGE = "Java";
```

The reference cannot be modified once it is established. The contents are immutable because there are no String methods that allow the contents to be changed.
Classes
Preparation

- Scene so far has been background material and experience
  - Computing systems and problem solving
  - Variables
  - Types
  - Input and output
  - Expressions
  - Assignments
  - Objects
  - Standard classes and methods
Ready

- Experience what Java is really about
  - Design and implement objects representing information and physical world objects
Object-oriented programming

• Basis
  – Create and manipulate objects with attributes and behaviors that the programmer can specify

• Mechanism
  – Classes

• Benefits
  – An information type is design and implemented once
    • Reused as needed
      – No need reanalysis and re-justification of the representation
First class – ColoredRectangle

- Purpose
  - Introduce the basics of object design and implementation
  - Represent a colored rectangle in a window
Background

- **JFrame**
  - Principal Java class for representing a titled, bordered graphical window.
  
  - Standard class
    - Part of the swing library

```java
import javax.swing.*;
```

Example

- Consider
  
  ```java
  JFrame w1 = new JFrame("Bigger");
  JFrame w2 = new JFrame("Smaller");
  w1.setSize(200, 125);
  w2.setSize(150, 100);
  w1.setVisible(true);
  w2.setVisible(true);
  ```
Example

• Consider

    JFrame w1 = new JFrame("Bigger");
    JFrame w2 = new JFrame("Smaller");
    w1.setSize(200, 125);
    w2.setSize(150, 100);
    w1.setVisible(true);
    w2.setVisible(true);
Example

- Consider
  ```java
  JFrame w1 = new JFrame("Bigger");
  JFrame w2 = new JFrame("Smaller");
  w1.setSize(200, 125);
  w2.setSize(150, 100);
  w1.setVisible(true);
  w2.setVisible(true);
  ```
Example

- Consider
  
  ```java
  JFrame w1 = new JFrame("Bigger");
  JFrame w2 = new JFrame("Smaller");
  w1.setSize(200, 125);
  w2.setSize(150, 100);
  w1.setVisible(true);
  w2.setVisible(true);
  ```
Class ColoredRectangle – initial version

- Purpose
  - Support the display of square window containing a blue filled-in rectangle
    - Window has side length of 200 pixels
    - Rectangle is 40 pixels wide and 20 pixels high
    - Upper left hand corner of rectangle is at (80, 90)
  - Limitations are temporary
    - Remember BMI.java preceded BMICalculator.java
    - Lots of concepts to introduce
ColoredRectangle in action

• Consider

   ColoredRectangle r1 = new ColoredRectangle();
   ColoredRectangle r2 = new ColoredRectangle();

   System.out.println("Enter when ready");
   System.in.read();

   r1.paint(); // draw the window associated with r1
   r2.paint(); // draw the window associated with r2
ColoredRectangle in action

- Consider

```java
ColoredRectangle r1 = new ColoredRectangle();
ColoredRectangle r2 = new ColoredRectangle();

System.out.println("Enter when ready");
System.in.read();

r1.paint();  // draw the window associated with r1
r2.paint();  // draw the window associated with r2
```
ColoredRectangle in action

- Consider
  
  ```java
  ColoredRectangle r1 = new ColoredRectangle();
  ColoredRectangle r2 = new ColoredRectangle();

  System.out.println("Enter when ready");
  System.in.read();

  r1.paint(); // draw the window associated with r1
  r2.paint(); // draw the window associated with r2
  ```
ColoredRectangle in action

- Consider

```java
ColoredRectangle r1 = new ColoredRectangle();
ColoredRectangle r2 = new ColoredRectangle();

System.out.println("Enter when ready");
System.in.read();

r1.paint(); // draw the window associated with r1
r2.paint(); // draw the window associated with r2
```

The messages instruct the objects to display themselves.
import javax.swing.*;
import java.awt.*;

public class ColoredRectangle {
  // instance variables for holding object attributes
  private int width;
  private int height;
  private int x;
  private int y;
  private JFrame window;
  private Color color;

  // ColoredRectangle(): default constructor
  public ColoredRectangle() { // ...
  }

  // paint(): display the rectangle in its window
  public void paint() { // ...
  }
}
Instance variables and attributes

• Data field
  – Java term for an object attribute

• Instance variable
  – Symbolic name for a data field
  – Usually has private access
    • Assists in information hiding by encapsulating the object’s attributes
  – Default initialization
    • Numeric instance variables initialized to 0
    • Logical instance variables initialized to false
    • Object instance variables initialized to null
public class ColoredRectangle {

// instance variables for holding object attributes
private int width;
private int x;
private int height;
private int y;
private JFrame window;
private Color color;

// ColoredRectangle(): default constructor
public ColoredRectangle() {
    window = new JFrame("Box Fun");
    window.setSize(200, 200);
    width = 40;
    x = 80;
    height = 20;
    y = 90;
    color = Color.BLUE;
    window.setVisible(true);
}

// paint(): display the rectangle in its window
public void paint(Graphics g) {
    g.setColor(color);
    g.fillRect(x, y, width, height);
}

}
public class ColoredRectangle {

    // instance variables for holding object attributes
    private int width;                  private int x;
    private int height;                private int y;
    private JFrame window;             private Color color;

    // ColoredRectangle(): default constructor
    public ColoredRectangle() {
        window = new JFrame("Box Fun");
        window.setSize(200, 200);
        width = 40;            x = 80;
        height = 20;          y = 90;
        color = Color.BLUE;
        window.setVisible(true);
    }

    // paint(): display the rectangle in its window
    public void paint() {
        Graphics g = window.getGraphics();
        g.setColor(color);
        g.fillRect(x, y, width, height);
    }
}

ColoredRectangle default constructor

```java
public class ColoredRectangle {
    // instance variables to describe object attributes
    ...

    // ColoredRectangle(): default constructor
    public ColoredRectangle() {
        ...
        The name of a constructor always matches the name of its class
    }

    ...
    A constructor does not list its return type. A constructor always returns a reference to a new object of its class
}
```
public class ColoredRectangle {

    // instance variables for holding object attributes
    private int width;     private int x;
    private int height;    private int y;
    private JFrame window; private Color color;

    // ColoredRectangle(): default constructor
    public ColoredRectangle() {
        window = new JFrame("Box Fun");
        window.setSize(200, 200);
        width = 40;      x = 80;
        height = 20;    y = 90;
        color = Color.BLUE;
        window.setVisible(true);
    }

    // paint(): display the rectangle in its window
    public void paint() {
        Graphics g = window.getGraphics();
        g.setColor(color);
        g.fillRect(x, y, width, height);
    }
}
Color constants

- Color.BLACK
- Color.BLUE
- Color.CYAN
- Color.DARK_GRAY
- Color.GRAY
- Color.GREEN
- Color.LIGHT_GRAY
- Color.MAGENTA
- Color.ORANGE
- Color.PINK
- Color.RED
- Color.WHITE
- Color.YELLOW
The value of a ColoredRectangle variable is a reference to a ColoredRectangle object.

```java
ColoredRectangle r = new ColoredRectangle();
```

```
ColorRectangle
- width = 40
- height = 20
- x = 80
- y = 90
- window =
- color =

+ paint() : void
```

```
String
- text = "Box Fun"
- ...

+ length() : int
+ ...
```

```
Color
- color =
- ...

+ brighter() : Color
+ ...
```

```
JFrame
- width = 200
- height = 200
- title =
- ...

+ setVisible(boolean status) : void
+ ...
```
public class ColoredRectangle {

    // instance variables for holding object attributes
    private int width;     private int x;
    private int height;    private int y;
    private JFrame window; private Color color;

   // ColoredRectangle(): default constructor
    public ColoredRectangle() {
        window = new JFrame("Box Fun");
        window.setSize(200, 200);
        width = 40;       x = 80;
        height = 20;      y = 90;
        color = Color.BLUE;
        window.setVisible(true);
    }

    // paint(): display the rectangle in its window
    public void paint() {
        Graphics g = window.getGraphics();
        g.setColor(color);
        g.fillRect(x, y, width, height);
    }
}

Graphical context

- Graphics
  - Defined in java.awt.Graphics
  - Represents the information for a rendering request
    - Color
    - Component
    - Font
    - ...
  - Provides methods
- Text drawing
  - Line drawing
  - Shape drawing
    - Rectangles
    - Ovals
    - Polygons
public class ColoredRectangle {

    // instance variables for holding object attributes
    private int width;     private int x;
    private int height;    private int y;
    private JFrame window; private Color color;

    // ColoredRectangle(): default constructor
    public ColoredRectangle() {
        window = new JFrame("Box Fun");
        window.setSize(200, 200);
        width = 40;      x = 80;
        height = 20;    y = 90;
        color = Color.BLUE;
        window.setVisible(true);
    }

    // paint(): display the rectangle in its window
    public void paint() {
        Graphics g = window.getGraphics();
        g.setColor(color);
        g.fillRect(x, y, width, height);
    }

}
Method invocation

- Consider
  
  ```java
  r1.paint(); // display window associated with r1
  r2.paint(); // display window associated with r2
  ```

- Observe
  
  - When an instance method is being executed, the attributes of the object associated with the invocation are accessed and manipulated
  
  - Important that you understand what object is being manipulated
Method invocation

```java
public class ColoredRectangle {
    // instance variables to describe object attributes
    ...

    // paint(): display the rectangle in its window
    public void paint() {
        window.setVisible(true);
        Graphics g = window.getGraphics();
        g.setColor(color);
        g.fillRect(x, y, width, height);
    }
    ...
}
```

Instance variable window references the JFrame attribute of the object that caused the invocation. That is, the invocation `r1.paint()` causes the window attribute of the ColoredRectangle referenced by `r1` to be accessed. Similarly, the invocation `r2.paint()` causes the window attribute of the ColoredRectangle referenced by `r2` to be accessed.

The values of these instance variables are also from the ColoredRectangle object that invoked method `paint()`.
Improving ColoredRectangle

- Analysis
  - A ColoredRectangle object should
    - Be able to have any color
    - Be positionable anywhere within its window
    - Have no restrictions on its width and height
    - Accessible attributes
    - Updateable attributes
Improving ColoredRectangle

- Additional constructions and behaviors
  - Specific construction
    - Construct a rectangle representation using supplied values for its attributes
  - Accessors
    - Supply the values of the attributes
    - Individual methods for providing the width, height, x-coordinate position, y-coordinate position, color, or window of the associated rectangle
  - Mutators
    - Manage requests for changing attributes
    - Ensure objects always have sensible values
    - Individual methods for setting the width, height, x-coordinate position, y-coordinate position, color, or window of the associated rectangle to a given value
A mutator method

- **Definition**
  
  ```java
  // setWidth(): width mutator
  public void setWidth(int w) {
      width = w;
  }
  ```

- **Usage**

  ```java
  ColoredRectangle s = new ColoredRectangle();
  s.setWidth(80);
  ```

  *Object to be manipulated is the one referenced by s*
  *Initial value of the formal parameter comes from the actual parameter*
  *Changes to the formal parameter do not affect the actual parameter*
Mutator `setWidth()` evaluation

```java
ColoredRectangle s = new ColoredRectangle();
s.setWidth(80);
```

The invocation sends a message to the `ColoredRectangle` referenced by `s` to modify its width attribute. To do so, there is a temporary transfer of flow of control to `setWidth()`. The value of the actual parameter is 80.

```java
public class ColoredRectangle {
    ...
    // setWidth(): width mutator
    public void setWidth(int w) {
        width = w;
    }
    ...
}
```

For this invocation of method `setWidth()`, `w` is initialized to 80. The object being referenced within the method body is the object referenced by `s`.

Method `setWidth()` sets the instance variable `width` of its `ColoredRectangle`. For this invocation, `width` is set to 80 and the `ColoredRectangle` is the one referenced by `s`.

Method `setWidth()` is completed. Control is transferred back to the statement that invoked `setWidth()`.
Subtleties

• Consider

```java
ColoredRectangle r = new ColoredRectangle();
r.paint();
r.setWidth(80);
r.paint();
r.paint();
```

• What is the width is the rectangle on the screen after the mutator executes?
Other mutators

public void setHeight(int h) {
    height = h;
}

public void setX(int ulx) {
    x = ulx;
}

public void setY(int uly) {
    y = uly;
}

public void setWindow(JFrame f) {
    window = f;
}

public void setColor(Color c) {
    color = c;
}
Mutator usage

```java
coloredrectangle u = new coloredrectangle();
coloredrectangle v = new coloredrectangle();
u.setHeight(100);
u.setColor(Color.PINK);
v.setX(25);
v.setY(50);
JFrame display = new JFrame("Fun");
v.setWindow(display);
```

Sends a message to `u`'s ColoredRectangle to modify its height attribute to 100

Sends a message to `u`'s ColoredRectangle to modify its color attribute to pink

Sends a message to `v`'s ColoredRectangle to modify its x-axis position to 25

Sends a message to `v`'s ColoredRectangle to modify its y-axis position to 50

Sends a message to `v`'s ColoredRectangle to modify its window attribute to display's JFrame
Accessors

- Properties
  - Do not require parameters
  - Each accessor execution produces a return value
    - Return value is the value of the invocation

The method return type precedes the name of the method in the method definition.

```java
public int getWidth() {
    return width;
}
```

For method `getWidth()`, the return value is the value of the width attribute for the ColoredRectangle associated with the invocation. In invocation `t.getWidth()`, the return value is the value of the instance variable `width` for the ColoredRectangle referenced by `t`. 
Accessor usage

```java
ColoredRectangle t = new ColoredRectangle();
int w = t.getWidth();
```

Invocation sends a message to the ColoredRectangle referenced by `t` to return the value of its width. To do so, there is a temporary transfer of flow of control to `getWidth()`.

```java
public class ColoredRectangle {
  ...
  // getWidth(): accessor
  public int getWidth() {
    return width;
  }
  ...
}
```

Method `getWidth()` starts executing. For this invocation, the object being referenced is the object referenced by `t`.

The return expression evaluates to 40 (the width attribute of the ColoredRectangle object referenced by `t`).

Method completes by supplying its return value (40) to the invoking statement. Also, invoking statement regains the flow of control. From there variable `w` is initialized with the return value of the invocation.
Specific construction

public ColoredRectangle(int w, int h, int ulx, int uly, 
                        JFrame f, Color c) {
    setWidth(w);
    setHeight(h);
    setX(ulx);
    setY(uly);
    setWindow(f);
    setColor(c);
}

• Requires values for each of the attributes
  JFrame display = new JFrame("Even more fun");
  display.setSize(400, 400);
  ColoredRectangle w = new ColoredRectangle(60, 80, 
                                           20, 20, display, Color.YELLOW);
Specific construction

```java
public ColoredRectangle(int w, int h, int ulx, int uly, JFrame f, Color c) {
    setWidth(w);
    setHeight(h);
    setX(ulx);
    setY(uly);
    setWindow(f);
    setColor(c);
}
```

- Advantages to using mutators
  - Readability
  - Less error prone
  - Facilitates enhancements through localization
import java.io.*;
import java.awt.*;

public class SeeingDouble {
    public static void main(String[] args)
        throws IOException {
        ColoredRectangle r = new ColoredRectangle();
        System.out.println("Enter when ready");
        System.in.read();
        r.paint();
        r.setY(50);
        r.setColor(Color.RED); r.paint();
    }
}
Seeing double
Decisions
Background

- Our problem-solving solutions so far have the straight-line property
  - They execute the same statements for every run of the program

```java
public class DisplayForecast
    // main(): application entry point
    public static void main(String[] args) {
        System.out.print("I think there is a world");
        System.out.print(" market for maybe five ");
        System.out.println("computers. ");
        System.out.print(" Thomas Watson, IBM, ");
        System.out.println("1943.");
    }
```
Background

- For general problem solving we need more capabilities
  - The ability to control which statements are executed
  - The ability to control how often a statement is executed

- We will concentrate first on controlling which statements are executed

- Java provides the if and switch conditional constructs to control whether a statement list is executed
  - The if constructs use logical expressions to determine their course of action

- Examination begins with logical expressions
Logical expressions

- The branch of mathematics dealing with logical expressions is Boolean algebra
  - Developed by the British mathematician George Boole
Logical expressions

• A logical expression has either the value logical true or logical false
  – Some expressions whose values are logical true
    • The year 2004 is a leap year
    • A meter equals 100 centimeters
  – Some expressions whose values are logical false
    • A triangle has four sides
    • The area of square is always equal to twice its perimeter
Logical expressions

- There are three primary logical operators for manipulating logical values
  - Logical and
  - Logical or
  - Logical not

- The operators work as most of us would expect
Truth tables

- We use truth tables to give formal specifications of the operators
  - “It works as most of us would expect” allows for ambiguity of interpretation
    - Jim is smiling or Patty is smiling
      - Can both Jim and Patty be smiling?

- Truth tables
  - Lists all combinations of operand values and the result of the operation for each combination

<table>
<thead>
<tr>
<th></th>
<th></th>
<th>p and q</th>
</tr>
</thead>
<tbody>
<tr>
<td>p</td>
<td>q</td>
<td></td>
</tr>
<tr>
<td>False</td>
<td>False</td>
<td>False</td>
</tr>
<tr>
<td>False</td>
<td>True</td>
<td>False</td>
</tr>
<tr>
<td>True</td>
<td>False</td>
<td>False</td>
</tr>
<tr>
<td>True</td>
<td>True</td>
<td>True</td>
</tr>
</tbody>
</table>
# Or and not truth tables

<table>
<thead>
<tr>
<th>p</th>
<th>q</th>
<th>p or q</th>
</tr>
</thead>
<tbody>
<tr>
<td>False</td>
<td>False</td>
<td>False</td>
</tr>
<tr>
<td>False</td>
<td>True</td>
<td>True</td>
</tr>
<tr>
<td>True</td>
<td>False</td>
<td>True</td>
</tr>
<tr>
<td>True</td>
<td>True</td>
<td>True</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>p</th>
<th>not p</th>
</tr>
</thead>
<tbody>
<tr>
<td>False</td>
<td>True</td>
</tr>
<tr>
<td>True</td>
<td>False</td>
</tr>
</tbody>
</table>
**Boolean algebra**

- Can create complex logical expressions by combining simple logical expressions
  - not (p and q)

<table>
<thead>
<tr>
<th>p</th>
<th>q</th>
<th>p and q</th>
<th>not (p and q)</th>
</tr>
</thead>
<tbody>
<tr>
<td>False</td>
<td>False</td>
<td>False</td>
<td>True</td>
</tr>
<tr>
<td>False</td>
<td>True</td>
<td>False</td>
<td>True</td>
</tr>
<tr>
<td>True</td>
<td>False</td>
<td>False</td>
<td>True</td>
</tr>
<tr>
<td>True</td>
<td>True</td>
<td>True</td>
<td>False</td>
</tr>
</tbody>
</table>
DeMorgan’s laws

- not (p and q) equals (not p) or (not q)

<table>
<thead>
<tr>
<th>p</th>
<th>q</th>
<th>p and q</th>
<th>not (p and q)</th>
<th>(not p)</th>
<th>(not q)</th>
<th>(not p) or (not q)</th>
</tr>
</thead>
<tbody>
<tr>
<td>False</td>
<td>False</td>
<td>False</td>
<td>True</td>
<td>True</td>
<td>True</td>
<td>True</td>
</tr>
<tr>
<td>False</td>
<td>True</td>
<td>False</td>
<td>True</td>
<td>True</td>
<td>False</td>
<td>True</td>
</tr>
<tr>
<td>True</td>
<td>False</td>
<td>False</td>
<td>True</td>
<td>False</td>
<td>True</td>
<td>True</td>
</tr>
<tr>
<td>True</td>
<td>True</td>
<td>True</td>
<td>False</td>
<td>False</td>
<td>False</td>
<td>False</td>
</tr>
</tbody>
</table>
DeMorgan’s laws

- not (p or q) equals (not p) and (not q)

<table>
<thead>
<tr>
<th>p</th>
<th>q</th>
<th>p or q</th>
<th>not (p or q)</th>
<th>(not p)</th>
<th>(not q)</th>
<th>(not p) and (not q)</th>
</tr>
</thead>
<tbody>
<tr>
<td>False</td>
<td>False</td>
<td>False</td>
<td>True</td>
<td>True</td>
<td>True</td>
<td>True</td>
</tr>
<tr>
<td>False</td>
<td>True</td>
<td>True</td>
<td>False</td>
<td>True</td>
<td>False</td>
<td>False</td>
</tr>
<tr>
<td>True</td>
<td>False</td>
<td>True</td>
<td>False</td>
<td>True</td>
<td>True</td>
<td>False</td>
</tr>
<tr>
<td>True</td>
<td>True</td>
<td>True</td>
<td>False</td>
<td>False</td>
<td>True</td>
<td>False</td>
</tr>
</tbody>
</table>
A boolean type

• Java has the logical type `boolean`

• Type `boolean` has two literal constants
  – `true`
  – `false`

• Operators
  – The `and` operator is `&&`
  – The `or` operator is `||`
  – The `not` operator is `!`
Defining boolean variables

- Local boolean variables are uninitialized by default

```java
boolean isWhitespace;
boolean receivedAcknowledgement;
boolean haveFoundMissingLink;
```
Defining boolean variables

- Local boolean variables with initialization

```java
boolean canProceed = true;
boolean preferCyan = false;
boolean completedSecretMission = true;
```
Other operators

- Equality operators == and !=
  - Operator ==
    - Evaluates to true if the operands have the same value; otherwise, evaluates to false
  - Operator !=
    - Evaluates to true if the operands have different values; otherwise, evaluates to false

- The operators work with all types of values
Evaluating boolean expressions

• Suppose
  boolean p = true;
  boolean q = false;
  boolean r = true;
  boolean s = false;

• What is the value of
  p
  !s
  q
  p && r
  q || s
  p && s
  p == q
  q != r
  q == r
  r == s
  q != s
Evaluating boolean expressions

• Suppose
  ```
  int i = 1;
  int j = 2;
  int k = 2;
  char c = '#';
  char d = '%';
  char e = '#';
  ```

• What is the value of
  ```
  j == k  i != k
  i == j  j != k
  c == e  d != e
  c == d  c != e
  ```
Take care with floating-point values

- Consider
  ```java
double a = 1;
double b = 0.1 + 0.1 + 0.1 + 0.1 + 0.1 + 0.1
    + 0.1 + 0.1 + 0.1 + 0.1;
double c = .9999999999999999;
  ```

- Two false Java expressions!
  ```java
  a == b    b != c
  ```

- Two true Java expressions!
  ```java
  a != b    b == c
  ```

- Problem lies with the finite precision of the floating-point types
  - Instead test for closeness with floating-point values
    ```java
    Math.abs(a - b) < Small number
    ```
Ordering operators

- Java provides ordering operators for the primitive types
  - Four ordering operators, <, >, <=, and >=
  - They correspond to mathematical operators of <, >, >, and >

- Together the equality and ordering operators are known as the relational operators

- False is less than true
Evaluation boolean expressions

- Suppose
  
  ```
  int i = 1;
  int j = 2;
  int k = 2;
  ```

- What is the value of
  
  ```
  i < j
  j < k
  i <= k
  j >= k
  i >= k
  ```
Unicode values

• Character comparisons are based on their Unicode values

• Characters ‘0’, ‘1’, … ‘9’ have expected order
  – Character ‘0’ has the encoding 48
  – Character ‘1’ has the encoding 49, and so on.

• Upper case Latin letters ‘A’, ‘B’, … ‘Z’ have expected order
  – Character ‘A’ has the encoding 65, character ‘B’ has the encoding 66, and so on.

• Lower case Latin letters ‘a’, ‘b’, … ‘z’ have expected order
  • Character ‘a’ has the encoding 97
  • Character ‘b’ has the encoding 98, and so on.
Evaluation boolean expressions

• Suppose
  char c = '2';
  char d = '3';
  char e = '2';

• What is the value of
  c < d
  c < e
  c <= e
  d >= e
  c >= e
Operator precedence revisited

- Highest to lowest
  - Parentheses
  - Unary operators
  - Multiplicative operators
  - Additive operators
  - Relational ordering
  - Relational equality
  - Logical and
  - Logical or
  - Assignment
Conditional constructs

• Provide
  – Ability to control whether a statement list is executed

• Two constructs
  – If statement
    • if
    • if-else
    • if-else-if
  – Switch statement
Basic if statement

• Syntax
  
  ```
  if (Expression)
  Action
  ```

• If the `Expression` is true then execute `Action`

• `Action` is either a single statement or a group of statements within braces

• For us, it will always be a group of statements within braces
Example

if (value < 0) {
    value = -value;
}

Is our number negative?

If Value is not less than zero then our number is fine as is.

If Value is less than zero then we need to update its value to that of its additive inverse.

Our number is now definitely nonnegative.
Sorting two values

System.out.print("Enter an integer number: ");
int value1 = stdin.nextInt();
System.out.print("Enter another integer number: ");
int value2 = stdin.nextInt();

// rearrange numbers if necessary
if (value2 < value1) {
    // values are not in sorted order
    int rememberValue1 = value1;
    value1 = value2;
    value2 = rememberValue1;
}

// display values
System.out.println("The numbers in sorted order are "+value1 + " and then " + value2);

What happens if the user enters 11 and 28?
What happens if the user enters 11 and 4?
If semantics

Are the numbers out of order

Rearrange value1 and value2 to put their values in the proper order

value2 < value1

true

false

int rememberValue1 = value1
value1 = value2
value2 = rememberValue1

The numbers were rearranged into the proper order

The numbers were initially in order

The numbers are in order
Why we always use braces

• What is the output?

```java
int m = 5;
int n = 10;

if (m < n)
    ++m;
    ++n;

System.out.println(" m = " + m + " n = " + n);
```
The if-else statement

- Syntax

  ```
  if (Expression)
      Action_1
  else
      Action_2
  ```

- If `Expression` is true then execute `Action_1` otherwise execute `Action_2`

- The actions are either a single statement or a list of statements within braces
Finding the maximum of two values

```java
System.out.print("Enter an integer number: ");
int value1 = stdin.nextInt();
System.out.print("Enter another integer number: ");
int value2 = stdin.nextInt();

int maximum;
if (value1 < value2) {   // is value2 larger?
    maximum = value2;     // yes: value2 is larger
} else { // (value1 >= value2)
    maximum = value1;     // no: value2 is not larger
}
System.out.println("The maximum of "+ value1
    + " and "+ value2 + " is "+ maximum);
```
Finding the maximum of two values

Is value2 larger than value1

Yes, it is. So value2 is larger than value1. In this case, maximum is set to value2

No, it's not. So value1 is at least as large as value2. In this case, maximum is set to value1

Either case, maximum is set correctly

true

false

maximum = value2

maximum = value1
Why we use whitespace

• What does the following do?

System.out.print("Enter an integer number: ");
int value1 = stdin.nextInt();
System.out.print("Enter another integer number: ");
int value2 = stdin.nextInt();
if (value2 < value1) {
    int rememberValue1 = value1;
    value1 = value2;
    value1 = rememberValue1;
    value2 = rememberValue2;
}
System.out.println("The numbers in sorted order are "+ value1 + " and then "+ value2);
Testing objects for equality

- Consider

  System.out.print("Enter an integer number: ");
  int n1 = stdin.nextInt();
  System.out.print("Enter another integer number: ");
  int n2 = stdin.nextInt();

  if (n1 == n2) {
    System.out.println("Same");
  } else {
    System.out.println("Different");
  }

What is the output if the user enters 88 both times?
What is the output if the user enters 88 and 3?
Testing objects for equality

- Consider
  ```java
  System.out.print("Enter a string: ");
  String s1 = stdin.next();
  System.out.print("Enter another string: ");
  String s2 = stdin.next();

  if (s1 == s2) {
      System.out.println("Same");
  } else {
      System.out.println("Different");
  }
  ```

What is the output if the user enters "pastel" both times?
Testing objects for equality

- When it is executed
  System.out.print("Enter a string: ");
  String s1 = stdin.next();
  System.out.print("Enter another string: ");
  String s2 = stdin.next();

- Memory looks like

- As a result no matter what is entered s1 and s2 are not the same
  - They refer to different objects
Testing operators for equality

- Consider
  ```java
  System.out.print("Enter a string: ");
  String s1 = stdin.next();
  System.out.print("Enter another string: ");
  String s2 = stdin.next();

  if (s1.equals(s2)) {
    System.out.println("Same");
  }
  else {
    System.out.println("Different");
  }
  ```

Tests whether s1 and s2 represent the same object.

All objects have a method equals(). Their implementation is class-specific. The String equals() method – like many others – tests for equivalence in representation.
Some handy String class methods

- isDigit()
  - Tests whether character is numeric
- isLetter()
  - Tests whether character is alphabetic
- isLowerCase()
  - Tests whether character is lowercase alphabetic
- isWhiteSpace()
  - Tests whether character is one of the space, tab, formfeed, or newline characters
Some handy String class methods

- `isUpperCase()`: Tests whether character is uppercase alphabetic.
- `toLowerCase()`: If the character is alphabetic then the lowercase equivalent of the character is returned; otherwise, the character is returned.
- `toUpperCase()`: If the character is alphabetic then the uppercase equivalent of the character is returned; otherwise, the character is returned.
If-else-if

Consider

if (number == 0) {
    System.out.println("zero");
} else if (number > 0) {
    System.out.println("positive");
} else {
    System.out.println("negative");
}
If-else-if

• Better expresses the meaning of what is going on

Same results as previous segment - but this segment

```java
if (number == 0) {
    System.out.println("zero");
} else if (number > 0) {
    System.out.println("positive");
} else {
    System.out.println("negative");
}
```

Better

If-else-if
Sorting three values

- For sorting values $n_1$, $n_2$, and $n_3$ there are six possible orderings
  - $n_1 \leq n_2 \leq n_3$
  - $n_1 \leq n_3 \leq n_2$
  - $n_2 \leq n_1 \leq n_3$
  - $n_2 \leq n_3 \leq n_1$
  - $n_3 \leq n_1 \leq n_2$
  - $n_3 \leq n_2 \leq n_1$

- Suppose $s_1$, $s_2$, $s_3$ are to made a sorted version of $n_1$, $n_2$, and $n_3$
Sorting three values

if ((n1 <= n2) && (n2 <= n3)) {  // n1 <= n2 <= n2
    s1 = n1;   s2 = n2;   s3 = n3;
} else if ((n1 <= n3) && (n3 <= n2)) {  // n1 <= n3 <= n2
    s1 = n1; s2 = n3; s3 = n2;
} else if ((n2 <= n1) && (n1 <= n3)) {  // n2 <= n1 <= n3
    s1 = n2; s2 = n1; s3 = n3;
} else if ((n2 <= n3) && (n3 <= n1)) {  // n2 <= n3 <= n1
    s1 = n2; s2 = n3; s3 = n1;
} else if ((n3 <= n1) && (n1 <= n2)) {  // n3 <= n1 <= n2
    s1 = n3; s2 = n1; s3 = n2;
} else {  // n3 <= n2 <= n1
    s1 = n3; s2 = n2; s3 = n1;
}
Switch statement

- Software engineers often confronted with programming tasks where required action depends on the values of integer expressions.

  - The if-else-if construct can be used separately compare the desired expression to a particular value.
  - If the expression and value are equal, then perform the appropriate action.

- Because such programming tasks occur frequently, Java includes a switch statement.
  - The task is often more readable with the switch then with the if-else-if.
Switch statement

```
switch (SwitchExpression) {
    case CaseExpression1:
        Action1;
    case CaseExpression2:
        Action2;
    ...
    case CaseExpression_n:
        Action_n;
    default:
        Action_{n+1};
}
```
Testing for vowel-ness

```java
switch (ch) {
    case 'a': case 'A':
    case 'e': case 'E':
    case 'i': case 'I':
    case 'o': case 'O':
    case 'u': case 'U':
        System.out.println("vowel");
        break;  // The break causes an exiting of the switch
    default:
        System.out.println("not a vowel");
}
```

Handles all of the other cases
Processing a request

System.out.print("Enter a number: ");
int n1 = stdin.nextInt();

System.out.print("Enter another number: ");
int n2 = stdin.nextInt();

System.out.print("Enter desired operator: ");
char operator = stdin.next().charAt(0);

switch (operator) {
    case '+': System.out.println(n1 + n2); break;
    case '-': System.out.println(n1 - n2); break;
    case '*': System.out.println(n1 * n2); break;
    case '/': System.out.println(n1 / n2); break;
    default: System.out.println("Illegal request");
}
Short-circuit evaluation

• The value of a logical expression can be known before all the operands have been considered
  – If left operand of && is false, then the value must be false
  – If right operand of || is true, then the value must be true

• Java uses these properties to make logical operations efficient
  – Evaluates left operand before it evaluates right operand
  – If the operator value is determined from the left operand, then the right operand is not evaluated
• The operation is short-circuited
Short-circuit evaluation

• Short-circuit evaluation is useful when some property must be true for some other expression to be evaluated.

• Suppose you are interested in knowing whether scoreSum divided by nbrScores is greater than value:
  – The condition can be evaluated only if nbrScores is nonzero.

• The following expression correctly represents the condition:
  \[(\text{nbrScores } \neq 0) \land ((\text{scoreSum} / \text{nbrScores}) > \text{value})\]
ColoredTriangle

• Background
  – Triangles are an important shape in the world of computer graphics
  – When computer animations are created, scenes are typically decomposed into a collection of colored triangles

• Informal specification
  – Represent a colored triangle in two-dimensional space
  – Provide the constructors and methods a reasonable user would expect
Colored Triangle – see the cat
ColoredTriangle – expected constructors

- Default construction
  - Construct a reasonable triangle representation even though no explicit attributes values are given
    
    public ColoredTriangle()

- Specific construction
  - Construct a triangle representation from explicit attributes values
    
    public ColoredTriangle(Point v1, Point v2, Point v3, Color c)
ColoredTriangle – expected behaviors

- Provide the area
  - Return the area of the associated triangle
    
    public double getArea()

- Provide the perimeter
  - Return the perimeter of the associated triangle
    
    public double getPerimeter()

- Access an endpoint
  - Provide a requested endpoint of the associated triangle
    
    public Point getPoint(int i)
ColoredTriangle – expected behaviors

- Access the color
  - Provide the color of the associated triangle
  
  public Point getColor()

- Set an endpoint
  - Set a particular endpoint point of the associated triangle to a given value
  
  public void setPoint(int i, Point p)

- Set color of the triangle
  - Set the color of the associated triangle to a given value
  
  public void setColor(Color c)
ColoredTriangle – expected behaviors

- Render
  - Draw the associated triangle in a given graphical context
    public void paint(Graphics g)

- Test equality
  - Report whether the associated triangle is equivalent to a given triangle
    public boolean equals(Object v)

- String representation
  - Provide a textual representation of the attributes of the associated triangle
    public String toString()
ColoredTriangle – attributes

- To implement the behaviors
  - Knowledge of the triangle color and three endpoints suffices
  - Endpoint can be represented using two int values per location or as a Point
    - Point seem more natural

private Color color
  - Color of the associated triangle
private Point p1
  - References the first point of the associated triangle
private Point p2
  - References the second point of the associated triangle
private Point p3
  - References the third point of the associated triangle
Default constructor – implementation

// coloredTriangle(): default constructor
public ColoredTriangle() {
    Point a = new Point(1, 1);
    Point b = new Point(2, 2);
    Point c = new Point(3, 3);
    setPoint(1, a);
    setPoint(2, b);
    setPoint(3, c);
    setColor(Color.BLACK);
}
Implementation – accessor getPoint()

// getPoint(): endpoint accessor
public Point getPoint(int i) {
    if (i == 1) {
        return p1;
    }
    else if (i == 2) {
        return p2;
    }
    else if (i == 3) {
        return p3;
    } else {
        System.out.println("Unexpected endpoint access: "+ i);
        System.exit(i);
        return null;
    }
}

Won’t be executed but compiler wants every execution path to end with a return
Implementation – facilitator toString()

// toString(): string facilitator
public String toString() {
    Point v1 = getPoint(1);
    Point v2 = getPoint(2);
    Point v3 = getPoint(3);
    Color c = getColor();

    return "ColoredRectangle[" + v1 + ", " + v2 + ", " + v3 + ", " + c + "]";
}

Standard to include class name when expected use is for debugging
Implementation – facilitator toString()

Point a = new Point(2,1),
Point b = new Point(1,2)
Point c = new Point(3,2);
ColoredTriangle u = new ColoredTriangle(a, b, c, Color.RED);
System.out.println(u);  // displays string version of u

ColoredTriangle[java.awt.Point[x=2,y=1],
    java.awt.Point[x=1,y=2], java.awt.Point[x=3,y=2],
    java.awt.Color[r=255,g=0,b=0]]
// equals(): equals facilitator
public boolean equals(Object p) {
    if (p instanceof ColoredTriangle) {
        Point v1 = getPoint(1);
        Point v2 = getPoint(2);
        Point v3 = getPoint(3);
        Color c = getColor();
        ColoredTriangle t = (ColoredTriangle) p;

        return v1.equals(t.getPoint(1))
            && v2.equals(t.getPoint(2))
            && v3.equals(t.getPoint(3))
            && c.equals(t.getColor());
    } else {
        return false;
    }
}
Implementation – facilitator equals()

```java
ColoredTriangle e = new ColoredTriangle();
ColoredTriangle f = new ColoredTriangle(new Point(2,1),
    new Point(1,2), new Point(3,2), Color.YELLOW);
ColoredTriangle g = new ColoredTriangle(new Point(2,1),
    new Point(1,2), new Point(3,2), Color.YELLOW);

boolean flag1 = e.equals(f);
boolean flag2 = e.equals(g);
boolean flag2 = e.equals(g);

System.out.println(flag1 + " " + flag2 + " " + flag3);
```
Implementation – facilitator equals()

ColoredTriangle
p1: Point x: 1 y: 1
p2: Point x: 2 y: 2
p3: Point x: 3 y: 3
color: Color r: 0 y: 0 g: 0

ColoredTriangle
p1: Point x: 2 y: 1
p2: Point x: 1 y: 2
p3: Point x: 3 y: 2
color: Color r: 0 y: 255 g: 0

ColoredTriangle
p1: Point x: 2 y: 1
p2: Point x: 1 y: 2
p3: Point x: 3 y: 2
color:
Implementation – facilitator paint()

// paint(): render facilitator
public void paint(Graphics g) {
    Point v1 = getPoint(1);
    Point v2 = getPoint(2);
    Point v3 = getPoint(3);
    Color c = getColor();

    g.setColor(c);

    Polygon t = new Polygon();
t.addPoint(v1.x, v1.y);
t.addPoint(v2.x, v2.y);
t.addPoint(v3.x, v3.y);
    g.fillPolygon(t);
}

Part of awt
Renders a polygon using the list of points in the polygon referenced by t
Iteration
Java looping

• Options
  – while
  – do-while
  – for

• Allow programs to control how many times a statement list is executed
Averaging

• Problem
  – Extract a list of positive numbers from standard input and produce their average
    • Numbers are one per line
    • A negative number acts as a *sentinel* to indicate that there are no more numbers to process

• Observations
  – Cannot supply sufficient code using just assignments and conditional constructs to solve the problem
    • Don’t how big of a list to process
  – Need ability to repeat code as needed
Averaging

- Problem
  - Extract a list of positive numbers from standard input and produce their average
    - Numbers are one per line
    - A negative number acts as a *sentinel* to indicate that there are no more numbers to process
- Algorithm
  - Prepare for processing
  - Get first input
  - While there is an input to process do {
    - Process current input
    - Get the next input
  }
  - Perform final processing
Averaging

- **Problem**
  - Extract a list of positive numbers from standard input and produce their average
    - Numbers are one per line
    - A negative number acts as a *sentinel* to indicate that there are no more numbers to process

- **Sample run**
  Enter positive numbers one per line.
  Indicate end of list with a negative number.
  4.5
  0.5
  1.3
  -1
  Average 2.1
public class NumberAverage {
    // main(): application entry point
    public static void main(String[] args) throws IOException {
        // set up the list processing
        // prompt user for values
        // get first value
        // process values one-by-one
        while (value >= 0) {
            // add value to running total
            // processed another value
            // prepare next iteration - get next value
        }
        // display result
        if (valuesProcessed > 0) {
            // compute and display average
        } else {
            // indicate no average to display
        }
    }
}
System.out.println("Enter positive numbers 1 per line.
Indicate end of the list with a negative number.");

Scanner stdin = Scanner.create(System.in);

int valuesProcessed = 0;
double valueSum = 0;

double value = stdin.nextDouble();
while (value >= 0) {
    valueSum += value;
    ++valuesProcessed;
    value = stdin.nextDouble();
}

if (valuesProcessed > 0) {
    double average = valueSum / valuesProcessed;
    System.out.println("Average: " + average);
}
else {
    System.out.println("No list to average");
}
While syntax and semantics

\[
\text{while ( Expression ) Action}
\]

- Logical expression that determines whether Action is to be executed — if Expression evaluates to true, then Action is executed; otherwise, the loop is terminated.

- Action is either a single statement or a statement list within braces. The action is also known as the body of the loop. After the body is executed, the test expression is reevaluated. If the expression evaluates to true, the body is executed again. The process repeats until the test expression evaluates to false.
While semantics for averaging problem

```
// process values one-by-one
while (value >= 0) {
    // add value to running total
    valueSum += value;
    // processed another value
    ++valuesProcessed;
    // prepare to iterate -- get the next input
    value = stdin.nextDouble();
}
```

Test expression is evaluated at the start of each iteration of the loop. Its value indicates whether there is a number to process.

If test expression is true, these statements are executed. Afterward, the test expression is reevaluated and the process repeats.
While Semantics

Expression is evaluated at the start of each iteration of the loop.

If Expression is true, Action is executed.
If Expression is false, program execution continues with next statement.
```java
int valuesProcessed = 0;
double valueSum = 0;

double value = stdin.nextDouble());
while (value >= 0) {
    valueSum += value;
    ++valuesProcessed;
    value = stdin.nextDouble());
}
if (valuesProcessed > 0) {
    double average = valueSum / valuesProcessed;
    System.out.println("Average: " + average);
} else {
    System.out.println("No list to average");
}
```
Suppose input contains: 4.5 0.5 1.3 -1

```java
int valuesProcessed = 0;
double valueSum = 0;

double value = stdin.nextDouble();
while (value >= 0) {
    valueSum += value;
    ++valuesProcessed;
    value = stdin.nextDouble();
}
if (valuesProcessed > 0) {
    double average = valueSum / valuesProcessed;
    System.out.println("Average: " + average);
} else {
    System.out.println("No list to average");
}
```
Execution Trace

Suppose input contains: 4.5 0.5 1.3 -1

```java
int valuesProcessed = 0;
double valueSum = 0;

double value = stdin.nextDouble();

while (value >= 0) {
    valueSum += value;
    ++valuesProcessed;
    value = stdin.nextDouble();
}

if (valuesProcessed > 0) {
    double average = valueSum / valuesProcessed;
    System.out.println("Average: " + average);
}
else {
    System.out.println("No list to average");
}
```
Execution Trace

execute input contains: 4.5 0.5 1.3 -1

```java
int valuesProcessed = 0;
double valueSum = 0;
double value = stdin.nextDouble());

while (value >= 0) {
    valueSum += value;
    ++valuesProcessed;
    value = stdin.nextDouble());
}

if (valuesProcessed > 0) {
    double average = valueSum / valuesProcessed;
    System.out.println("Average: "+ average);
}
else {
    System.out.println("No list to average");
}
```
Suppose input contains: 4.5 0.5 1.3 -1

```
int valuesProcessed = 0;
double valueSum = 0;
double value = stdin.nextDouble());

while (value >= 0) {
    valueSum += value;
    ++valuesProcessed;
    value = stdin.nextDouble());
}

if (valuesProcessed > 0) {
    double average = valueSum / valuesProcessed;
    System.out.println("Average: " + average);
}
else {
    System.out.println("No list to average");
}
```
Suppose input contains: 4.5 0.5 1.3 -1

```
int valuesProcessed = 0;
double valueSum = 0;

double value = stdin.nextDouble();

while (value >= 0) {
    valueSum += value;
    ++valuesProcessed;
    value = stdin.nextDouble();
}

if (valuesProcessed > 0) {
    double average = valueSum / valuesProcessed;
    System.out.println("Average: " + average);
}
else {
    System.out.println("No list to average");
}
```
int valuesProcessed = 0;
double valueSum = 0;
double value = stdin.nextDouble());

while (value >= 0) {
    valueSum += value;
    ++valuesProcessed;
    value = stdin.nextDouble());
}

if (valuesProcessed > 0) {
    double average = valueSum / valuesProcessed;
    System.out.println("Average: " + average);
}
else {
    System.out.println("No list to average");
}
int valuesProcessed = 0;
double valueSum = 0;
double value = stdin.nextDouble();

while (value >= 0) {
    valueSum += value;
    ++valuesProcessed;
    value = stdin.nextDouble();
}

if (valuesProcessed > 0) {
    double average = valueSum / valuesProcessed;
    System.out.println("Average: "+ average);
} else {
    System.out.println("No list to average");
}
Suppose input contains: 4.5 0.5 1.3 -1

Execution Trace

```
int valuesProcessed = 0;
double valueSum = 0;
double value = stdin.nextDouble();

while (value >= 0) {
    valueSum += value;
    ++valuesProcessed;
    value = stdin.nextDouble();
}

if (valuesProcessed > 0) {
    double average = valueSum / valuesProcessed;
    System.out.println("Average: " + average);
}
else {
    System.out.println("No list to average");
}
```
Suppose input contains: 4.5 0.5 1.3 -1

```java
int valuesProcessed = 0;
double valueSum = 0;
double value = stdin.nextDouble();

while (value >= 0) {
    valueSum += value;
    ++valuesProcessed;
    value = stdin.nextDouble();
}

if (valuesProcessed > 0) {
    double average = valueSum / valuesProcessed;
    System.out.println("Average: "+average);
} else {
    System.out.println("No list to average");
}
```
```java
int valuesProcessed = 0;
double valueSum = 0;
double value = stdin.nextDouble());

while (value >= 0) {
    valueSum += value;
    ++valuesProcessed;
    value = stdin.nextDouble());
}

if (valuesProcessed > 0) {
    double average = valueSum / valuesProcessed;
    System.out.println("Average: " + average);
}
else {
    System.out.println("No list to average");
}
```

Suppose input contains: 4.5 0.5 1.3 -1
Execution Trace

Suppose input contains: 4.5 0.5 1.3 -1

```java
int valuesProcessed = 0;
double valueSum = 0;
double value = stdin.nextDouble();

while (value >= 0) {
    valueSum += value;
    ++valuesProcessed;
    value = stdin.nextDouble();
}

if (valuesProcessed > 0) {
    double average = valueSum / valuesProcessed;
    System.out.println("Average: " + average);
} else {
    System.out.println("No list to average");
}
```
Suppose input contains: 4.5 0.5 1.3 -1

```
int valuesProcessed = 0;
double valueSum = 0;

double value = stdin.nextDouble();
while (value >= 0) {
    valueSum += value;
    ++valuesProcessed;
    value = stdin.nextDouble();
}
if (valuesProcessed > 0) {
    double average = valueSum / valuesProcessed;
    System.out.println("Average: " + average);
}
else {
    System.out.println("No list to average");
}
```
int valuesProcessed = 0;
double valueSum = 0;
double value = stdin.nextDouble();

while (value >= 0) {
    valueSum += value;
    ++valuesProcessed;
    value = stdin.nextDouble();
}

if (valuesProcessed > 0) {
    double average = valueSum / valuesProcessed;
    System.out.println("Average: "+ average);
} else {
    System.out.println("No list to average");
}
Suppose input contains: 4.5 0.5 1.3 -1

```
int valuesProcessed = 0;
double valueSum = 0;
double value = stdin.nextDouble();

while (value >= 0) {
    valueSum += value;
    ++valuesProcessed;
    value = stdin.nextDouble();
}

if (valuesProcessed > 0) {
    double average = valueSum / valuesProcessed;
    System.out.println("Average: "+ average);
}
else {
    System.out.println("No list to average");
}
```
Execution Trace

Suppose input contains: 4.5 0.5 1.3 -1

```
int valuesProcessed = 0;
double valueSum = 0;
double value = stdin.nextDouble();

while (value >= 0) {
    valueSum += value;
    ++valuesProcessed;
    value = stdin.nextDouble();
}

if (valuesProcessed > 0) {
    double average = valueSum / valuesProcessed;
    System.out.println("Average: " + average);
}
else {
    System.out.println("No list to average");
}
```
Suppose input contains: 4.5 0.5 1.3 -1

Execution Trace

```java
int valuesProcessed = 0;
double valueSum = 0;
double value = stdin.nextDouble();
while (value >= 0) {
    valueSum += value;
    ++valuesProcessed;
    value = stdin.nextDouble();
}
if (valuesProcessed > 0) {
    double average = valueSum / valuesProcessed;
    System.out.println("Average: " + average);
} else {
    System.out.println("No list to average");
}
```
```java
int valuesProcessed = 0;
double valueSum = 0;
double value = stdin.nextDouble();
while (value >= 0) {
    valueSum += value;
    ++valuesProcessed;
    value = stdin.nextDouble();
}
if (valuesProcessed > 0) {
    double average = valueSum / valuesProcessed;
    System.out.println("Average: " + average);
} else {
    System.out.println("No list to average");
}
```

Suppose input contains: 4.5 0.5 1.3 -1

<table>
<thead>
<tr>
<th>valuesProcessed</th>
<th>valueSum</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>6.3</td>
<td>-1</td>
</tr>
</tbody>
</table>
Execution Trace

int valuesProcessed = 0;
double valueSum = 0;
double value = stdin.nextDouble();

while (value >= 0) {
    valueSum += value;
    ++valuesProcessed;
    value = stdin.nextDouble();
}

if (valuesProcessed > 0) {
    double average = valueSum / valuesProcessed;
    System.out.println("Average: " + average);
}
else {
    System.out.println("No list to average");
}
Execution Trace

```java
int valuesProcessed = 0;
double valueSum = 0;
double value = stdin.nextDouble();

while (value >= 0) {
    valueSum += value;
    ++valuesProcessed;
    value = stdin.nextDouble();
}

if (valuesProcessed > 0) {
    double average = valueSum / valuesProcessed;
    System.out.println("Average: "+ average);
} else {
    System.out.println("No list to average");
}
```

Suppose input contains: 4.5 0.5 1.3 -1

<table>
<thead>
<tr>
<th>valuesProcessed</th>
<th>valueSum</th>
<th>value</th>
<th>average</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>6.3</td>
<td>-1</td>
<td>2.1</td>
</tr>
</tbody>
</table>
Converting text to strictly lowercase

```java
public static void main(String[] args) {
    Scanner stdin = Scanner.create(System.in);
    System.out.println("Enter input to be converted:");
    String converted = "";

    while (stdin.hasNext()) {
        String currentLine = stdin.nextLine();
        String currentConversion =
            currentLine.toLowerCase();
        converted += (currentConversion + "\n");
    }

    System.out.println("\nConversion is:\n" + converted);
}
```
Sample run

A Ctrl+z was entered. It is the Windows escape sequence for indicating end-of-file.
public static void main(String[] args) {
    Scanner stdin = Scanner.create(System.in);
    System.out.println("Enter input to be converted:");
    String converted = "";

    while (stdin.hasNext()) {
        String currentLine = stdin.nextLine();
        String currentConversion =
            currentLine.toLowerCase();
        converted += (currentConversion + "\n");
    }

    System.out.println("\nConversion is:\n" + converted);
}
Program trace

public static void main(String[] args) {
    Scanner stdin = Scanner.create(System.in);
    System.out.println("Enter input to be converted:");
    String converted = "";

    while (stdin.hasNext()) {
        String currentLine = stdin.nextLine();
        String currentConversion =
            currentLine.toLowerCase();
        converted += (currentConversion + "\n");
    }

    System.out.println("\nConversion is:\n" + converted);
}
Program trace

public static void main(String[] args) {
    Scanner stdin = Scanner.create(System.in);
    System.out.println("Enter input to be converted:");
    String converted = "";

    while (stdin.hasNext()) {
        String currentLine = stdin.nextLine();
        String currentConversion =
            currentLine.toLowerCase();
        converted += (currentConversion + "\n");
    }

    System.out.println("\nConversion is:\n" + converted);
}
public static void main(String[] args) {
    Scanner stdin = Scanner.create(System.in);

    System.out.println("Enter input to be converted:");

    String converted = "";

    while (stdin.hasNext()) {
        String currentLine = stdin.nextLine();
        String currentConversion =
            currentLine.toLowerCase();
        converted += (currentConversion + "\n");
    }

    System.out.println("\nConversion is:\n" + converted);
}
The append assignment operator updates the representation of converted to include the current input line.

- **Representation of lower case conversion of current input line**
- **Newline character is needed because method readLine() "strips" them from the input**
Converting text to strictly lowercase

```java
public static void main(String[] args) {
    Scanner stdin = Scanner.create(System.in);
    System.out.println("Enter input to be converted:");
    String converted = "";

    while (stdin.hasNext()) {
        String currentLine = stdin.nextLine();
        String currentConversion =
            currentLine.toLowerCase();
        converted += (currentConversion + "\n");
    }

    System.out.println("\nConversion is:\n" + converted);
}
```
Loop design

• Questions to consider in loop design and analysis
  – What initialization is necessary for the loop’s test expression?
  – What initialization is necessary for the loop’s processing?
  – What causes the loop to terminate?
  – What actions should the loop perform?
  – What actions are necessary to prepare for the next iteration of the loop?
  – What conditions are true and what conditions are false when the loop is terminated?
  – When the loop completes what actions are need to prepare for subsequent program processing?
Reading a file

- Background

Scanner provides a way to process text input

System.in is an InputStream variable. The stream contains text

Scanner stdin = new Scanner(System.in);}
Reading a file

• Class File
  – Provides a system-independent way of representing a file name

• Constructor File(String s)
  – Creates a File with name s
  – Name can be either an absolute pathname or a pathname relative to the current working folder
Reading a file

Scanner stdin = Scanner.create(System.in);

System.out.print("Filename: ");
String filename = stdin.next();

File file = new File(filename);
Scanner fileIn = Scanner.create(file);

while (fileIn.hasNext()) {
    String currentLine = fileIn.nextLine();
    System.out.println(currentLine);
}

fileIn.close();
Reading a file

Scanner stdin = Scanner.create(System.in);
System.out.print("Filename: ");
String filename = stdin.next();

File file = new File(filename);
Scanner fileIn = Scanner.create(file);

while (fileIn.hasNext()) {
    String currentLine = fileIn.nextLine();
    System.out.println(currentLine);
}

fileIn.close();

Set up standard input stream
Reading a file

Scanner stdin = Scanner.create(System.in);
System.out.print("Filename: ");
String filename = stdin.next();

File file = new File(filename);
Scanner fileIn = Scanner.create(file);

while (fileIn.hasNext()) {
    String currentLine = fileIn.nextLine();
    System.out.println(currentLine);
}

fileIn.close();
Reading a file

Scanner stdin = Scanner.create(System.in);
System.out.print("Filename: ");
String filename = stdin.next();

File file = new File(filename);
Scanner fileIn = Scanner.create(file);

while (fileIn.hasNext()) {
    String currentLine = fileIn.nextLine();
    System.out.println(currentLine);
}

fileIn.close();

Determine the associated file
Reading a file

Scanner stdin = Scanner.create(System.in);

System.out.print("Filename: ");
String filename = stdin.next();

File file = new File(filename);
Scanner fileIn = Scanner.create(file);

while (fileIn.hasNext()) {
    String currentLine = fileIn.nextLine();
    System.out.println(currentLine);
}
fileIn.close();

Set up file stream
Reading a file

Scanner stdin = Scanner.create(System.in);
System.out.println("Filename: ");
String filename = stdin.next();

File file = new File(filename);
Scanner fileIn = Scanner.create(file);

while (fileIn.hasNext()) {
    String currentLine = fileIn.nextLine();
    System.out.println(currentLine);
}

fileIn.close();

Process lines one by one
Reading a file

Scanner stdin = Scanner.create(System.in);
System.out.print("Filename: ");
String filename = stdin.next();

File file = new File(filename);
Scanner fileIn = Scanner.create(file);

while (fileIn.hasNext()) {
    String currentLine = fileIn.nextLine();
    System.out.println(currentLine);
}

fileIn.close();

Is there any text
Reading a file

Scanner stdin = Scanner.create(System.in);
System.out.print("Filename: ");
String filename = stdin.next();

File file = new File(filename);
Scanner fileIn = Scanner.create(file);

while (fileIn.hasNext()) {
    String currentLine = fileIn.nextLine();
    System.out.println(currentLine);
}

fileIn.close();

Get the next line of text
Reading a file

Scanner stdin = Scanner.create(System.in);

System.out.print("Filename: ");
String filename = stdin.next();

File file = new File(filename);
Scanner fileIn = Scanner.create(file);

while (fileIn.hasNext()) {
    String currentLine = fileIn.nextLine();
    System.out.println(currentLine);
}

fileIn.close();

Display current line
Reading a file

Scanner stdin = Scanner.create(System.in);
System.out.print("Filename: ");
String filename = stdin.next();

File file = new File(filename);
Scanner fileIn = Scanner.create(file);

while (fileIn.hasNext()) {
    String currentLine = fileIn.nextLine();
    System.out.println(currentLine);
}

fileIn.close();

Make sure there is another to process
If not, loop is done
Reading a file

Scanner stdin = Scanner.create(System.in);
System.out.print("Filename: ");
String filename = stdin.next();

File file = new File(filename);
Scanner fileIn = Scanner.create(file);

while (fileIn.hasNext()) {
    String currentLine = fileIn.nextLine();
    System.out.println(currentLine);
}

fileIn.close();

Close the stream
The For Statement

```java
int currentTerm = 1;

for (int i = 0; i < 5; ++i) {
    System.out.println(currentTerm);
    currentTerm *= 2;
}
```
The For Statement

```java
int currentTerm = 1;
for (int i = 0; i < 5; ++i) {
    System.out.println(currentTerm);
    currentTerm *= 2;
}
```

Initialization step is performed only once -- just prior to the first evaluation of the test expression.
The For Statement

Initialization step is performed only once -- just prior to the first evaluation of the test expression.

```java
int currentTerm = 1;
for (int i = 0; i < 5; ++i) {
    System.out.println(currentTerm);
    currentTerm *= 2;
}
```

The body of the loop iterates while the test expression is true.
The For Statement

Initialization step is performed only once -- just prior to the first evaluation of the test expression.

```java
int currentTerm = 1;
for (int i = 0; i < 5; ++i) {
    System.out.println(currentTerm);
    currentTerm *= 2;
}
```

The body of the loop iterates while the test expression is true.

The body of the loop displays the current term in the number series. It then determines what is to be the new current number in the series.
The For Statement

Initialization step is performed only once -- just prior to the first evaluation of the test expression.

```java
int currentTerm = 1;
for (int i = 0; i < 5; ++i) {
    System.out.println(currentTerm);
    currentTerm *= 2;
}
```

The body of the loop displays the current term in the number series. It then determines what is to be the new current number in the series.

The body of the loop iterates while the test expression is true.

After each iteration of the body of the loop, the update expression is reevaluated.
The ForExpr is evaluated at the start of each iteration of the loop.

ForInit

The ForExpr is evaluated once at the beginning of the for statement's execution.

ForExpr

true

If ForExpr is true, Action is executed.

false

If ForExpr is false, program execution continues with the next statement.

Action

After the Action has completed, the PostExpression is evaluated.

PostExpr

After evaluating the PostExpression, the next iteration of the loop starts.
For statement syntax

Logical test expression that determines whether the action and update step are executed.

Initialization step prepares for the first evaluation of the test expression.

Update step is performed after the execution of the loop body.

The body of the loop iterates whenever the test expression evaluates to true.

```
for ( ForInit ; ForExpression ; ForUpdate ) Action
```
Execution Trace

for (int i = 0; i < 3; ++i) {
    System.out.println("i is " + i);
}

System.out.println("all done");
Execution Trace

```java
for (int i = 0; i < 3; ++i) {
    System.out.println("i is " + i);
}

System.out.println("all done");
```
Execution Trace

for (int i = 0; i < 3; ++i) {
    System.out.println("i is " + i);
}

System.out.println("all done");

i is 0
Execution Trace

```java
for (int i = 0; i < 3; ++i) {
    System.out.println("i is "+i);
}
System.out.println("all done");

i is 0
```
for (int i = 0; i < 3; ++i) {
    System.out.println("i is " + i);
}
System.out.println("all done");
Execution Trace

for (int i = 0; i < 3; ++i) {
    System.out.println("i is " + i);
}

System.out.println("all done");

i is 0
Execution Trace

```java
for (int i = 0; i < 3; ++i) {
    System.out.println("i is "+i);
}
System.out.println("all done");
```

i is 0
i is 1
Execution Trace

for (int i = 0; i < 3; ++i) {
    System.out.println("i is "+ i);
}

System.out.println("all done");

i is 0
i is 1
Execution Trace

for (int i = 0; i < 3; ++i) {
    System.out.println("i is "+ i);
}

System.out.println("all done");

i is 0
i is 1
Execution Trace

```java
for (int i = 0; i < 3; ++i) {
    System.out.println("i is " + i);
}

System.out.println("all done");
```

i is 0
i is 1
for (int i = 0; i < 3; ++i) {
    System.out.println("i is "+i);
}
System.out.println("all done");
Execution Trace

for (int i = 0; i < 3; ++i) {
    System.out.println("i is " + i);
}

System.out.println("all done");

i is 0
i is 1
i is 2
Execution Trace

for (int i = 0; i < 3; ++i) {
    System.out.println("i is " + i);
}

System.out.println("all done");

i is 0
i is 1
i is 2
for (int i = 0; i < 3; ++i) {
    System.out.println("i is " + i);
}
System.out.println("all done");
Execution Trace

```java
for (int i = 0; i < 3; ++i) {
    System.out.println("i is " + i);
}

System.out.println("all done");
```

i is 0
i is 1
i is 2
all done

Variable i has gone out of scope – it is local to the loop
Nested loops

int m = 2;
int n = 3;
for (int i = 0; i < n; ++i) {
    System.out.println("i is " + i);
    for (int j = 0; j < m; ++j) {
        System.out.println(" j is " + j);
    }
}

Nested loops

```java
int m = 2;
int n = 3;
for (int i = 0; i < n; ++i) {
    System.out.println("i is " + i);
    for (int j = 0; j < m; ++j) {
        System.out.println("   j is " + j);
    }
}
```

```
i is 0
   j is 0
   j is 1
i is 1
   j is 0
   j is 1
i is 2
   j is 0
   j is 1
```
The do-while statement

- **Syntax**
  
  do *Action*
  
  while (*Expression*)

- **Semantics**
  
  - Execute *Action*
  
  - If *Expression* is true then execute *Action* again
  
  - Repeat this process until *Expression* evaluates to false

- *Action* is either a single statement or a group of statements within braces
Picking off digits

- Consider
  ```java
  System.out.print("Enter a positive number: ");
  int number = stdin.nextInt());
  do {
      int digit = number % 10;
      System.out.println(digit);
      number = number / 10;
  } while (number != 0);
  ```

- Sample behavior
  Enter a positive number: 1129
  9
  2
  1
  1
Problem solving
<table>
<thead>
<tr>
<th>Importance of Internet per 100 people for 189 entities</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.09</td>
</tr>
<tr>
<td>0.16</td>
</tr>
<tr>
<td>1.44</td>
</tr>
<tr>
<td>41.3</td>
</tr>
<tr>
<td>5.67</td>
</tr>
<tr>
<td>1.86</td>
</tr>
<tr>
<td>14.43</td>
</tr>
<tr>
<td>0.7</td>
</tr>
<tr>
<td>0.39</td>
</tr>
<tr>
<td>40.25</td>
</tr>
<tr>
<td>22.66</td>
</tr>
<tr>
<td>2.74</td>
</tr>
<tr>
<td>21.67</td>
</tr>
<tr>
<td>2.81</td>
</tr>
<tr>
<td>3.57</td>
</tr>
<tr>
<td>0.1</td>
</tr>
<tr>
<td>2.7</td>
</tr>
<tr>
<td>7.73</td>
</tr>
<tr>
<td>11.09</td>
</tr>
<tr>
<td>25.86</td>
</tr>
<tr>
<td>3.17</td>
</tr>
</tbody>
</table>
Data set manipulation

- Often five values of particular interest
  - Minimum
  - Maximum
  - Mean
  - Standard deviation
  - Size of data set

- Let’s design a data set representation
What facilitators are needed?
Implication on facilitators

• public double getMinimum()
  – Returns the minimum value in the data set. If the data set is empty, then Double.NaN is returned, where Double.NaN is the Java double value representing the status not-a-number

• public double getMaximum()
  – Returns the maximum value in the data set. If the data set is empty, then Double.NaN is returned
Implication on facilitators

• public double getAverage()
  – Returns the average value in the data set. If the data set is empty, then Double.NaN is returned

• public double getStandardDeviation()
  – Returns the standard deviation value of the data set. If the data set is empty, then Double.NaN is returned

  • Left to the interested student

• public int getSize()
  – Returns the number of values in the data set being represented
What constructors are needed?
Constructors

- public DataSet()
  - Initializes a representation of an empty data set

- public DataSet(String s)
  - Initializes the data set using the values from the file with name s

- public DataSet(File file)
  - Initializes the data set using the values from the file
    - Left to interested student
Other methods

• public void addValue(double x)
  – Adds the value x to the data set being represented

• public void clear()
  – Sets the representation to that of an empty data set

• public void load(String s)
  – Adds the values from the file with name s to the data set being represented

• public void load(File file)
  – Adds the values from the file to the data set being represented
  • Left to interested student
What instance variables are needed?
Instance variables

- private int n
  - Number of values in the data set being represented

- private double minimumValue
  - Minimum value in the data set being represented

- private double maximumValue
  - Maximum value in the data set being represented

- private double xSum
  - The sum of values in the data set being represented
Example usage

DataSet dataset = new DataSet("age.txt");
System.out.println();
System.out.println("Minimum: " + dataset.getMinimum());
System.out.println("Maximum: " + dataset.getMaximum());
System.out.println("Mean: " + dataset.getAverage());
System.out.println("Size: " + dataset.getSize());
System.out.println();
dataset.clear();

dataset.load("stature.txt");
System.out.println("Minimum: " + dataset.getMinimum());
System.out.println("Maximum: " + dataset.getMaximum());
System.out.println("Mean: " + dataset.getAverage());
System.out.println("Size: " + dataset.getSize());
System.out.println();
dataset.clear();
Example usage

dataset.load("foot-length.txt");
System.out.println("Minimum: " + dataset.getMinimum());
System.out.println("Maximum: " + dataset.getMaxIMUM());
System.out.println("Mean: " + dataset.getAverage());
System.out.println("Size: " + dataset.getSize());
dataset.clear();
System.out.println("Minimum: " + dataset.getMinimum());
System.out.println("Maximum: " + dataset.getMaxIMUM());
System.out.println("Mean: " + dataset.getAverage());
System.out.println("Size: " + dataset.getSize());
Example usage

cmd: javac DataSet.java

cmd: javac DataSetTester.java

cmd: java DataSetTester

Minimum: 18.0
Maximum: 50.0
Mean: 26.182065217391305
Size: 2208

Minimum: 142.8
Maximum: 187.0
Mean: 162.93722826086938
Size: 2208

Minimum: 20.3
Maximum: 29.0
Mean: 24.438496376811592
Size: 2208

Minimum: NaN
Maximum: NaN
Mean: NaN
Size: 0

cmd:
Methods `getMinimum()` and `getMaximum()`

- Straightforward implementations given correct setting of instance variables

```java
public double getMinimum() {
    return minimumValue;
}
```

```java
public double getMaximum() {
    return maximumValue;
}
```
Method getSize()

- Straightforward implementations given correct setting of instance variables

```
public int getSize() {
    return n;
}
```
Method getAverage()

- Need to take into account that data set might be empty

```java
public double getAverage() {
    if (n == 0) {
        return Double.NaN;
    } else {
        return xSum / n;
    }
}
```
• Straightforward using clear() and load()

```
public DataSet() {
clear();
}
public DataSet(String s) throws IOException {
clear();
load(s);
}
```
Facilitator clear()

```java
public void clear() {
    n = 0;
    xSum = 0;
    minimumValue = Double.NaN;
    maximumValue = Double.NaN;
}
```
Facilitator add()

    public void addValue(double x) {
        xSum += x;
        ++n;
        if (n == 1) {
            minimumValue = maximumValue = x;
        }
        else if (x < minimumValue) {
            minimumValue = x;
        }
        else if (x > maximumValue) {
            maximumValue = x;
        }
    }
public void load(String s) throws IOException {
    // get a reader for the file
    Scanner fileIn = Scanner.create( new File(s) );

    // add values one by one
    while (fileIn.hasNext()) {
        double x = fileIn.nextDouble();
        addValue(x);
    }

    // close up file
    fileIn.close();
}
Programming with methods and classes
Methods

• Instance method
  – Operates on a object (i.e., and *instance* of the class)

    String s = new String("Help every cow reach its "
    + "potential!");
    int n = s.length(); ← Instance method

• Class method
  – Service provided by a class and it is not associated with a particular object

    String t = String.valueOf(n); ← Class method
Data fields

• Instance variable and instance constants
  – Attribute of a particular object
  – Usually a variable

    Point p = new Point(5, 5);
    int px = p.x; ← Instance variable

• Class variable and constants
  – Collective information that is not specific to individual objects of the class
  – Usually a constant

    Color favoriteColor = Color.MAGENTA;
    double favoriteNumber = Math.PI - Math.E;
Task – Conversion.java

- Support conversion between English and metric values
  - 1 gallon = 3.785411784 liters

- 1 mile = 1.609344 kilometers
- \( d \) degrees Fahrenheit = \( \frac{d - 32}{1.8} \) degrees Celsius
- 1 ounce (avdp) = 28.349523125 grams
- 1 acre = 0.0015625 square miles = 0.40468564 hectares
Conversion Implementation

public class Conversion {

    // conversion equivalencies
    private static final double LITERS_PER_GALLON = 3.785411784;
    private static final double KILOMETERS_PER_MILE = 1.609344;
    private static final double GRAMS_PER_OUNCE = 28.349523125;
    private static final double HECTARES_PER_ACRE = 0.40468564;
}
Conversion implementation

```
public static double gallonsToLiters(double g) {
    return gallons * LITERS_PER_GALLON;
}
```

Modifier public indicates other classes can use the method
Modifier static indicates the method is a class method

Observe there is no reference in the method to an attribute of an implicit Conversion object (i.e., a "this" object). This absence is a class method requirement. Class methods are invoked without respect to any particular object.
Conversion Implementation

// temperature conversions methods
public static double fahrenheitToCelsius(double f) {
    return (f - 32) / 1.8;
}

public static double celsiusToFahrenheit(double c) {
    return 1.8 * c + 32;
}

// length conversions methods
public static double kilometersToMiles(double km) {
    return km / KILOMETERS_PER_MILE;
}
Conversion Implementation

// mass conversions methods
public static double litersToGallons(double liters) {
    return liters / LITERS_PER_GALLON;
}

public static double gallonsToLiters(double gallons) {
    return gallons * LITERS_PER_GALLON;
}

public static double gramsToOunces(double grams) {
    return grams / GRAMS_PER_OUNCE;
}

public static double ouncesToGrams(double ounces) {
    return ounces * GRAMS_PER_OUNCE;
}
Conversion Implementation

// area conversions methods
public static double hectaresToAcres(double hectares) {
    return hectares / HECTARES_PER_ACRE;
}

public static double acresToHectares(double acres) {
    return acres * HECTARES_PER_ACRE;
}
Conversion use

Consider

Scanner stdin = Scanner.create(System.in);
System.out.print("Enter number of gallons: ");
double liters = stdin.nextDouble();
double gallons = Conversion.litersToGallons(liters);
System.out.println(gallons + " gallons = " + liters + " liters");

Produces

Number of gallons: 3.0
3.00 gallons = 11.356235351999999 liters
A preferred Conversion use

NumberFormat style = NumberFormat.getNumberInstance();

style.setMaximumFractionDigits(2);
style.setMinimumFractionDigits(2);

System.out.println(gallons + " gallons = " + style.format(liters) + " liters");

3.0 gallons = 11.36 gallons
Method invocations

- Actual parameters provide information that is otherwise unavailable
  
  ```java
double gallons = Conversion.litersToGallons(liters);
  ```

- When a method is invoked
  - Java sets aside *activation record* memory for that particular invocation
    - Activation record stores, among other things, the values of the formal parameters and local variables
  
  - Formal parameters initialized using the actual parameters
    - After initialization, the actual parameters and formal parameters are independent of each other

  - Flow of control is transferred temporarily to that method
public class Demo {
    public static double add(double x, double y) {
        double result = x + y;
        return result;
    }

    public static double multiply(double x, double y) {
        x = x * y;
        return x;
    }

    public static void main(String[] args) {
        double a = 8;
        double b = 11;

        double sum = add(a, b);
        System.out.println(a + " + " + b + " = " + sum);

        double product = multiply(a, b);
        System.out.println(a + " * " + b + " = " + product);
    }
}
Value parameter passing demonstration

multiply() does not change the actual parameter a
Demo.java walkthrough

double sum = add(a, b);

Initial values of formal parameters come from the actual parameters

public static double add(double x, double y) {
    double result = x + y
    return result;
}

<table>
<thead>
<tr>
<th>a</th>
<th>8.0</th>
</tr>
</thead>
<tbody>
<tr>
<td>b</td>
<td>11.0</td>
</tr>
<tr>
<td>sum</td>
<td>19.0</td>
</tr>
<tr>
<td>product</td>
<td>-</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>x</th>
<th>8.0</th>
</tr>
</thead>
<tbody>
<tr>
<td>y</td>
<td>11.0</td>
</tr>
<tr>
<td>result</td>
<td>19.0</td>
</tr>
</tbody>
</table>
Demo.java walkthrough

double multiply = multiply(a, b);

Initial values of formal parameters come from the actual parameters

public static double multiply(double x, double y) {
    x = x + y
    return x;
}

<table>
<thead>
<tr>
<th>main()</th>
<th>multiply()</th>
</tr>
</thead>
<tbody>
<tr>
<td>a: 8.0</td>
<td>x: 88.0</td>
</tr>
<tr>
<td>b: 11.0</td>
<td>y: 11.0</td>
</tr>
<tr>
<td>sum: 19.0</td>
<td></td>
</tr>
<tr>
<td>product: 88.0</td>
<td></td>
</tr>
</tbody>
</table>
public class PassingReferences {
    public static void f(Point v) {
        v = new Point(0, 0);
    }

    public static void g(Point v) {
        v.setLocation(0, 0);
    }

    public static void main(String[] args) {
        Point p = new Point(10, 10);
        System.out.println(p);
        f(p);
        System.out.println(p);
        g(p);
        System.out.println(p);
    }
}
PassingReferences.java run

```
java.awt.Point[x=10,y=10]
java.awt.Point[x=10,y=10]
java.awt.Point[x=0,y=0]
```

g() can change the attributes of the object to which p refers
public static void main(String[] args) {
    Point p = new Point(10, 10);
    System.out.println(p);
    f(p);
}

Method main()'s variable p and method f()'s formal parameter v have the same value, which is a reference to an object representing location (10, 10)

java.awt.Point[x=10,y=10]
public static void f(Point v) {
    v = new Point(0, 0);
}

Point x: 10
y: 10

Point x: 0
y: 0

PassingReferences.java
public static void main(String[] args) {
    Point p = new Point(10, 10);
    System.out.println(p);

    f(p);
    System.out.println(p);
    g(p);
}

Method main()'s variable p and method g()'s formal parameter v have the same value, which is a reference to an object representing location (10, 10)
public static void g(Point v) {
    v.setLocation(0, 0);
}

Method main()'s variable p and method g()'s formal parameter v have the same value, which is a reference to an object representing location (10, 10)
public static void main(String[] args) {
    Point p = new Point(10, 10);
    System.out.println(p);
    f(p);
    System.out.println(p);
    g(p);
    System.out.println(p);
}

java.awt.Point[x=10,y=10]
java.awt.Point[x=10,y=10]
java.awt.Point[x=0,y=0]
class Scope {
    public static void f(int a) {
        int b = 1;  // local definition
        System.out.println(a);  // print 10
        a = b;  // update a
        System.out.println(a);  // print 1
    }

    public static void main(String[] args) {
        int i = 10;  // local definition
        f(i);  // invoking f() with i as parameter
        System.out.println(a);
        System.out.println(b);
    }
}

Variables a and b do not exist in the scope of method main()
Blocks and scope rules

• A block is a list of statements nested within braces
  – A method body is a block
  – A block can be placed anywhere a statement would be legal
    • A block contained within another block is a nested block

• A formal parameter is considered to be defined at the beginning of the method body

• A local variable can be used only in a statement or nested blocks that occurs after its definition

• An identifier name can be reused as long as the blocks containing the duplicate declarations are not nested one within the other

• Name reuse within a method is permitted as long as the reuse occurs in distinct blocks
class Scope2 {
    public static void main(String[] args) {
        int a = 10;
        f(a);
        System.out.println(a);
    }

    public static void f(int a) {
        System.out.println(a);
        a = 1;
        System.out.println(a);
    }
}
Legal but not recommended

public void g() {
    {
        int j = 1;        // define j
        System.out.println(j);      // print 1
    }
    {
        int j = 10;       // define a different j
        System.out.println(j);  // print 10
    }
    {
        char j = '@';     // define a different j
        System.out.println(j); // print '@'
    }
}
What’s the output?

```java
for (int i = 0; i < 3; ++i) {
    int j = 0;
    ++j;
    System.out.println(j);
}
```

- The scope of variable `j` is the body of the for loop
  - `j` is not in scope when `++i`
  - `j` is not in scope when `i < 3` are evaluated
  - `j` is redefined and re-initialized with each loop iteration
Task – Triple.java

- Represent objects with three integer attributes

- What constructors should we have?

- What accessors and mutators should we have?

- What facilitators should we have?
Task – Triple.java

- public Triple()
  - Constructs a default Triple value representing three zeros

- public Triple(int a, int b, int c)
  - Constructs a representation of the values a, b, and c
Task – Triple.java

- public int getValue(int i)
  - Returns the i-th element of the associated Triple

- public void setValue(int i, int value)
  - Sets the i-th element of the associated Triple to value
Task – Triple.java

- public String toString()
  - Returns a textual representation of the associated Triple

- public Object clone()
  - Returns a new Triple whose representation is the same as the associated Triple

- public boolean equals(Object v)
  - Returns whether v is equivalent to the associated Triple

These three methods are overrides of inherited methods
Triple.java implementation

// Triple(): specific constructor
public Triple(int a, int b, int c) {
    setValue(1, a);
    setValue(2, b);
    setValue(3, c);
}

Triple.java implementation

// Triple(): specific constructor - alternative definition
public Triple(int a, int b, int c) {
    this.setValue(1, a);
    this.setValue(2, b);
    this.setValue(3, c);
}
Triple.java implementation

// Triple(): default constructor
public Triple() {
    this(0, 0, 0);
}

The new Triple object (the this object) is constructed by invoking the Triple constructor expecting three int values as actual parameters.

public Triple() {
    int a = 0;
    int b = 0;
    int c = 0;
    this(a, b, c);
}

Illegal this() invocation. A this() invocation must begin its statement body.
Triple.java implementation

- Class Triple like every other Java class
  - Automatically an extension of the standard class Object
  - Class Object specifies some basic behaviors common to all objects
    - These behaviors are said to be inherited
  - Three of the inherited Object methods
    - toString()
    - clone()
    - equals()
Recommendation

- Classes should override (i.e., provide a class-specific implementation)
  - toString()
  - clone()
  - equals()

- By doing so, the programmer-expected behavior can be provided

  System.out.println(p);  // displays string version of
                          // object referenced by p
  System.out.println(q);  // displays string version of
                          // object referenced by q
Triple.java toString() implementation

public String toString() {
    int a = getValue(1);
    int b = getValue(2);
    int c = getValue(3);

    return "Triple[" + a + ", " + b + ", " + c + "]";
}

• Consider

    Triple t1 = new Triple(10, 20, 30);
    System.out.println(t1);

    Triple t2 = new Triple(8, 88, 888);
    System.out.println(t2);

• Produces

    Triple[10, 20, 30]
    Triple[8, 88, 888]
**Triple.java clone() implementation**

```java
public Object clone() {
    int a = getValue(1);  // Return type is Object
    int b = getValue(2);
    int c = getValue(3);

    return new Triple(a, b, c);
}
```

- **Consider**
  ```java
  Triple t1 = new Triple(9, 28, 29);
  Triple t2 = (Triple) t1.clone();  // Must cast!

  System.out.println("t1 = " + t1);
  System.out.println("t2 = " + t2);
  ```

- **Produces**
  ```java
  Triple[9, 28, 29]
  Triple[9, 28, 29]
  ```
public boolean equals(Object v) {
    if (v instanceof Triple) {
        int a1 = getValue(1);
        int b1 = getValue(2);
        int c1 = getValue(3);

        Triple t = (Triple) v;
        int a2 = t.getValue(1);
        int b2 = t.getValue(2);
        int c2 = t.getValue(3);

        return (a1 == a2) && (b1 == b2) && (c1 == c2);
    } else {
        return false;
    }
}
Triple.java equals()

```java
Triple e = new Triple(4, 6, 10);
Triple f = new Triple(4, 6, 11);
Triple g = new Triple(4, 6, 10);
Triple h = new Triple(4, 5, 11);
boolean flag1 = e.equals(f);
```
Triple.java equals()

```java
Triple e = new Triple(4, 6, 10);
Triple f = new Triple(4, 6, 11);
Triple g = new Triple(4, 6, 10);
Triple h = new Triple(4, 5, 11);
boolean flag2 = e.equals(g);
```
```java
Triple e = new Triple(4, 6, 10);
Triple f = new Triple(4, 6, 11);
Triple g = new Triple(4, 6, 10);
Triple h = new Triple(4, 5, 11);
boolean flag3 = g.equals(h);
```
Overloading

• Have seen it often before with operators
  
  ```
  int i = 11 + 28;
  double x = 6.9 + 11.29;
  String s = "April " + "June";
  ```

• Java also supports method overloading
  
  – Several methods can have the same name
  – Useful when we need to write methods that perform similar tasks but different parameter lists
  – Method name can be overloaded as long as its signature is different from the other methods of its class
    
    • Difference in the names, types, number, or order of the parameters
public static int min(int a, int b, int c) {
    return Math.min(a, Math.min(b, c));
}

public static int min(int a, int b, int c, int d) {
    return Math.min(a, min(b, c, d));
}
public static int power(int x, int n) {
    int result = 1;
    for (int i = 1; i <= n; ++i) {
        result *= x;
    }
    return result;
}

public static double power(double x, int n) {
    double result = 1;
    for (int i = 1; i <= n; ++i) {
        result *= x;
    }
    return result;
}
What’s the output?

```java
public static void f(int a, int b) {
    System.out.println(a + b);
}

public static void f(double a, double b) {
    System.out.println(a - b);
}

public static void main(String[] args) {
    int i = 19;
    double x = 54;

    f(i, x);
}
```
Arrays
Consider arrays first. Java provides arrays and the collection classes. A list may be one-dimensional or multi-dimensional. Values are a list. Programmer often need the ability to represent a group of.
Basic terminology

- List is composed of *elements*
- Elements in a list have a *common name*
- The list as a whole is referenced through the common name
- List elements are of the same type — the base type
- Elements of a list are referenced by *subscripting* (indexing) the common name
Java array features

- Has features common to all other objects
- Array is an object
- Data field length specifies the number of elements in the list
- Ensures any reference to an array element is valid
- Automatic bounds checking
- Where n is the number of elements
- Index type is integer and the index range must be 0 ... n-1
- Size of array can be specified at run time
- Base (element) type can be any type
- Subscripts are denoted as expressions within brackets: [ ]

Java array features

- Subscripts are denoted as expressions within brackets: [ ]
Array variable definition styles

• Without initialization

```
ElementType[] id ;
```

Type of values in list

Brackets indicate array variable being defined

Name of list
Array variable definition styles

- With initialization

```java
ElementType[] id = new ElementType[n];
```

Nonnegative integer expression specifying the number of elements in the array

Reference to a new array of n elements
Example

• Definitions
  ```java
  char[] c;
  int[] value = new int[10];
  ```

• Causes
  - Array object variable `c` is un-initialized
  - Array object variable `v` references a new ten element list of integers
    • Each of the integers is default initialized to 0
Consider

```java
int[] v = new int[10];
int i = 7;
int j = 2;
int k = 4;
v[0] = 1;
v[i] = 5;
v[j] = v[i] + 3;
v[j+1] = v[i] + v[0];
v[v[j]] = 12;
System.out.println(v[2]);
v[k] = stdin.nextInt();
```

v

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
</tr>
</thead>
</table>
Consider

```java
int[] v = new int[10];
int i = 7;
int j = 2;
int k = 4;
v[0] = 1;
v[i] = 5;
v[j] = v[i] + 3;
v[j+1] = v[i] + v[0];
v[v[j]] = 12;
System.out.println(v[2]);
v[k] = stdin.nextInt();
```

![Array representation]

```
v  1 0 0 0 0 0 0 0 0 0
```
Consider

```java
int[] v = new int[10];
int i = 7;
int j = 2;
int k = 4;
v[0] = 1;
v[i] = 5;
v[j] = v[i] + 3;
v[j+1] = v[i] + v[0];
v[v[j]] = 12;
System.out.println(v[2]);
v[k] = stdin.nextInt();
```

```
   v  1  0  0  0  0  0  0  0  5  0  0
```
Consider

```java
int[] v = new int[10];
int i = 7;
int j = 2;
int k = 4;
v[0] = 1;
v[i] = 5;
v[j] = v[i] + 3;
v[j+1] = v[i] + v[0];
v[v[j]] = 12;
System.out.println(v[2]);
v[k] = stdin.nextInt();
```
Consider

```java
int[] v = new int[10];
int i = 7;
int j = 2;
int k = 4;
v[0] = 1;
v[i] = 5;
v[j] = v[i] + 3;
v[j+1] = v[i] + v[0];
v[v[j]] = 12;
System.out.println(v[2]);
v[k] = stdin.nextInt();
```

```
   1     0     8     6     0     0     0     5     0     0
```
Consider

```java
int[] v = new int[10];
int i = 7;
int j = 2;
int k = 4;
v[0] = 1;
v[i] = 5;
v[j] = v[i] + 3;
v[j+1] = v[i] + v[0];
v[v[j]] = 12;
System.out.println(v[2]);
v[k] = stdin.nextInt();
```

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>0</th>
<th>8</th>
<th>6</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>5</th>
<th>12</th>
<th>0</th>
</tr>
</thead>
</table>
Consider

int[] v = new int[10];
int i = 7;
int j = 2;
int k = 4;
v[0] = 1;
v[i] = 5;
v[j] = v[i] + 3;
v[j+1] = v[i] + v[0];
v[v[j]] = 12;
System.out.println(v[2]);
8 is displayed
v[k] = stdin.nextInt();

<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>v</td>
<td>1</td>
<td>0</td>
<td>8</td>
<td>6</td>
<td>0</td>
</tr>
</tbody>
</table>
Consider

```java
int[] v = new int[10];
int i = 7;
int j = 2;
int k = 4;
v[0] = 1;
v[i] = 5;
v[j] = v[i] + 3;
v[j+1] = v[i] + v[0];
v[v[j]] = 12;
System.out.println(v[2]);
v[k] = stdin.nextInt();
```

Suppose 3 is extracted
Consider

- Segment
  ```java
  int[] b = new int[100];
  b[-1] = 0;
  b[100] = 0;
  ```

- Causes
  - Array variable to reference a new list of 100 integers
    - Each element is initialized to 0
  - Two exceptions to be thrown
    - -1 is not a valid index – too small
    - 100 is not a valid index – too large
      - IndexOutOfBoundsException
Consider

```java
Point[] p = new Point[3];
p[0] = new Point(0, 0);
p[1] = new Point(1, 1);
p[2] = new Point(2, 2);
p[0].setX(1);
p[1].setY(p[2].getY());
Point vertex = new Point(4, 4);
p[1] = p[0];
p[2] = vertex;
```
Consider

```
Point[] p = new Point[3];
p[0] = new Point(0, 0);
p[1] = new Point(1, 1);
p[2] = new Point(2, 2);
p[0].setX(1);
p[1].setY(p[2].getY());
Point vertex = new Point(4,4);
p[1] = p[0];
p[2] = vertex;
```

Consider

Point[] p = new Point[3];
p[0] = new Point(0, 0);
p[1] = new Point(1, 1);
p[2] = new Point(2, 2);
p[0].setX(1);
p[1].setY(p[2].getY());
Point vertex = new Point(4,4);
p[1] = p[0];
p[2] = vertex;
Consider

Point[] p = new Point[3];
p[0] = new Point(0, 0);
p[1] = new Point(1, 1);
p[2] = new Point(2, 2);
p[0].setX(1);
p[1].setY(p[2].getY());
Point vertex = new Point(4,4);
p[1] = p[0];
p[2] = vertex;
Consider

Point[] p = new Point[3];
p[0] = new Point(0, 0);
p[1] = new Point(1, 1);
p[2] = new Point(2, 2);
p[0].setX(1);
p[1].setY(p[2].getY());
Point vertex = new Point(4, 4);
p[1] = p[0];
p[2] = vertex;
Consider

Point[] p = new Point[3];
p[0] = new Point(0, 0);
p[1] = new Point(1, 1);
p[2] = new Point(2, 2);
p[0].setX(1);
p[1].setY(p[2].getY());
Point vertex = new Point(4, 4);
p[1] = p[0];
p[2] = vertex;
Consider

Point[] p = new Point[3];
p[0] = new Point(0, 0);
p[1] = new Point(1, 1);
p[2] = new Point(2, 2);
p[0].setX(1);
p[1].setY(p[2].getY());
Point vertex = new Point(4, 4);
p[1] = p[0];
p[2] = vertex;

Point: (1, 0)
Point: (4, 4)
Explicit initialization

- Syntax

```c
ElementType[] id = { exp_0, exp_1, ... exp_{n-1} };
```

id references an array of n elements. id[0] has value \(\text{exp}_0\), id[1] has value \(\text{exp}_1\), and so on. Each \(\text{exp}_i\) is an expression that evaluates to type ElementType.
Explicit initialization

- Example
  ```java
  String[] puppy = { "nilla", "darby", "galen", "panther" };  
  int[] unit = { 1 };  
  ```

- Equivalent to
  ```java
  String[] puppy = new String[4];  
  
  int[] unit = new int[1];  unit[0] = 1;  ```
Array members

- Member length
  - Size of the array
    ```java
    for (int i = 0; i < puppy.length; ++i) {
        System.out.println(puppy[i]);
    }
    ```
Array members

• Member clone()
  – Produces a shallow copy
    ```
    Point[] u = { new Point(0, 0), new Point(1, 1)};
    Point[] v = u.clone();
    v[1] = new Point(4, 30);
    ```
Array members

• Member clone()
  – Produces a shallow copy

```java
Point[] u = { new Point(0, 0), new Point(1, 1)};
Point[] v = u.clone();
v[1] = new Point(4, 30);
```

![Diagram showing the shallow copy of an array of Point objects]
Array members

- Member clone()
  - Produces a shallow copy
    ```java
    Point[] u = { new Point(0, 0), new Point(1, 1)};
    Point[] v = u.clone();
    v[1] = new Point(4, 30);
    ```
Making a deep copy

- Example

```java
Point[] w = new Point[u.length];
for (int i = 0; i < u.length; ++i) {
    w[i] = u[i].clone();
}
```
Making a deep copy
Searching for a value

```java
System.out.println("Enter search value (number): ");
int key = stdin.nextInt();

int i;
for (i = 0; i < data.length; ++i) {
    if (key == data[i]) {
        break;
    }
}

if (i != data.length) {
    System.out.println(key + " is the " + I + "-th element");
} else {
    System.out.println(key + " is not in the list");
}
```
Searching for a value

System.out.println("Enter search value (number): ");
int key = stdin.nextInt();

int i;
for (i = 0; i < data.length; ++i) {
    if (key == data[i]) {
        break;
    }
}
if (i != data.length) {
    System.out.println(key + " is the " + I + "-th element");
} else {
    System.out.println(key + " is not in the list");
}
Searching for a value

System.out.println("Enter search value (number): ");
int key = stdin.nextInt();

int i;
for (i = 0; i < data.length; ++i) {
    if (key == data[i]) {
        break;
    }
}

if (i != data.length) {
    System.out.println(key + " is the " + i + "-th element");
} else {
    System.out.println(key + " is not in the list");
}
Searching for a value

System.out.println("Enter search value (number): ");
int key = stdin.nextInt();

int i;
for (i = 0; i < data.length; ++i) {
    if (key == data[i]) {
        break;
    }
}
if (i != data.length) {
    System.out.println(key + " is the " + i + "-th element");
} else {
    System.out.println(key + " is not in the list");
}
Searching for a value

```java
System.out.println("Enter search value (number): ");
int key = stdin.nextInt();

int i;
for (i = 0; i < data.length; ++i) {
    if (key == data[i]) {
        break;
    }
}
if (i != data.length) {
    System.out.println(key + " is the " + I + "-th element");
} else {
    System.out.println(key + " is not in the list");
}
```
Searching for a value

System.out.println("Enter search value (number): ");
int key = stdin.nextInt();

int i;
for (i = 0; i < data.length; ++i) {
    if (key == data[i]) {
        break;
    }
}

if (i != data.length) {
    System.out.println(key + " is the " + i + "-th element");
} else {
    System.out.println(key + " is not in the list");
}
Searching for a value

System.out.println("Enter search value (number): ");
int key = stdin.nextInt();

int i;
for (i = 0; i < data.length; ++i) {
    if (key == data[i]) {
        break;
    }
}

if (i != data.length) {
    System.out.println(key + " is the " + i + "-th element");
}
else {
    System.out.println(key + " is not in the list");
}
Searching for a value

System.out.println("Enter search value (number): ");
int key = stdin.nextInt();

int i;
for (i = 0; i < data.length; ++i) {
    if (key == data[i]) {
        break;
    }
}

if (i != data.length) {
    System.out.println(key + " is the " + I + "-th element");
} else {
    System.out.println(key + " is not in the list");
}
Searching for a value

System.out.println("Enter search value (number): ");
int key = stdin.nextInt();

int i;
for (i = 0; i < data.length; ++i) {
    if (key == data[i]) {
        break;
    }
}
if (i != data.length) {
    System.out.println(key + " is the " + i + "-th element");
} else {
    System.out.println(key + " is not in the list");
}
Searching for a value

System.out.println("Enter search value (number): ");
int key = stdin.nextInt();

int i;
for (i = 0; i < data.length; ++i) {
    if (key == data[i]) {
        break;
    }
}

if (i != data.length) {
    System.out.println(key + " is the " + I + "-th element");
}
else {
    System.out.println(key + " is not in the list");
}
Searching for a value

System.out.println("Enter search value (number): ");
int key = stdin.nextInt();

int i;
for (i = 0; i < data.length; ++i) {
    if (key == data[i]) {
        break;
    }
}
if (i != data.length) {
    System.out.println(key + " is the " + i + "-th element");
}
else {
    System.out.println(key + " is not in the list");
}
Searching for a value

```java
System.out.println("Enter search value (number): ");
int key = stdin.nextInt();

int i;
for (i = 0; i < data.length; ++i) {
    if (key == data[i]) {
        break;
    }
}
if (i != data.length) {
    System.out.println(key + " is the " + I + "-th element");
} else {
    System.out.println(key + " is not in the list");
}
```
Searching for a value

```java
System.out.println("Enter search value (number): ");
int key = stdin.nextInt();

int i;
for (i = 0; i < data.length; ++i) {
    if (key == data[i]) {
        break;
    }
}

if (i != data.length) {
    System.out.println(key + " is the " + I + "-th element");
} else {
    System.out.println(key + " is not in the list");
}
```
Searching for the minimum value

- Segment

```java
int minimumSoFar = sample[0];
for (int i = 1; i < sample.length; ++i) {
    if (sample[i] < minimumSoFar) {
        minimumSoFar = sample[i];
    }
}
```
ArrayTools.java method
sequentialSearch()

```java
public static int sequentialSearch(int[] data, int key) {
    for (int i = 0; i < data.length; ++i) {
        if (data[i] == key) {
            return i;
        }
    }
    return -1;
}
```

- Consider
  ```java
  int[] score = { 6, 9, 82, 11, 29, 85, 11, 28, 91 }; 
  int i1 = sequentialSearch(score, 11); 
  int i2 = sequentialSearch(score, 30); 
  ```
ArrayTools.java method

sequentialSearch()

```java
public static int sequentialSearch(int[] data, int key) {
    for (int i = 0; i < data.length; ++i) {
        if (data[i] == key) {
            return i;
        }
    }
    return -1;
}
```

Consider

```java
int[] score = { 6, 9, 82, 11, 29, 85, 11, 28, 91 };
int i1 = sequentialSearch(score, 11);
int i2 = sequentialSearch(score, 30);
```
ArrayTools.java method putList()

```java
public static void putList(int[] data) {
    for (int i = 0; i < data.length; ++i) {
        System.out.println(data[i]);
    }
}
```

- Consider
  ```java
  int[] score = { 6, 9, 82, 11, 29, 85, 11, 28, 91 }; 
  putList(score);
  ```
public static int[] getList() {
    Scanner stdin = Scanner.create(System.in);
    int[] buffer = new int[MAX_LIST_SIZE];
    int listSize = 0;
    for (int i = 0; (stdin.hasNextInt()) && i < MAX_LIST_SIZE; ++i) {
        buffer[i] = stdin.nextInt();
        ++listSize;
    }
    int[] data = new int[listSize];
    for (int i = 0; i < listSize; ++i) {
        data[i] = buffer[i];
    }
    return data;
}
public class ArrayTools {

    // class constant
    private static final int MAX_LIST_SIZE = 1000;

    // sequentialSearch(): examine unsorted list for key
    public static int binarySearch(int[] data, int key) { ... }

    // valueOf(): produces a string representation
    public static void putList(int[] data) { ... }

    // getList(): extract and return up to MAX_LIST_SIZE values
    public static int[] getList() throws IOException { ... }

    // reverse(): reverses the order of the element values
    public static void reverse(int[] list) { ... }

    // binarySearch(): examine sorted list for a key
    public static int binarySearch(char[] data, char key) { ... }
}


import java.util.*;

public class Demo {
    // main(): application entry point
    public static void main(String[] args) {
        System.out.println("\n");
        System.out.println("Enter list of integers: ");
        int[] number = ArrayTools.getList();

        System.out.println("\n");
        System.out.println("Your list");
        ArrayTools.putList(number);

        ArrayTools.reverse(number);
        System.out.println("\n");
        System.out.println("Your list in reverse");
        ArrayTools.putList(number);
        System.out.println();
    }
}
cmd: javac ArrayTools.java

cmd: javac Demo.java

cmd: java Demo

Enter list of integers, one per line:
12
11
10
AZ

Your list
12
11
10

Your list in reverse
10
11
12

cmd:
Sorting

• Problem
  – Arranging elements so that they are ordered according to some desired scheme
    • Standard is non-decreasing order
      – Why don't we say increasing order?

• Major tasks
  – Comparisons of elements
  – Updates or element movement
Selection sorting

- Algorithm basis
  - On iteration $i$, a selection sorting method
    - Finds the element containing the $i$th smallest value of its list $v$ and exchanges that element with $v[i]$

- Example – iteration 0
  - Swaps smallest element with $v[0]$
  - This results in smallest element being in the correct place for a sorted result
Selection sorting

• Algorithm basis
  – On iteration i, a selection sorting method
    • Finds the element containing the ith smallest value of its list $v$ and exchanges that element with $v[i]$

• Example – iteration 0
  – Swaps smallest element with $v[0]$
  – This results in smallest element being in the correct place for a sorted result

\[
\begin{array}{ccccccccccc}
0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\
\hline
E & W & Q & R & T & Y & U & I & O & P \\
\end{array}
\]
Selection sorting

- Algorithm basis
  - On iteration $i$, a selection sorting method
    - Finds the element containing the $i$th smallest value of its list $v$ and exchanges that element with $v[i]$

- Example – iteration 1
  - Swaps second smallest element with $v[1]$
  - This results in second smallest element being in the correct place for a sorted result
Selection sorting

- Algorithm basis
  - On iteration i, a selection sorting method
  - Finds the element containing the ith smallest value of its list v and exchanges that element with v[i]

- Example – iteration 1
  - Swaps second smallest element with v[1]
  - This results in second smallest element being in the correct place for a sorted result
public static void selectionSort(char[] v) {
    for (int i = 0; i < v.length - 1; ++i) {
        // guess the location of the ith smallest element
        int guess = i;
        for (int j = i + 1; j < v.length; ++j) {
            if (v[j] < v[guess]) { // is guess ok?
                // update guess to index of smaller element
                guess = j;
            }
        }
        // guess is now correct, so swap elements
        char rmbr = v[i];
        v[i] = v[guess];
        v[guess] = rmbr;
    }
}
Iteration i

// guess the location of the ith smallest element
int guess = i;
for (int j = i+1; j < v.length; ++j) {
    if (v[j] < v[guess]) // is guess ok?
        // update guess with index of smaller element
        guess = j;
}

// guess is now correct, swap elements v[guess] and v[0]
Multidimensional arrays

- Many problems require information be organized as a two-dimensional or multidimensional list

Examples
  - Matrices
  - Graphical animation
  - Economic forecast models
  - Map representation
  - Time studies of population change
  - Microprocessor design
Example

- Segment
  ```java
  int[][][] m = new int[3][];
  m[0] = new int[4];
  m[1] = new int[4];
  m[2] = new int[4];
  ```
- Produces
Example

- Segment
  ```java
  for (int r = 0; r < m.length; ++r) {
    for (int c = 0; c < m[r].length; ++c) {
      System.out.print("Enter a value: ");
      m[r][c] = stdin.nextInt();
    }
  }
  ```
Example

- Segment

  ```java
  String[][] s = new String[4][];
  s[0] = new String[2];
  s[1] = new String[2];
  s[2] = new String[4];
  s[3] = new String[3];
  ```

- Produces
Example

- Segment
  \[
  \text{int } c[][] = \{\{1, 2\}, \{3, 4\}, \{5, 6\}, \{7, 8, 9\}\};
  \]

- Produces
Matrices

- A two-dimensional array is sometimes known as a matrix because it resembles that mathematical concept.

- A matrix $a$ with $m$ rows and $n$ columns is represented mathematically in the following manner:

\[
\begin{bmatrix}
  a_{1,1} & a_{1,2} & \cdots & a_{1,n} \\
  a_{2,1} & a_{2,2} & \cdots & a_{2,n} \\
  \vdots & \vdots & \ddots & \vdots \\
  a_{m,1} & a_{m,2} & \cdots & a_{m,n}
\end{bmatrix}
\]
Matrix addition

- Definition \( C = A + B \)
  
  \[ c_{ij} = a_{1i}b_{1j} + a_{i2}b_{2j} + \ldots + a_{in}b_{nj} \]

  - \( c_{ij} \) is sum of terms produced by multiplying the elements of a’s row \( i \) with b’s column \( c \)
Matrix addition

```java
public static double[][][] add(double[][][] a, double[][][] b) {
    // determine number of rows in solution
    int m = a.length;
    // determine number of columns in solution
    int n = a[0].length;
    // create the array to hold the sum
    double[][][] c = new double[m][n];
    // compute the matrix sum row by row
    for (int i = 0; i < m; ++i) {
        // produce the current row
        for (int j = 0; j < n; ++j) {
            c[i][j] = a[i][j] + b[i][j];
        }
    }
    return c;
}
```
Inheritance and Polymorphism
Inheritance

- Organizes objects in a top-down fashion from most general to least general
- Inheritance defines a “is-a” relationship
  - A mountain bike “is a” kind of bicycle
  - A SUV “is a” kind of automobile
  - A border collie “is a” kind of dog
  - A laptop “is a” kind of computer
Musical instrument hierarchy

Musical Instruments

- Idiophones
- Membranophones
- Aerophones
- Chordophones
- Electrophones

Lip vibrating
Split column
Reeded

Clarinets
Saxophones
Bassoons
Musical instrument hierarchy

- The hierarchy helps us understand the relationships and similarities of musical instruments
  - A clarinet “is a” kind of reeded instrument
  - Reeded instruments “are a” kind of aerophone
- The “is-a” relationship is transitive
  - A clarinet “is a” kind of reeded instrument
  - A reeded instrument “is a” kind of aerophone
  - A clarinet “is a” kind of aerophone
Object-oriented terminology

- In object-oriented programming languages, a class created by extending another class is called a *subclass*
- The class used for the basis is called the *superclass*
- Alternative terminology
  - The superclass is also referred to as the *base* class
  - The subclass is also referred to as the *derived* class
Build a new class `ThreeDimensionalPoint` using inheritance:

- `ThreeDimensionalPoint` extends the `Point` class
- `Point` is the superclass (base class)
- `ThreeDimensionalPoint` is the subclass (derived class)
- `ThreeDimensionalPoint` extends `Point` by adding a new property—a z-coordinate

(x, y, z)
Class ThreeDimensionalPoint

```java
package geometry;

import java.awt.*;

public class ThreeDimensionalPoint extends Point {
    private final static int DEFAULT_Z = 0;

    public int z = DEFAULT_Z;
}
```

Keyword `extends` indicates that `ThreeDimensionalPoint` is a subclass of `Point`.

New instance variable:
```
public int z = DEFAULT_Z;
```
Allow definitions to be collected together into a single entity—a package.

- ThreeDimensionalPoint will be added to the geometry package.

- Classes and names in the same package are stored in the same folder.

- Classes in a package go into their own namespace and therefore the names in a particular package do not conflict with other names in other packages.

- For example, a package called Graph might have a different definition of ThreeDimensionalPoint.

- When defining members of a class or interface, Java does not require an explicit access specification. The implicit specification is known as default access. Members of a class with default access can be accessed only by members of the package.
Java’s Mother-of-all-objects—Class Object

Three Dimensional Point
ThreeDimensionalPoint

ThreeDimensionalPoint a =
    new ThreeDimensionalPoint(6, 21, 54);
a.translate(1, 1);     // invocation of superclass translate()
a.translate(2, 2, 2); // invocation of subclass translate()

• Java determines which method to use based on the number of parameters in the invocation
• After the first call to translate, what is the value of a?
• After the second call to translate, what is the value of a?
ThreeDimensionalPoint

- Methods `toString()`, `equals()`, and `clone()` should not have different signatures from the Point versions.

```java
ThreeDimensionalPoint c = new ThreeDimensionalPoint(1, 4, 9);

ThreeDimensionalPoint d = (ThreeDimensionalPoint) c.clone();

String s = c.toString();

boolean b = c.equals(d);
```

Cast is necessary as return type of subclass method `clone()` is `Object`.

Invocation of subclass `toString()` method.

Invocation of subclass `equals()` method.
ThreeDimensionalPoint

- Constructors

  // ThreeDimensionalPoint(): default constructor
  public ThreeDimensionalPoint() {
    super();
  }

  // ThreeDimensionalPoint(): specific constructor
  public ThreeDimensionalPoint(int a, int b, int c) {
    super(a, b);
    setZ(c);
  }
ThreeDimensionalPoint

- Accessors and mutators

  // getZ(): z-coordinate accessor
  public double getZ() {
      return z;
  }

  // setZ(): y-coordinate mutator
  public void setZ(int value) {
      z = value;
  }
ThreeDimensionalPoint

- Facilitators

```java
// translate(): shifting facilitator
public void translate(int dx, int dy, int dz) {
    translate(dx, dy);

    int zValue = (int) getZ();

    setZ(zValue + dz);
}
```
ThreeDimensionalPoint

- Facilitators

    // toString(): conversion facilitator
    public String toString() {
        int a = (int) getX();
        int b = (int) getY();
        int c = (int) getZ();
        return getClass() +
                "[" + a + ", " + b + ", " + c + "]";
    }
ThreeDimensionalPoint

- Facilitators

```java
// equals(): equality facilitator
public boolean equals(Object v) {
    if (v instanceof ThreeDimensionalPoint) {
        ThreeDimensionalPoint p =
            (ThreeDimensionalPoint) v;
        int z1 = (int) getZ();
        int z2 = (int) p.getZ();

        return super.equals(p) && (z1 == z2);
    } else {
        return false;
    }
}
}
ThreeDimensionalPoint

• Facilitators
  
  // clone(): clone facilitator
  public Object clone() {
      int a = (int) getX();
      int b = (int) getY();
      int c = (int) getZ();

      return new ThreeDimensionalPoint(a, b, c);
  }
ColoredPoint

- Suppose an application calls for the use of colored points.
- We can naturally extend class Point to create ColoredPoint
- Class ColoredPoint will be added to package geometry

```java
package geometry;

import java.awt.*;

public class ColoredPoint extends Point {
    // instance variable
    Color color;

    ...
```
ColoredPoint

- Constructors
  // ColoredPoint(): default constructor
  public ColoredPoint() {
    super();
    setColor(Color.blue);
  }

  // ColoredPoint(): specific constructor
  public ColoredPoint(int x, int y, Color c) {
    super(x, y);
    setColor(c);
  }
ColoredPoint

- Accessors and mutators

  \[\text{getColor(): color property accessor}\]
  \[
  \text{public Color getColor()} \{
      \text{return color;}
  \}
  \]

  \[\text{setColor(): color property mutator}\]
  \[
  \text{public void setColor(Color c)} \{
      \text{color = c;}
  \}
  \]
ColoredPoint

- Facilitators

  // clone(): clone facilitator
  public Object clone() {
      int a = (int) getX();
      int b = (int) getY();
      Color c = getColor();
      return new ColoredPoint(a, b, c);
  }
ColoredPoint

- Facilitators

  // toString(): string representation facilitator
  public String toString() {
    int a = (int) getX();
    int b = (int) getY();
    Color c = getColor();
    return getClass() +
           "[" + a + ", " + b + ", " + c + "]";
  }

ColoredPoint

- Facilitators
  // toString(): string representation facilitator

```java
public String toString() {
    int a = (int) getX();
    int b = (int) getY();
    Color c = getColor();
    return getClass() +
           "[" + a + ", " + b + ", " + c + "]";
}
```
ColoredPoint

- Facilitators

  // equals(): equal facilitator
  public boolean equals(Object v) {
    if (v instanceof ColoredPoint) {
      Color c1 = getColor();
      Color c2 = ((ColoredPoint) v).getColor();
      return super.equals(v) && c1.equals(c2);
    }
    else {
      return false;
    }
  }

Colored3DPoint

- Suppose an application needs a colored, three-dimensional point.
- Can we create such a class by extending both ThreeDimensionalPoint and ColoredPoint?
  - No. Java does not support multiple inheritance
  - Java only supports single inheritance

```java
package Geometry;
import java.awt.*;

public class Colored3DPoint extends ThreeDimensionalPoint {
    // instance variable
    Color color;
}
```
Colored3DPoint

• Constructors
  
  // Colored3DPoint(): default constructor
  public Colored3DPoint() {
      setColor(Color.blue);
  }

  // Colored3DPoint(): specific constructor
  public Colored3DPoint(int a, int b, int c, Color d) {
      super(a, b, c);
      setColor(d);
  }
Colored3DPoint

- Accessors and mutators
  
  // getColor(): color property accessor
  public Color getColor() {
    return color;
  }

  // setColor(): color property mutator
  public void setColor(Color c) {
    color = c;
  }
class Colored3DPoint {
    // Clone facilitator
    public Object clone() {
        int a = (int) getX();
        int b = (int) getY();
        int c = (int) getZ();
        Color d = getColor();
        return new Colored3DPoint(a, b, c, d);
    }
    // Facilitators
    public Facilitators() {
    }
}
Colored3DPoint

- Facilitators

  // toString(): string representation facilitator
  public String toString() {
      int a = (int) getX();
      int b = (int) getY();
      int c = (int) getZ();
      Color d = getColor();
      return getClass() +
          "[" + a + ", " + b + ", " + c + ", " + d + "]";
  }

Colored3DPoint

• Facilitators

    // equals(): equal facilitator
    public boolean equals(Object v) {
        if (v instanceof Colored3DPoint) {
            Color c1 = getColor();
            Color c2 = ((Colored3DPoint) v).getColor();
            return super.equals(v) && c1.equals(c2);
        }
        else {
            return false;
        }
    }
Polymorphism

• A code expression can invoke different methods depending on the types of objects being manipulated
• Example: function overloading like method min() from java.lang.Math
  - The method invoked depends on the types of the actual arguments

  Example

  ```java
  int a, b, c;
double x, y, z;
...
c = min(a, b); // invokes integer min()
z = min(x, y); // invokes double min
  ```
Polymorphism

• Two types of polymorphism
  – Syntactic polymorphism—Java can determine which method to invoke at compile time
    • Efficient
    • Easy to understand and analyze
    • Also known as primitive polymorphism
  – Pure polymorphism—the method to invoke can only be determined at execution time
Polymorphism

- Pure polymorphism example

```java
public class PolymorphismDemo {
    // main(): application entry point
    public static void main(String[] args) {
        Point[] p = new Point[4];

        p[0] = new Colored3DPoint(4, 4, 4, Color.BLACK);
        p[1] = new ThreeDimensionalPoint(2, 2, 2);
        p[2] = new ColoredPoint(3, 3, Color.RED);
        p[3] = new Point(4, 4);

        for (int i = 0; i < p.length; ++i) {
            String s = p[i].toString();
            System.out.println("p[" + i + "]: " + s);
        }

        return;
    }
}
```
Inheritance nuances

- When a new object that is a subclass is constructed, the constructor for the superclass is always called.
  - Constructor invocation may be implicit or explicit

Example

```java
public class B {
    // B(): default constructor
    public B() {
        System.out.println("Using B's default constructor");
    }
    // B(): specific constructor
    public B(int i) {
        System.out.println("Using B's int constructor");
    }
}
```
Inheritance nuances

class C extends B {
    // C(): default constructor
    public C() {
        System.out.println("Using C's default constructor");
        System.out.println();
    }

    // C(int a): specific constructor
    public C(int a) {
        System.out.println("Using C's int constructor");
        System.out.println();
    }
}
Inheritance nuances

// c(int a, int b): specific constructor
public C(int a, int b) {
    super(a + b);
    System.out.println("Using C's int-int constructor");
    System.out.println();
}

// main(): application entry point
public static void main(String[] args) {
    C c1 = new C();
    C c2 = new C(2);
    C c3 = new C(2,4);
    return;
}
Inheritance nuances

Output

Using B's default constructor
Using C's default constructor

Using B's default constructor
Using C's int constructor

Using B's int constructor
Using C's int-int constructor
Controlling access

- Class access rights

<table>
<thead>
<tr>
<th>Member Restriction</th>
<th>this</th>
<th>Subclass</th>
<th>Package</th>
<th>General</th>
</tr>
</thead>
<tbody>
<tr>
<td>public</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>protected</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>—</td>
</tr>
<tr>
<td>default</td>
<td>✓</td>
<td>—</td>
<td>✓</td>
<td>—</td>
</tr>
<tr>
<td>private</td>
<td>✓</td>
<td>—</td>
<td>—</td>
<td>—</td>
</tr>
</tbody>
</table>
Controlling access

Example

```java
package demo;

public class P {
    // instance variable
    private int data;

    // P(): default constructor
    public P() {
        setData(0);
    }

    // getData(): accessor
    public int getData() {
        return data;
    }
}
```
Controlling access

Example (continued)

// setData(): mutator
protected void setData(int v) {
    data = v;
}

// print(): facilitator
void print() {
    System.out.println();
}
}
Controlling access

Example

```java
import demo.P;

public class Q extends P {
    // Q(): default constructor
    public Q() {
        super(); ← Q can access superclass’s public default constructor
    }

    // Q(): specific constructor
    public Q(int v) {
        setData(v); ← Q can access superclass’s protected mutator
    }
}
```
Controlling access

Example

// toString(): string facilitator
public String toString() {
    int v = getData();
    return String.valueOf(v);
}

// invalid1(): illegal method
public void invalid1() {
    data = 12;
}

// invalid2(): illegal method
public void invalid2() {
    print();
}

- Q can access superclass’s public accessor
- Q cannot access superclass’s private data field
- Q cannot directly access superclass’s default access method print()
Controlling access

Example

```java
package demo;

public class R {
    // instance variable
    private P p;

    // R(): default constructor
    public R() {
        p = new P();
    }

    // set(): mutator
    public void set(int v) {
        p.setData(v);
    }
}
```

R can access P’s public default constructor

R can access P’s protected mutator
Controlling access

Example

// get(): accessor
public int get() {
    return p.getData();
}

// use(): facilitator
public void use() {
    p.print();
}

// invalid(): illegal method
public void invalid() {
    p.data = 12;
}
Controlling access

Example

```java
import demo.P;

public class S {
    // instance variable
    private P p;

    // S(): default constructor
    public S() {
        p = new P();
    }

    // get(): inspector
    public int get() {
        return p.getData();
    }
}
```

S can access P’s public default constructor

S can access P’s public accessor
Controlling access

Example

```java
// illegal1(): illegal method
public void illegal1(int v) {
    p.setData(v);
}

// illegal2(): illegal method
public void illegal2() {
    p.data = 12;
}

// illegal3(): illegal method
public void illegal3() {
    p.print();
}
```

S cannot access P’s protected mutator

S cannot access directly P’s private data field

S cannot access directly P’s default access method print()
Data fields

- A superclass’s instance variable can be hidden by a subclass’s definition of an instance variable with the same name.

Example

```java
public class D {
    // D instance variable
    protected int d;

    // D(): default constructor
    public D() {
        d = 0;
    }

    // D(): specific constructor
    public D(int v) {
        d = v;
    }
}
```
Data fields

Class D (continued)

    // printD(): facilitator
    public void printD() {
        System.out.println("D's d: " + d);
        System.out.println();
    }
}
Data fields

- Class F extends D and introduces a new instance variable named d. F’s definition of d hides D’s definition.

```java
public class F extends D {
    // F instance variable
    int d;

    // F(): specific constructor
    public F(int v) {
        d = v; // Modification of this’s d
        super.d = v*100; // Modification of superclass’s d
    }
}
```
Data fields

Class F (continued)

// printF(): facilitator
public void printF() {
    System.out.println("D's d: " + super.d);
    System.out.println("F's d: " + this.d);
    System.out.println();
}

Inheritance and types

Example

```java
public class X {
    // default constructor
    public X() {
        // no body needed
    }
    // isX(): class method
    public static boolean isX(Object v) {
        return (v instanceof X);
    }
    // isObject(): class method
    public static boolean isObject(X v) {
        return (v instanceof Object);
    }
}
```
Inheritance and types

Example

```java
public class Y extends X {
    // Y(): default constructor
    public Y() {
        // no body needed
    }

    // isY(): class method
    public static boolean isY(Object v) {
        return (v instanceof Y);
    }
}
```
Inheritance and types

Example (continued)

```java
public static void main(String[] args) {
    X x = new X();
    Y y = new Y();
    X z = y;

    System.out.println("x is an Object: "+ X.isObject(x));
    System.out.println("x is an X: " + X.isX(x));
    System.out.println("x is a Y: " + Y.isY(x));
    System.out.println();
```
Inheritance and types

Example (continued)
System.out.println("y is an Object: " + X.isObject(y));
System.out.println("y is an X: " + X.isX(y));
System.out.println("y is a Y: " + Y.isY(y));
System.out.println();

System.out.println("z is an Object: " + X.isObject(z));
System.out.println("z is an X: " + X.isX(z));
System.out.println("z is a Y: " + Y.isY(z));
return;
}
Inheritance and types

- The program outputs the following:
  - x is an Object: true
  - x is an X: true
  - x is a Y: false
  - y is an Object: true
  - y is an X: true
  - y is a Y: true
  - z is an Object: true
  - z is an X: true
  - z is a Y: true
Polymorphism and late binding

Example

```java
public class L {
    // L(): default constructor
    public L() {
    }
    // f(): facilitator
    public void f() {
        System.out.println("Using L's f()");
        g();
    }
    // g(): facilitator
    public void g() {
        System.out.println("using L's g()");
    }
}
```
Polymorphism and late binding

Example

```java
public class M extends L {
    // M(): default constructor
    public M() {
        // no body needed
    }
    // g(): facilitator
    public void g() {
        System.out.println("Using M's g()");
    }
}
```
Polymorphism and late binding

Example

// main(): application entry point
public static void main(String[] args) {
    L l = new L();
    M m = new M();
    l.f();
    l.f();
    m.f();
    m.f();
    return;
}

Outputs

Using L's f()
Using L's g()
Using L's f()
Using M's g()
Finality

- A final class is a class that cannot be extended.
  - Developers may not want users extending certain classes
  - Makes tampering via overriding more difficult

Example

```java
final public class U {
    // U(): default constructor
    public U() {
    }

    // f(): facilitator
    public void f() {
        System.out.println("f() can’t be overridden: "
            + "U is final");
    }
}
```
Finality

- A final method is a method that cannot be overridden.

Example

```java
public class V {
    // V(): default constructor
    public V() {
    }

    // f(): facilitator
    final public void f() {
        System.out.println("Final method f() can’t be " + " overridden");
    }
}
```
Abstract base classes

- Allows creation of classes with methods that correspond to an abstract concept (i.e., there is not an implementation)
- Suppose we wanted to create a class GeometricObject
  - Reasonable concrete methods include
    - getPosition()
    - setPosition()
    - getColor()
    - setColor()
    - paint()
  - For all but paint(), we can create implementations.
  - For paint(), we must know what kind of object is to be painted. Is it a square, a triangle, etc.
  - Method paint() should be an abstract method
Abstract base classes

Example

```java
import java.awt.*;

abstract public class GeometricObject {
    // instance variables
    Point position;
    Color color;

    // getPosition(): return object position
    public Point getPosition() {
        return position;
    }

    // setPosition(): update object position
    public void setPosition(Point p) {
        position = p;
    }
}

Makes GeometricObject an abstract class
```
Abstract base classes

Example (continued)

// getColor(): return object color
public Color getColor() {
    return color;
}

// setColor(): update object color
public void setColor(Color c) {
    color = c;
}

// paint(): render the shape to graphics context g
abstract public void paint(Graphics g);

Indicates that an implementation of method paint() will not be supplied
Interfaces

- An interface is a template that specifies what must be in a class that implements the interface
  - An interface cannot specify any method implementations
  - All the methods of an interface are public
  - All the variables defined in an interface are public, final, and static
Interfaces

• An interface for a colorable object

```java
public interface Colorable {
    // getColor(): return the color of the object
    public Color getColor();

    // setColor(): set the color of the object
    public void setColor(Color c);
}
```

• Now the interface can be used to create classes that implement the interface
Interfaces

- ColorablePoint

```java
import java.awt.*;

public class ColorablePoint extends Point implements Colorable {
    // instance variable
    Color color;

    // ColorablePoint(): default constructor
    public ColorablePoint() {
        super();
        setColor(Color.blue);
    }

    ...}
```

Class ColorablePoint must provide implementations of getColor() and setColor()
Exceptions
Exception

• Abnormal event occurring during program execution

• Examples
  – Manipulate nonexistent files
    ```java
    FileReader in = new FileReader("numbers.txt");
    ```
  – Improper array subscripting
    ```java
    int[] a = new int[3];
    a[4] = 1000;
    ```
  – Improper arithmetic operations
    ```java
    a[2] = 1000 / 0;
    ```
Java treatment of an exception

- If exception occurs and a *handler* is in effect
  - Flow of control is transferred to the handler
  - After handler completes flow of control continues with the statement following the handler

- If exception occurs and there is no handler for it
  - The program terminates
Task

• Prompt and extract the name of a file

• From that file, two integer values are to be extracted

• Compute and display the quotient of the values
public static void main(String[] args) throws IOException {

    Scanner stdin = Scanner.create(System.in);
    System.out.print("Filename: ");
    String s = stdin.nextLine();

    File file = new File(s);
    Scanner fileIn = Scanner.create(file);

    int a = fileIn.nextInt();
    int b = fileIn.nextInt();

    System.out.println( a / b );
}
public static void main(String[] args) throws IOException {
    Scanner stdin = Scanner.create(System.in);
    System.out.print("Filename: ");
    String s = stdin.nextLine();
    File file = new File(s);
    Scanner fileIn = Scanner.create(file);
    int a = fileIn.nextInt();
    int b = fileIn.nextInt();
    System.out.println(a / b);
}
How can we deal with the problems?

```java
public static void main(String[] args) throws IOException {

    Scanner stdin = Scanner.create(System.in);
    System.out.print("Filename: ");
    String s = stdin.nextLine();

    File File = new File(s);
    Scanner fileIn = Scanner.create(file);

    int a = fileIn.nextInt();
    int b = fileIn.nextInt();

    System.out.println(a / b);
}
```
Exception handlers

• Code that might generate an exception is put in a try block
  – If there is no exception, then the handlers are ignored
• For each potential exception type there is a catch handler
  – When handler finishes the program continues with statement after the handlers

```java
try {
    Code that might throw exceptions of types E or F
} catch (E e) {
    Handle exception e
} catch (F f) {
    Handle exception f
} More code
```
Introduce try-catch blocks

public static void main(String[] args) throws IOException {

    Scanner stdin = Scanner.create(System.in);
    System.out.print("Filename: ");
    String s = stdin.nextLine();

    File file = new File(s);
    Scanner fileIn = Scanner.create(file);

    int a = fileIn.nextInt();
    int b = fileIn.nextInt();

    System.out.println(a / b);
}

Setting up the file stream processing

Scanner stdin = Scanner.create(System.in);
System.out.print("Filename: ");
String s = stdin.nextLine();
// set up file stream for processing
Scanner fileIn = null;

try {
    File file = new File(s);
    fileIn = Scanner.create(file);
}
catch (FileNotFoundException e) {
    System.err.println(s + ": Cannot be opened for reading");
    System.exit(0);
}

How come the main() throws expression did not indicate it could throw a FileNotFoundException?
Getting the inputs

try {
    int a = fileIn.nextInt();
    int b = fileIn.nextInt();
    System.out.println( a / b );
}
catch (InputMismatchException e) {
    System.err.println(s + ": contains nonnumeric inputs");
    System.exit(0);
}
Converting the inputs

```java
try {
    int a = fileIn.nextInt();
    int b = fileIn.nextInt();
    System.out.println( a / b );
}
catch (InputMismatchException e) {
    System.err.println(s + " contains nonnumeric inputs");
    System.exit(0);
}
catch (NoSuchElementException e) {
    System.err.println(s + " does not contain two inputs");
    System.exit(0);
}
catch (ArithmeticException e) {
    System.err.println(s + " unexpected 0 input value");
    System.exit(0);
}
```
Run time exceptions

• Java designers realized
  – Runtime exceptions can occur throughout a program
  – Cost of implementing handlers for runtime exceptions typically exceeds the expected benefit

• Java makes it optional for a method to catch them or to specify that it throws them

• However, if a program does not handle its runtime exceptions it is terminated when one occurs
Commands type and cat

- Most operating systems supply a command for listing the contents of files
  - Windows: type
  - Unix, Linux, and OS X: cat

\[
type \text{ filename}_1 \text{ filename}_2 \ldots \text{ filename}_n
\]

- Displays the contents of filename\textsubscript{1}, filename\textsubscript{2}, ..., and filename\textsubscript{n} to the console window
Possible method main() for Type.java

```java
class Type {
    public static void main(String[] args) {
        for (int i = 0; i < args.length; ++i) {
            File file = new File(args[i]);
            Scanner fileIn = Scanner.create(file);

            while (fileIn.hasNext()) {
                String s = fileIn.nextLine();
                System.out.println(s);
            }
            fileIn.close();
        }
    }
}
```

What can go wrong?
Use a finally block

```java
public static void main(String[] args) {
    for (int i = 0; i < args.length; ++i) {
        File fileIn = new File(args[i]);
        Scanner fileIn = Scanner.create(file);
        while (fileIn.hasNext()) {
            String s = fileIn.nextLine();
            System.out.println(s);
        }
        fileIn.close();
    }
}
```

File should be closed once its processing stops, regardless why it stopped.
Use a finally block

```java
try {
    fileIn = null;
    File file = new File(args[i]);
    fileIn = Scanner.create(file);
    while (fileIn.hasNext()) {
        String s = fileIn.nextLine();
        System.out.println(s);
    }
} catch (FileNotFoundException e) {
    System.err.println(args[i] + " cannot be opened");
} catch (IOException e) {
    System.err.println(args[i] + " processing error");
} finally {
    if (fileIn != null) {
        filein.close();
    }
}
```
Exceptions

- Can create your exception types
- Can throw exceptions as warranted
Task

- Represent the depositing and withdrawing of money from a bank account

- What behaviors are needed
  - Construct a new empty bank account
    BankAccount()
  - Construct a new bank account with initial funds
    BankAccount(int n)
  - Deposit funds
    addFunds(int n)
  - Withdraw funds
    removeFunds(int n)
  - Get balance
    Int getBalance()
public static void main(String[] args)
    
    Scanner stdin = Scanner.create(System.in);
    
    BankAccount savings = new BankAccount();
    
    System.out.print("Enter deposit: ");
    int deposit = stdin.nextInt();
    savings.addFunds(deposit);
    
    System.out.print("Enter withdrawal: ");
    int withdrawal = stdin.nextInt();
    savings.removeFunds(withdrawl);
    
    System.out.println("Closing balance: 
        + savings.getBalance()");
}
Task

- Represent the depositing and withdrawing of money from a bank account

- What behaviors are needed
  - Construct a new empty bank account
    BankAccount()
  - Construct a new bank account with initial funds
    BankAccount(int n)
  - Deposit funds
    addFunds(int n)
  - Withdraw funds
    removeFunds(int n)
  - Get balance
    int getBalance()

What can go wrong?
Create a NegativeAmountException

// Represents an abnormal bank account event

public class NegativeAmountException extends Exception {
    // NegativeAmountException(): creates exception with
    // message s
    public NegativeAmountException(String s) {
        super(s);
    }
}

- Class Exception provides the exceptions behavior that might be needed
- Class NegativeAmountException gives the specialization of exception type that is needed
Sample usage

```java
public static void main(String[] args) throws IOException, NegativeAmountException {
    Scanner stdin = Scanner.create(System.in);
    BankAccount savings = new BankAccount();
    System.out.print("Enter deposit: ");
    int deposit = stdin.nextInt();
    savings.addFunds(deposit);
    System.out.print("Enter withdrawal: ");
    int withdrawal = stdin.nextInt();
    savings.removeFunds(withdrawal);
    System.out.println("Closing balance: "+ savings.getBalance());
}
```
Class BankAccount

- Instance variable
  balance
- Construct a new empty bank account
  BankAccount()
- Construct a new bank account with initial funds
  BankAccount(int n) throws NegativeAmountException
- Deposit funds
  addFunds(int n) throws NegativeAmountException
- Withdraw funds
  removeFunds(int n) throws NegativeAmountException
- Get balance
  Int getBalance()
Class BankAccount

// BankAccount(): default constructor for empty balance
public BankAccount() {
    balance = 0;
}

// BankAccount(): specific constructor for a new balance n
public BankAccount(int n) throws NegativeAmountException {
    if (n >= 0) {
        balance = n;
    }
    else {
        throw new NegativeAmountException("Bad balance");
    }
}
Class BankAccount

// getBalance(): return the current balance
public int getBalance() {
    return balance;
}

// addFunds(): deposit amount n
public void addFunds(int n) throws NegativeAmountException {
    if (n >= 0) {
        balance += n;
    } else {
        throw new NegativeAmountException("Bad deposit");
    }
}
Class BankAccount

// removeFunds(): withdraw amount n
public void removeFunds(int n) throws NegativeAmountException {
    if (n < 0) {
        throw new NegativeAmountException("Bad withdrawal");
    }
    else if (balance < n) {
        throw new NegativeAmountException("Bad balance");
    }
    else {
        balance -= n;
    }
}
Using NegativeAmountException

System.out.print("Enter deposit: ");
try {
    int deposit = stdin.nextInt();
    savings.addFunds(deposit);
}
catch (NegativeAmountException e) {
    System.err.println("Cannot deposit negative funds");
    System.exit(0);
}
Using NegativeAmountException

System.out.print("Enter withdrawal: ");
try {
    int withdrawal = stdin.nextInt();
    savings.removeFunds(withdrawal);
} catch (NegativeAmountException e) {
    if (e.message().equals("Bad withdrawal"))
        System.err.println("Cannot withdraw negative funds");
    else {
        System.err.println("Withdrawal cannot leave "
                          "negative balance");
    }
    System.exit(0);
}
Recursion
Recursive definitions

• A definition that defines something in terms of itself is called a recursive definition.
  – The *descendants* of a person are the person’s children and all of the *descendants* of the person’s children.
  – A *list of numbers* is a number or a number followed by a comma and a *list of numbers*.

• A recursive algorithm is an algorithm that invokes itself to solve smaller or simpler instances of a problem instances.
  – The factorial of a number \( n \) is \( n \) times the factorial of \( n-1 \).
Factorial

• An imprecise definition

\[ n! = \begin{cases} 
1 & n = 0 \\
(n-1) \cdots 1 & n \geq 1 
\end{cases} \]

Ellipsis tells the reader to use intuition to recognize the pattern

• A precise definition

\[ n! = \begin{cases} 
1 & n = 0 \\
(n-1)! & n \geq 1 
\end{cases} \]
Recursive methods

- A recursive method generally has two parts.
  - A termination part that stops the recursion.
    - This is called the base case.
    - The base case should have a simple or trivial solution.
  - One or more recursive calls.
    - This is called the recursive case.
    - The recursive case calls the same method but with simpler or smaller arguments.
Method factorial()

public static int factorial(n) {
    if (n == 0) {
        return 1;  // Base case
    }
    else {
        return n * factorial(n-1);  // Recursive case deals with a simpler (smaller) version of the task
    }
}
Method factorial()

```java
public static int factorial(n) {
    if (n == 0) {
        return 1;
    } else {
        return n * factorial(n-1);
    }
}

public static void main(String[] args) {
    Scanner stdin = Scanner.create(System.in);
    int n = stdin.nextInt();
    int nfactorial = factorial(n);
    System.out.println(n + "! = " + nfactorial);
}
```
Recursive invocation

- A new activation record is created for every method invocation
  - Including recursive invocations

```java
main()

int nfactorial = factorial(n);
```
Recursive invocation

- A new activation record is created for every method invocation
  - Including recursive invocations

```java
main()

int n = 3

factorial()

int nfactorial = factorial(n);

return n * factorial(n-1);
```
Recursive invocation

- A new activation record is created for every method invocation
  - Including recursive invocations

```
int nfactorial = factorial(n);
```

```
main()
return n * factorial(n-1);
```

```
factorial()
return n * factorial(n-1);
```

```
factorial()
return n * factorial(n-1);
```

```
factorial()
return n * factorial(n-1);
```

```
factorial()
return n * factorial(n-1);
```

```
factorial()
return n * factorial(n-1);
```

```
factorial()
return n * factorial(n-1);
```

```
factorial()
return n * factorial(n-1);
```

```
factorial()
return n * factorial(n-1);
```

```
factorial()
return n * factorial(n-1);
```
Recursive invocation

- A new activation record is created for every method invocation – Including recursive invocations

```java
int nfactorial = factorial(n);

main()
return n * factorial(n-1);
n = 3

factorial()
return n * factorial(n-1);

n = 2

factorial()
return n * factorial(n-1);

n = 1

factorial()
return n * factorial(n-1);
```
Recursive invocation

- A new activation record is created for every method invocation
  - Including recursive invocations

```java
int nfactorial = factorial(n);
```

```
main()
return n * factorial(n-1);
```

```
factorial()
return n * factorial(n-1);
```

```
factorial()
return n * factorial(n-1);
```

```
factorial()
return n * factorial(n-1);
```

```
factorial()
return n * factorial(n-1);
```

```
factorial()
return 1;
```
Recursive invocation

- A new activation record is created for every method invocation
  - Including recursive invocations

```java
int nfactorial = factorial(n);
main()
return n * factorial(n-1);
n = 3
factorial()
return n * factorial(n-1);
n = 2
factorial()
return n * factorial(n-1);
n = 1
factorial()
return n * factorial(n-1);
n = 0
factorial()
return 1;
```
Recursive invocation

- A new activation record is created for every method invocation
  - Including recursive invocations

```java
int nfactorial = factorial(n);

int factorial() {
    if (n == 0) {
        return 1;
    } else {
        return n * factorial(n-1);
    }
}
```

```
main()
return n * factorial(n-1);
```

```
factorial()
return n * factorial(n-1);
```

```
factorial()
return n * factorial(n-1);
```

```
factorial()
return n * 1;
```

```
factorial()
return 1;
```
Recursive invocation

- A new activation record is created for every method invocation
  - Including recursive invocations

```java
int nfactorial = factorial(n);
main()
  return n * factorial(n-1);

n = 3
factorial()
  return n * factorial(n-1);
n = 2
factorial()
  return n * factorial(n-1);
n = 1
factorial()
  return 1 * 1;
```
Recursive invocation

- A new activation record is created for every method invocation
  - Including recursive invocations

```
main()
    int nfactorial = factorial(n);

factorial()
    return n * factorial(n-1);
    n = 3

factorial()
    return n * 1
    n = 2

factorial()
    return 1 * 1
    n = 1
```
Recursive invocation

- A new activation record is created for every method invocation
  - Including recursive invocations

```java
int nfactorial = factorial(n);

main()
return n * factorial(n-1);

n = 3
factorial()
return 2 * 1

n = 2
factorial()
return 2 * 1
```
Recursive invocation

- A new activation record is created for every method invocation
  - Including recursive invocations

```plaintext
main()
  nfactorial = factorial(n);

factorial()
  n = 3
  return 2 * 1;

factorial()
  n = 2
  return n * 2;
```
Recursive invocation

- A new activation record is created for every method invocation
  - Including recursive invocations

```java
int nfactorial = factorial(n);

return 3 * 2;
```

```java
main()

n = 3

factorial()
```
Recursive invocation

- A new activation record is created for every method invocation
  - Including recursive invocations

```java
int nfactorial = 6;

main()
return 3 * 2;

n = 3
factorial()
return 3 * 2;
```
Recursive invocation

- A new activation record is created for every method invocation
  - Including recursive invocations

```java
int nfactorial = 6;
```
Fibonacci numbers

• Developed by Leonardo Pisano in 1202.
  – Investigating how fast rabbits could breed under idealized circumstances.
  – Assumptions
    • A pair of male and female rabbits always breed and produce another pair of male and female rabbits.
    • A rabbit becomes sexually mature after one month, and that the gestation period is also one month.
  – Pisano wanted to know the answer to the question how many rabbits would there be after one year?
Fibonacci numbers

• The sequence generated is: 1, 1, 2, 3, 5, 8, 13, 21, 34, ...
• The number of pairs for a month is the sum of the number of pairs in the two previous months.
• Insert Fibonacci equation
Fibonacci numbers

Recursive method pattern

```java
public static int fibonacci(int n) {
    if (n <= 2) {
        return 1;
    } else {
        return fibonacci(n-1) + fibonacci(n-2);
    }
}
```
Fibonacci numbers
Infinite recursion

• A common programming error when using recursion is to not stop making recursive calls.
  – The program will continue to recurse until it runs out of memory.
  – Be sure that your recursive calls are made with simpler or smaller subproblems, and that your algorithm has a base case that terminates the recursion.
Binary search

- Compare the entry to the middle element of the list. If the entry matches the middle element, the desired entry has been located and the search is over.
- If the entry doesn’t match, then if the entry is in the list it must be either to the left or right of the middle element.
- The correct sublist can be searched using the same strategy.
public class AddressEntry {
    private String personName;
    private String telephoneNumber;

    public AddressEntry(String name, String number) {
        personName = name;
        telephoneNumber = number;
    }

    public String getName() {
        return personName;
    }

    public String getNumber() {
        return telephoneNumber;
    }

    public void setName(String Name) {
        personName = Name;
    }

    public void setTelephoneNumber(String number) {
        telephoneNumber = number;
    }

}
Binary search

- Public interface
  - Should be as simple as possible
  - No extraneous parameters

  ```java
  public static AddressEntry recSearch(AddressEntry[] addressBook, String name)
  ```

- Private interface
  - Invoked by implementation of public interface
  - Should support recursive invocation by implementation of private interface

  ```java
  private static AddressEntry recSearch(AddressEntry[] addressBook, String name, int first, int last)
  ```
Binary search

- Public interface implementation

```java
public static AddressEntry recSearch(AddressEntry[] addressBook, String name) {
    return recSearch(addressBook, name, 0, addressBook.length-1);
}
```
Private interface implementation

```java
static AddressEntry recSearch(AddressEntry[] addressBook,
   String name, int first, int last) {
    // base case: if the array section is empty, not found
    if (first > last)
      return null;
    else {
      int mid = (first + last) / 2;
      // if we found the value, we're done
      if (name.equalsIgnoreCase(addressBook[mid].getName()))
        return addressBook[mid];
      else if (name.compareToIgnoreCase(
                 addressBook[mid].getName()) < 0) {
        // if value is there at all, it's in the left half
        return recSearch(addressBook, name, first, mid-1);
      }
      else { // array[mid] < value
        // if value is there at all, it's in the right half
        return recSearch(addressBook, name, mid+1, last);
      }
    }
}
```
Testing

- Develop tests cases to exercise every possible unique situation

```java
public static void main(String[] args) {
    // list must be in sorted order
    AddressEntry addressBook[] = {
        new AddressEntry("Audrey", "434-555-1215"),
        new AddressEntry("Emily", "434-555-1216"),
        new AddressEntry("Jack", "434-555-1217"),
        new AddressEntry("Jim", "434-555-2566"),
        new AddressEntry("John", "434-555-2222"),
        new AddressEntry("Lisa", "434-555-3415"),
        new AddressEntry("Tom", "630-555-2121"),
        new AddressEntry("Zach", "434-555-1218")
    };
```
Testing

- Search for first element

    AddressEntry p;
    // first element
    p = recSearch(addressBook, "Audrey");
    if (p != null) {
        System.out.println("Audrey's telephone number is " +
            p.getNumber());
    } else {
        System.out.println("No entry for Audrey");
    }
Testing

- Search for middle element

```java
p = recSearch(addressBook, "Jim");
if (p != null) {
    System.out.println("Jim's telephone number is " +
    p.getNumber());
} else {
    System.out.println("No entry for Jim");
}
```
Testing

- Search for last element

```java
p = recSearch(addressBook, "Zach");
if (p != null) {
    System.out.println("Zach's telephone number is " +
                        p.getNumber());
} else {
    System.out.println("No entry for Zach");
}
```
Testing

- Search for non-existent element

```java
p = recSearch(addressBook, "Frank");
if (p != null) {
    System.out.println("Frank's telephone number is " + 
    p.getNumber());
}
else {
    System.out.println("No entry for Frank");
}
```
Efficiency of binary search

- Height of a binary tree is the worst case number of comparisons needed to search a list
- Tree containing 31 nodes has a height of 5
- In general, a tree with $n$ nodes has a height of $\log_2(n+1)$
- Searching a list with a billion nodes only requires 31 comparisons
- Binary search is efficient!
Mergesort

• Mergesort is a recursive sort that conceptually divides its list of \( n \) elements to be sorted into two sublists of size \( n/2 \).
  – If a sublist contains more than one element, the sublist is sorted by a recursive call to mergeSort().
  – After the two sublists of size \( n/2 \) are sorted, they are merged together to produce a single sorted list of size \( n \).

• This type of strategy is known as a divide-and-conquer strategy—the problem is divided into subproblems of lesser complexity and the solutions of the subproblems are used to produced the overall solution.

• The running time of method mergeSort() is proportional to \( n \log n \).

• This performance is sometimes known as \textit{linearithmic} performance.
Mergesort

- Suppose we are sorting the array shown below.

  
  

- After sorting the two sublists, the array would look like

  
  

- Now we can do the simple task of merging the two arrays to get

  
  

  E ' R ' T ' Y'
Mergesort

• Public interface
  - Should be as simple as possible
  - No extraneous parameters

  public static void mergeSort(char[] a)

• Private interface
  - Invoked by implementation of public interface
  - Should support recursive invocation by implementation of private interface

  private static void mergeSort(char[] a, int left, int right)
Mergesort

private static void mergeSort(char[] a, int left, int right) {
    if (left < right) {
        // there are multiple elements to sort.

        // first, recursively sort the left and right sublists
        int mid = (left + right) / 2;
        mergeSort(a, left, mid);
        mergeSort(a, mid+1, right);
    }
}
Mergesort

// next, merge the sorted sublists into
// array temp
char[] temp = new char[right - left + 1];

int j = left; // index of left sublist smallest ele.
int k = mid + 1; // index of right sublist smallest ele.
for (int i = 0; i < temp.length; ++i) {
  // store the next smallest element into temp
  if ((j <= mid) && (k <= right)) {
    // need to grab the smaller of a[j]
    if (a[j] <= a[k]) { // left has the smaller element
      temp[i] = a[j];
      ++j;
    }
    else { // right has the smaller element
      temp[i] = a[k];
      ++k;
    }
  }
  else if (a[j] <= a[k]) { // left has the smaller element
    temp[i] = a[j];
    ++j;
  }
  else { // right has the smaller element
    temp[i] = a[k];
    ++k;
  }
}
Mergesort

else if (j <= mid) { // can only grab from left half
    temp[i] = a[j];
    ++j;
}
else { // can only grab from right half
    temp[i] = a[k];
    ++k;
}
}
Mergesort

// lastly, copy temp into a
for (int i = 0; i < temp.length; ++i) {
    a[left + i] = temp[i];
}
}
Mergesort

Invocation considers the elements of `EQRSTWUYOP` with subscripts from interval 0 \( \leq i \leq 9 \) and produces `EQRSTWUYOP`.

Initial invocation considers the elements of `QWERTYUIOP` from interval 0 \( \leq i \leq 9 \) and produces `EIOPTUQRYW`.

```
0 ... 9
QWERTYUIOP
EIOPTUQRYW

5 ... 9
EQRSTWUYOP
EQRSTWUYOP

5 ... 7
EQRSTWUYOP
EQRSTWUYOP

3 ... 4
EQRSTWUYOP
EQRSTWUYOP

2 ... 2
EQRSTWUYOP
EQRSTWUYOP
```
Recursion versus iteration

- Iteration can be more efficient
  - Replaces method calls with looping
  - Less memory is used (no activation record for each call)
- Many problems are more naturally solved with recursion
  - Towers of Hanoi
  - Mergesort
- Choice depends on problem and the solution context
A TRAPEZE ARTIST CAN DO THIS WITH A COMPUTER.

You can find out about anything here.

Unscramble these four Jumbles, one letter to each square, to form four ordinary words.

MEZIA
CIDDE
HUCNHA
BOUSTE

Print answer here: THE

Now arrange the circled letters to form the answer, as suggested by the above cartoon.
Daily Jumble

• Task
  – Generate all possible permutation of letters
  • For n letters there are n! possibilities
    – n choices for first letter
    – n-1 choices for second letter
    – n-2 choices for third letter
    – ...

• Iterator
  – Object that produces successive values in a sequence

• Design
  – Iterator class PermuteString that supports the enumeration of permutations
Class PermuteString

• Constructor
  public PermuteString(String s)
  • Constructs a permutation generator for string s

• Methods
  public String nextPermutation()
  • Returns the next permutation of the associated string

  public boolean morePermutations()
  • Returns whether there is unenumerated permutation of the associated string
Class PermuteString

• Instance variables
  
  private String word
  • Represents the word to be permuted

  private int index
  • Position within the word being operated on

  public PermuteString substringGenerator
  • Generator of substrings
Constructor

public PermuteString(String s) {
    word = s;
    index = 0;
    if (s.length() > 1) {
        String substring = s.substring(1);
        substringGenerator = new PermuteString(substring);
    } else {
        substringGenerator = null;
    }
}
Consider

- What happens?
  
  ```java
  PermuteString p = new PermuteString("ath");
  ```

```
word: "ath"
index: 0
substringGenerator:
```
```
word: "th"
index: 0
substringGenerator:
```
```
word: "h"
index: 0
substringGenerator: null
```
public boolean morePermutations() {
    return index < word.length;
}
public boolean nextPermutation() {
    if (word.length() == 1) {
        ++index;
        return word;
    }
    else {
        String r = word.charAt(index) + substringGenerator.nextPermutation();
        if (!substringGenerator.morePermutations()) {
            ++index;
            if (index < word.length()) {
                String tail = word.substring(0, index) + word.substring(index + 1);
                substringGenerator = new permuteString(tail);
            }
        }
        return r;
    }
}
Threads
Story so far

• Our programs have consisted of single flows of control
  – Flow of control started in the first statement of method main() and worked its way statement by statement to the last statement of method main()
  – Flow of control could be passed temporarily to other methods through invocations, but the control returned to main() after their completion

• Programs with single flows of control are known as sequential processes

```java
Single-threaded Program {
  Statement 1;
  Statement 2;
  ...
  Statement k;
}
```

Although the statements within a single flow of control may invoke other methods, the next statement is not executed until the current statement completes
Processes

- The ability to run more than one process at the same time is an important characteristic of modern operating systems
  - A user desktop may be running a browser, programming IDE, music player, and document preparation system

- Java supports the creation of programs with concurrent flows of control – threads
  - Threads run within a program and make use of its resources in their execution
    - Lightweight processes
Processes

- Java Console
- Adobe FrameMaker - [C:\]Documents and Sett...
- UPS Package Tracking - Microsoft Internet Ex...
- C:\Documents and Settings\cohoon\My Doc...
- CD Writing Wizard
- F:\
- "Dust Bowl" by Natalie Merchant - MUSICMAT...
- PlayListMgr
- LapLink Gold

- dllhost.exe: SYSTEM, CPU 00, Mem Usage 824 K
- msdtc.exe: NETWORK SERVICE, CPU 00, Mem Usage 76 K
- taskmgr.exe: cohoon, CPU 00, Mem Usage 4,220 K
- dllhost.exe: IWAM_FREEDOM, CPU 00, Mem Usage 1,208 K
- LLSEQV~1.EXE: cohoon, CPU 06, Mem Usage 252 K
- laplink.exe: cohoon, CPU 00, Mem Usage 1,764 K
- mmj.exe: cohoon, CPU 00, Mem Usage 10,212 K
- imapi.exe: SYSTEM, CPU 00, Mem Usage 2,496 K
- FrameMaker.exe: cohoon, CPU 00, Mem Usage 10,180 K
- realplay.exe: cohoon, CPU 00, Mem Usage 2,024 K
- wcescomm.exe: cohoon, CPU 00, Mem Usage 380 K
- SpeedKey.exe: cohoon, CPU 00, Mem Usage 380 K
- LLSeqEng.exe: cohoon, CPU 00, Mem Usage 428 K
- TSIRCSR.exe: cohoon, CPU 00, Mem Usage 64 K
- mmdiag.exe: cohoon, CPU 00, Mem Usage 2,184 K
- acrotray.exe: cohoon, CPU 00, Mem Usage 260 K
- zm32nt.exe: cohoon, CPU 00, Mem Usage 340 K
- TSIRCUSR.exe: SYSTEM, CPU 00, Mem Usage 60 K
- Tablet.exe: SYSTEM, CPU 00, Mem Usage 316 K

Processes: 50, CPU Usage: 7%, Commit Charge: 234924K / 63306K

Processes: 48, CPU Usage: 8%, Commit Charge: 219004K / 63306K
Multithread processing

This multithreaded program starts up thread A to assist in the program's task. After starting the thread, the program continues with its next statement.

Multithread Program {
  Statement 1;
  Statement 2;
  ...
  Statement i;
  ...
  Statement j;
  ...
  Statement m;
}

Thread A {
  A Statement 1;
  A Statement 2;
  ...
  A Statement n;
  ...
  A Statement p;
}

Thread B {
  B Statement 1;
  B Statement 2;
  ...
  B Statement q;
}

A multithreaded program ends when all of its individual flows of control (threads) end.

Thread A starts up thread C to assist in its program subtask. After starting the thread, thread A continues with its next statement.

Thread C {
  C Statement 1;
  C Statement 2;
  ...
  C Statement r;
}
Timer and TimerTask

• Among others, Java classes java.util.Timer and java.util.TimerTask support the creation and scheduling of threads

• Abstract class Timer has methods for creating threads after either some specified delay or at some specific time
  – public void schedule(TimerTask task, long m)
    • Runs task.run() after waiting m milliseconds.
  – public void schedule(TimerTask task, long m, long n)
    • Runs task.run() after waiting m milliseconds. It then repeatedly reruns task.run() every n milliseconds.
  – public void schedule(TimerTask task, Date y)
    • Runs task.run() at time t.

• A thread can be created By extending TimerTask and specifying a definition for abstract method run()
Running after a delay

- Class `DisplayCharSequence` extends `TimerTask` to support the creation of a thread that displays 20 copies of some desired character (e.g., “H”, “A”, or “N”)

```
<table>
<thead>
<tr>
<th>cmd:</th>
<th>javac DisplayCharSequence.java</th>
</tr>
</thead>
<tbody>
<tr>
<td>cmd:</td>
<td>java DisplayCharSequence</td>
</tr>
<tr>
<td></td>
<td>HHHHHHHHHHHHHHHANANANANANANNNN</td>
</tr>
<tr>
<td></td>
<td>NNNNNNAAAAAANHNNNAAAAANNNNNNN</td>
</tr>
<tr>
<td>cmd:</td>
<td>java DisplayCharSequence</td>
</tr>
<tr>
<td></td>
<td>HHHHHHHHHHHHHHHANANANANANANAN</td>
</tr>
<tr>
<td></td>
<td>NNNNNNAAAAAANHNNNAAAAANNNNNNN</td>
</tr>
<tr>
<td>cmd:</td>
<td>java DisplayCharSequence</td>
</tr>
<tr>
<td></td>
<td>HHHHHHHHHHHHHHHANANANANANANAN</td>
</tr>
<tr>
<td></td>
<td>NNNNNNAAAAAANHNNNAAAAANNNNNNN</td>
</tr>
<tr>
<td>cmd:</td>
<td></td>
</tr>
</tbody>
</table>
```
public static void main(String[] args) {
    DisplayCharSequence s1 =
        new DisplayCharSequence('H');

    DisplayCharSequence s2 =
        new DisplayCharSequence('A');

    DisplayCharSequence s3 =
        new DisplayCharSequence('N');
}
import java.util.*;

public class DisplayCharSequence extends TimerTask {
    private char displayChar;
    Timer timer;
    public DisplayCharSequence(char c) {
        displayChar = c;
        timer = new Timer();
        timer.schedule(this, 0);
    }
    public void run() {
        for (int i = 0; i < 20; ++i) {
            System.out.print(displayChar);
        }
        timer.cancel();
    }
}
Implementing a run() method

• A subclass implementation of TimerTask’s abstract method run() has typically two parts
  – First part defines the application-specific action the thread is to perform
  – Second part ends the thread
    • The thread is ended when the application-specific action has completed

```java
// run(): display the occurrences of the character of interest
public void run() {
    for (int i = 0; i < 20; ++i) {
        System.out.print(displayChar);
    }
    timer.cancel();
}
```
Running repeatedly

- Example
  - Having a clock face update every second

```java
public static void main(String[] args) {
    SimpleClock clock = new SimpleClock();
}
```
public class SimpleClock extends TimerTask {

    final static long MILLISECONDS_PER_SECOND = 1000;

    private JFrame window = new JFrame("Clock");
    private Timer timer = new Timer();
    private String clockFace = "";

    public SimpleClock() {
        window.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        window.setSize(200, 60);
        Container c = window.getContentPane();
        c.setBackground(Color.white);
        window.setVisible(true);
        timer.schedule(this, 0, 1*MILLISECONDS_PER_SECOND);
    }

    public void run() {
        Date time = new Date();
        Graphics g = window.getContentPane().getGraphics();
        g.setColor(Color.WHITE);
        g.drawString(clockFace, 10, 20);
        clockFace = time.toString();
        g.setColor(Color.BLUE);
        g.drawString(clockFace, 10, 20);
    }
}

SimpleClock scheduling

timer.schedule(this, 0, 1*MILLISECONDS_PER_SECOND);

The millisecond delay before the thread is first scheduled

The number of milliseconds between runs of the thread
Running at a chosen time

- Example
  - Scheduling calendar pop-ups using class DisplayAlert
public static void main(String[] args) {
    Calendar c = Calendar.getInstance();
    c.set(Calendar.HOUR_OF_DAY, 9);
    c.set(Calendar.MINUTE, 30);
    c.set(Calendar.SECOND, 0);
    Date studentTime = c.getTime();

    c.set(Calendar.HOUR_OF_DAY, 18);
    c.set(Calendar.MINUTE, 15);
    c.set(Calendar.SECOND, 0);
    Date danceTime = c.getTime();

    DisplayAlert alert1 = new DisplayAlert(
            "Prospective student meeting", studentTime);
    DisplayAlert alert2 = new DisplayAlert(
            "Dance recital", danceTime);
}
}

public void run() {
JOptionPane.showMessageDialog(null, message);
timer.cancel();
}

public DisplayAlert(String s, Date t) {
message = s + ": " + t;
timer = new Timer();
timer.schedule(this, t);
}

public class DisplayAlert extends TimerTask {
private String message;
private Timer timer;

import javax.swing.JOptionPane;
import java.awt.*;
import java.util.*;

Defining DisplayAlert


Sleeping

- Threads can be used to pause a program for a time
- Standard class `java.lang.Thread` has a class method `sleep()` for pausing a flow of control

```java
public static void sleep(long n) throws InterruptedException
- Pauses the current thread for n milliseconds. It then throws an InterruptedException.
```
Sleeping example

- Code
  ```java
  Date t1 = new Date();
  System.out.println(t1);
  try {
    Thread.sleep(10000);
  }
  catch (InterruptedException e) {
  }
  Date t2 = new Date();
  System.out.println(t2);
  ```

- Output
  Fri Jan 31 19:29:45 EST 2003
  Fri Jan 31 19:29:55 EST 2003
Testing and Debugging
Testing Fundamentals

• Test as you develop
  – Easier to find bugs early rather than later
  – Prototyping helps identify problems early
• Categories of bugs
  – software crashes or data corruption
  – failure to meet or satisfy the specification
  – poor or unacceptable performance
  – hard or difficult to use
Testing fundamentals

- Impossible to test a program completely
- Three distinct paths through the program
- If the loop executes 20 times, there are $3^{20}$ different sequences of executions
Reviews and inspections

- Inspections
  - Formal process of reviewing code
  - First employed by IBM in 1976
  - Early work showed that design and review inspections remove 60 percent of the bugs in a product
Reviews and inspections

- Roles of participants
  - Moderator
    - Runs the inspection
    - Ensure that the process moves along
    - Referees the discussion
    - Ensures action items done
  - Inspector
    - Someone other than author
    - Some interest in the code
    - Carefully review code before inspection meeting
  - Author
    - Minor role
    - May answer questions about code
Reviews and inspections

• Roles of participants
  – Scribe
    • Record all errors detected
    • Keep list of action items
Reviews and inspections

• Inspection process
  – Planning
    • Code to review chosen
    • Moderator assigns task
    • Checklists created
    • Moderator assigns a presenter (usually one of the inspectors)
  – Overview
    • Author describes high-level aspects of project that affected the design or code
    • Sometimes skipped (if all participants are knowledgeable)
Reviews and inspections

• Inspection process
  – Preparation
    • Working alone, each inspector reviews code noting problems or questions
    • Shouldn’t take more than a couple of hours
  – Inspection meeting
    • Presenter walks through the code
    • Problems are discussed
    • Scribe records all errors and action items
    • Errors are not fixed at this time
  – Inspection report
    • Moderator prepares a written report
Black-box and white-box testing

- White-box testing indicates that we can “see” or examine the code as we develop test cases
- Block-box testing indicates that we cannot examine the code as we devise test cases
  - Seeing the code can bias the test cases we create
  - Forces testers to use specification rather than the code
- Complementary techniques
Black-box and white-box testing

Test boundary conditions

```java
public static int binarySearch(char[] data, char key) {
    int left = 0;
    int right = data.length - 1;
    while (left <= right) {
        int mid = (left + right)/2;
        if (data[mid] == key) {
            return mid;
        } else if (data[mid] < key) {
            left = mid + 1;
        } else {
            right = mid - 1;
        }
    }
    return data.length;
}
```
Black-box and white-box testing

Boundary conditions

// validate input
if ((year < CALENDAR_START) || (month < 1) || (month > 12)) {
    System.output.println("Bad request: " + year + " " + month);
    return;
}

Black-box and white-box testing

- Suggests the following boundary tests

<table>
<thead>
<tr>
<th>Input Year</th>
<th>Input Month</th>
</tr>
</thead>
<tbody>
<tr>
<td>1582</td>
<td>2</td>
</tr>
<tr>
<td>1583</td>
<td>0</td>
</tr>
<tr>
<td>1583</td>
<td>13</td>
</tr>
<tr>
<td>1583</td>
<td>1</td>
</tr>
<tr>
<td>1583</td>
<td>12</td>
</tr>
</tbody>
</table>
Black-box and white-box testing

- Path coverage or path testing—create test cases that causes each edge of the program’s controlflow graph to be executed

Example

```java
if (x != y) {
    y = 5;
} else {
    z = z - z;
}
if (x > 1) {
    z = z / x;
} else {
    z = 0;
}
```
Black-box and white-box testing

• Testing tips
  – Test early
  – Use inspections
  – Test boundaries
  – Test exceptional conditions
  – Make testing easily repeatable
Integration and system testing

• Integration testing is done as modules or components are assembled.
  – Attempts to ensure that pieces work together correctly
  – Test interfaces between modules
• System testing occurs when the whole system is put together
Debugging

- Use the scientific method
  - Gather data
  - Develop a hypothesis
  - Perform experiments
  - Predict new facts
  - Perform experiments
  - Prove or disprove hypothesis
Debugging

• Tips and techniques
  – Simplify the problem
  – Stabilize the error
  – Locate the error
  – Explain the bug to someone else
  – Recognize common bugs
    • Oversubscripting
    • Dereferencing null
  – Recompile everything
  – Gather more information
  – Pay attention to compiler warnings
  – Fix bugs as you find them
  – Take a break
GUI programming

Graphical user interface-based programming
Windchill

- Windchill
  - There are several formulas for calculating the windchill temperature $t_{wc}$
  - The one provided by U.S. National Weather Service and is applicable for a windspeed greater than four miles per hour

$$t_{wc} = 0.081(t - 91.4)(3.71\sqrt{v} + 5.81 - 0.25v) + 91.4$$

- Where
  - Variable $t$ is the Fahrenheit temperature
  - Variable $v$ is the windspeed in miles per hour
Console program

Method main() {
    statement_1;
    statement_2;
    ... statement_m;
}

Console programs begin and end in method main()
Program needs to respond *whenever* the run button is clicked.

There needs to be an event loop that is looking for user interface events.
GUI-based programming

GUI program begins in method main(). The method creates a new instance of the GUI by invoking the GUI constructor. On completion, the event dispatching loop is begun.

GUI Program

```java
main() {
    GUI gui = new GUI();
}
GUI Constructor() {
    constructor1;
    constructor2;
    ...  constructorn;
}
Action Performer() {
    action1;
    action2;
    ...  actionk;
}
```

Constructor configures the components of the GUI. It also registers the listener-performer for user interactions.

Event-dispatching loop

```
do
    if an event occurs
        then signal its action listeners
        until program ends
```

The event-dispatching loop watches for user interactions with the GUI. When an event occurs, its listener-performers are notified.

The action performer implements the task of the GUI. After it completes, the event-dispatching loop is restarted.
Java support

- JFrame
  - Represents a titled, bordered window

- JLabel
  - Represents a display area suitable for one or both of a single-line text or image.

- JTextField
  - Represents an editable single-line text entry component

- JButton
  - Represents a push button

- JTextArea
  - Represents an editable multiline text entry component
Instance variables

- private JFrame window
  - References the window containing the other components of the GUI
Instance variables

- private JTextArea legendArea
  - References the text display for the multiline program legend
Instance variables

- private JLabel fahrTag
  - References the label for the data entry area supplying the temperature
Instance variables

- private JTextField fahrText
  - References the data area supplying the temperature
Instance variables

- private JLabel windTag
  - References the label for the data entry area supplying the windspeed
Instance variables

- private JTextField windText
  - References the data area supplying the windspeed
Instance variables

- private JLabel chillTag
  - References the label for the data area giving the windchill
Instance variables

• private JTextField chillText
  - References the data area giving the windchill
Class constants

- private static final String LEGEND = "This windchill calculator" + "is intended for velocities greater than 4 mph."
  - Program legend text
Class constants

- private static final int WINDOW_WIDTH = 350
  - Initial width of the GUI
Class constants

- private static final int WINDOW_HEIGHT = 185
  - Initial height of the GUI
Class constants

- private static final int AREA_WIDTH = 40
  - Width of the program legend in characters
Class constants

- private static final int FIELD_WIDTH = 40
  - Number of characters per data entry area
Class constants

- private static final FlowLayout LAYOUT_STYLE = new FlowLayout()
  - References manager that lays out GUI components in a top-to-bottom, left-to-right manner
Class constants

- private static FlowLayout LAYOUT_STYLE = new FlowLayout()
  - References manager that lays out GUI components in a top-to-bottom, left-to-right manner
Class constants

- private static FlowLayout LAYOUT_STYLE = new FlowLayout()
  - References manager that lays out GUI components in a top-to-bottom, left-to-right manner

![Windchill Calculator](image_url)
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

public class Windchill implements ActionListener {

// class constants

// instance variables with initialization

// Windchill(): default constructor

// actionPerformed(): run button action event handler

// main(): application entry point

}
private static final int WINDOW_WIDTH = 350;   // pixels
private static final int WINDOW_HEIGHT = 185;  // pixels
private static final int FIELD_WIDTH = 20;     // characters
private static final int AREA_WIDTH = 40;      // characters
private static final FlowLayout LAYOUT_STYLE =
    new FlowLayout();
private static final String LEGEND = "This windchill "+ "calculator is intended for velocities greater than 4 mph.";
Program Windchill.java – instance variables

// window for GUI
private JFrame window =
    new JFrame("Windchill Calculator");

// legend
private JTextArea legendArea = new JTextArea(LEGEND, 2,
    AREA_WIDTH);

// user entry area for temperature
private JLabel fahrTag = new JLabel("Fahrenheit temperature");
private JTextField fahrText = new JTextField(FIELD_WIDTH);
Program Windchill.java – instance variables

// user entry area for windspeed
private JLabel windTag = new JLabel("       Windspeed (mph)");  
private JTextField windText = new JTextField(FIELD_WIDTH);

// entry area for windchill result
// entry area for windchill result
private JLabel chillTag =  
    new JLabel(" Windchill temperature");
private JTextField chillText = new JTextField(FIELD_WIDTH);

// run button
private JButton runButton = new JButton("Run");
Program Windchill.java – constructor

```java
public Windchill() {
    // configure GUI

    // register event listener

    // add components to container

    // display GUI
}
```
public Windchill() {
    // configure GUI
    window.setSize(WINDOW_WIDTH, WINDOW_HEIGHT);
    window.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    legendArea.setEditable(false);
    legendArea.setLineWrap(true);
    legendArea.setWrapStyleWord(true);
    legendArea.setBackground(window.getBackground());
    chillText.setEditable(false);
    chillText.setBackground(Color.WHITE);
Bad line wrapping

It is important to make program legends uneditable.
public Windchill() {
    // configure GUI
    window.setSize(WINDOW_WIDTH, WINDOW_HEIGHT);
    window.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    legendArea.setEditable(false);
    legendArea.setLineWrap(true);
    legendArea.setWrapStyleWord(true);
    legendArea.setBackground(window.getBackground());
    chillText.setEditable(false);
    chillText.setBackground(Color.WHITE);
Dangers of an editable legend

This windchill calculator is intended for velocities greater than 4 mph.

Fahrenheit temperature

Windspeed (mph)

Windchill temperature

Run

Line wrapping in the middle of a word
public Windchill() {
    // configure GUI
    window.setSize(WINDOW_WIDTH, WINDOW_HEIGHT);
    window.setSizeCloseOperation(JFrame.EXIT_ON_CLOSE);
    legendArea.setEditable(false);
    legendArea.setLineWrap(true);
    legendArea.setWrapStyleWord(true);
    legendArea.setBackground(window.getBackground());
    chillText.setEditable(false);
    chillText.setBackground(Color.WHITE);
Bad line wrapping

A JLabel is noneditable by the user.

By default the text field of a JTextField is editable by the user.
public Windchill() {
    // configure GUI ...

    // register event listener
    runButton.addActionListener(this);
}
Run button action-event handling

An ActionListener has an actionPerformer() method that handles the class-specific activity.

GUI : Action Listener

actionPerformer() Method

- Get data entries from temperature and windspeed data areas
- Compute windchill according to the Weather Service formula
- Display result to windchill data area
public Windchill() {
    // configure GUI ...

    // register event listener ...

    // add components to container
    Container c = window.getContentPane();
    c.setLayout(LAYOUT_STYLE);
    c.add(legendArea);
    c.add(fahrTag);
    c.add(fahrText);
    c.add(windTag);
    c.add(windText);
    c.add(chillTag);
    c.add(chillText);
    c.add(runButton);
Program Windchill.java – constructor

```java
public Windchill() {
    // configure GUI ...

    // register event listener ...

    // add components to container ...

    // make GUI visible
    window.setVisible(true);
}
```
Laying out the GUI components

This windchill calculator is intended for velocities greater than 4 mph.

- Fahrenheit temperature
- Windspeed (mph)
- Windchill temperature

Run
Program Windchill.java – action performer

```java
public void actionPerformed(ActionEvent e) {
    // get user's responses

    // compute windchill

    // display windchill
}
```
public void actionPerformed(ActionEvent e) {
   // get user's responses
   String response1 = fahrText.getText();
   double t = Double.parseDouble(response1);
   String response2 = windText.getText();
   double v = Double.parseDouble(response2);

   // compute windchill

   // display windchill
}
Program Windchill.java – action performer

This windchill calculator is intended for velocities greater than 4 mph.

Fahrenheit temperature 21
Windspeed (mph) 6
Windchill temperature

Run
public void actionPerformed(ActionEvent e) {
    // get user’s responses
    String response1 = fahrText.getText();
    double t = Double.parseDouble(response1);
    String response2 = windText.getText();
    double v = Double.parseDouble(response2);

    // compute windchill
    double windchillTemperature = 0.081 * (t - 91.4)
        * (3.71*Math.sqrt(v) + 5.81 - 0.25*v) + 91.4;

    int perceivedTemperature =
        (int) Math.round(windchillTemperature);

    // display windchill
}
public void actionPerformed(ActionEvent e) {
    // get user's responses
    String response1 = fahrText.getText();
    double t = Double.parseDouble(response1);
    String response2 = windText.getText();
    double v = Double.parseDouble(response2);

    // compute windchill
    double windchillTemperature = 0.081 * (t - 91.4)
        * (3.71*Math.sqrt(v) + 5.81 - 0.25*v) + 91.4;

    int perceivedTemperature =
        (int) Math.round(windchillTemperature);

    // display windchill
    String output = String.valueOf(perceivedTemperature);
    chillText.setText(output);
}
Program Windchill.java – action performer

This windchill calculator is intended for velocities greater than 4 mph.

**Fahrenheit temperature** 21

**Windspeed (mph)** 6

**Windchill temperature** 15
public static void main(String[] args) {
    Windchill gui = new Windchill();
}

{ Windchill Calculator

This Windchill Calculator is intended to calculate the windchill based on the given temperature and wind speed.

- Windchill Temperature
- Wind Speed (mph)
- Fahrenheit Temperature
- If velocity is greater than 4 mph,

    Windchill Calculator
Another method main()

public static void main(String[] args) {
    Windchill gui1 = new Windchill();
    Windchill gui2 = new Windchill();
}
This wind chill calculator is intended for velocities greater than 4 mph.

Fahrenheit temperature

Windspeed (mph)

Wind chill temperature

Run
Inheritance

• Definition
  – Ability to define a new class using an existing class as a basis

• Terms
  – Subclass
    • The new class inheriting the attributes and behaviors of the class on which it is based
  – Superclass
    • The basis for a subclass
Inheritance

• Default
  – Class has Object as its superclass

• Java allows only single inheritance
  – A class can be defined in terms of one other class
Inheritance

- Subclass can define
  - Additional attributes
  - Additional behaviors

- Subclass can override
  - Superclass behaviors
Applets

• Java program run within a browser
  – Implies an applet is run from a web page

• Applets may not access or modify the file system running the applet

• A modern applet has JApplet as its superclass
  – JApplet is part of the swing package

• An applet does use JFrame
  – A JApplet has a content pane
Applets

- Important inherited methods
  - init()
    - Run when browser loads applet
  - start()
    - Run by browser to start applet execution
  - stop()
    - Run by browser to stop its execution
  - destroy()
    - Run by browser immediately before it its ended
  - paint(Graphics g)
    - Run by browser to refresh its GUI

- By default the inherited methods do nothing
import java.awt.*;
import javax.swing.*;
public class DisplayQuote extends JApplet {
    public void paint(Graphics g) {
        g.drawString("Anyone who spends their life on a "
                     + " computer is pretty unusual.", 20, 30);
        g.drawString("Bill Gates, Chairman of Microsoft", 25, 45);
    }
}
Web page – quote.htm

<html>
    <title> Windchill </title>
    <applet code="DisplayQuote.class" width=400 height=300>
        </applet>
    </html>