cs205: engineering software
university of virginia
fall 2006

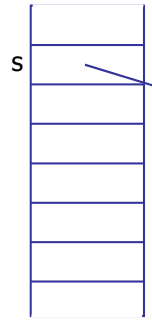
Semantics and Specifying Procedures

David Evans
www.cs.virginia.edu/cs205

Java Semantics

The Stack and Heap

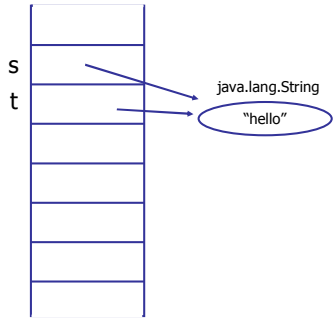
String s = new String ("hello");



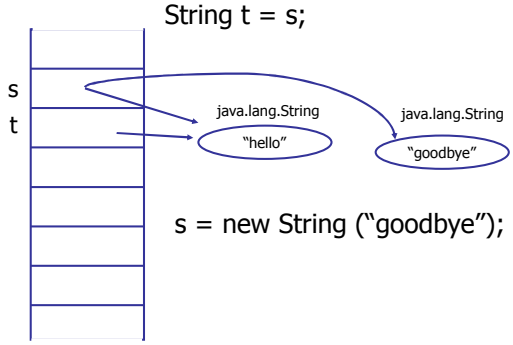
String is a type in the Java API for representing sequences of characters

Objects live on the heap
new creates an object on the heap
Local variables live on the stack
May point to objects on the heap

String s = new String ("hello");
String t = s;



String s = new String ("hello");
String t = s;



s = new String ("goodbye");

Primitive Types

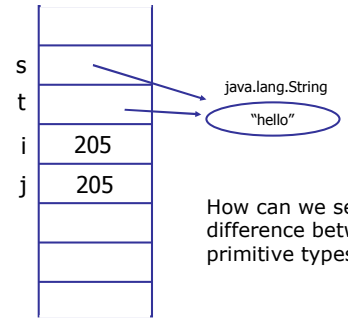
- Not everything in Java is an Object
- Some types are *primitive types*
  - **boolean**, byte, char, **double**, float, **int**, long, short
- Values of primitive types are stored directly on the stack

String s = new String ("hello");

String t = s;

int i = 205;

int j = i;



How can we see the difference between primitive types and objects?

Equality

x == *y*

- Object Types: same objects
- Primitive Types: same value

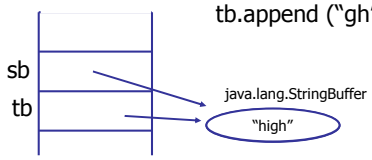
x.equals (*y*)

- Object Types: method that compares values of objects
- Primitive Types: doesn't exist

Mutability

- If an object is mutated, all references to the object see the new value

StringBuffer sb = new ("hi");
StringBuffer tb = sb;
tb.append ("gh");



Immutable/Mutable Types

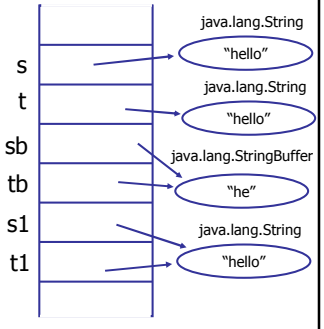
- Types can be mutable or immutable
  - Objects of an immutable type never change value after they are created
- String is immutable, StringBuffer is mutable
  - String.concat creates a new String object
  - StringBuffer.append mutates the old object


```
public class Strings {
   public static void test (String [] args) {
      String s = new String ("hello");
      String t = new String ("hello");
      StringBuffer sb = new StringBuffer ("he");
      StringBuffer tb = sb;
      String s1 = "hello";
      String t1 = "hello";

      sb.append ("llo");
      tb.append (" goodbye!");
      s.concat (" goodbye!");
      t = s.concat (" goodbye!");

         // What are the values of s, t, sb and tb now?
         // Which of these are true:
         //    a) s == t  b) s1 == t1  c) s == s1 d) s.equals (t)
         //    e) sb == tb   f) t.equals (tb)
   }
}
```


Java Semantics Question

```
public class Strings {
   public static void test () {
      String s = new String ("hello");
      String t = new String ("hello");
      StringBuffer sb = new StringBuffer
                        ("he");
      StringBuffer tb = sb;
      String s1 = "hello";
      String t1 = "hello";

      sb.append ("llo");
      tb.append (" goodbye!");
      s.concat (" goodbye!");
      t = s.concat (" goodbye!"); } }
```

String spec is not enough to determine if s, t, s1 and t1 are the same objects.

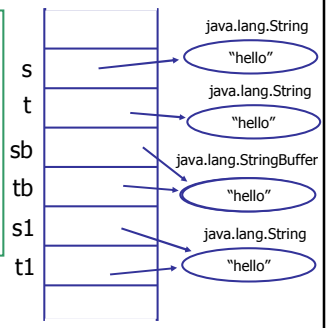


Java Language Specification
(Section 3.10.5: String Literals)

Each string literal is a reference (§4.3) to an instance (§4.3.1, §12.5) of class String (§4.3.3). String objects have a constant value. String literals-or, more generally, strings that are the values of constant expressions (§15.28)-are "interned" so as to share unique instances, using the method String.intern.

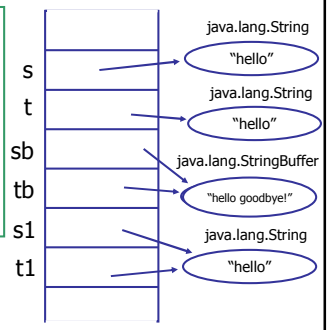
Java Semantics Question

```
public class Strings {
   public static void test () {
      String s = new String
("hello");
      String t = new String ("hello");
      StringBuffer sb = new
StringBuffer              ("he");
      StringBuffer tb = sb;
      String s1 = "hello";
      String t1 = "hello";

      sb.append ("llo");
      tb.append (" goodbye!");
      s.concat (" goodbye!");
      t = s.concat (" goodbye!"); } }
```



Java Semantics Question

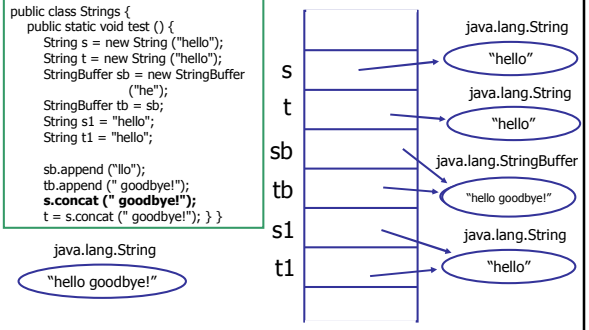
```
public class Strings {
   public static void test () {
      String s = new String ("hello");
      String t = new String ("hello");
      StringBuffer sb = new StringBuffer
                ("he");
      StringBuffer tb = sb;
      String s1 = "hello";
      String t1 = "hello";

      sb.append ("llo");
      tb.append (" goodbye!");
      s.concat (" goodbye!");
      t = s.concat (" goodbye!"); } }
```



Java Semantics Question

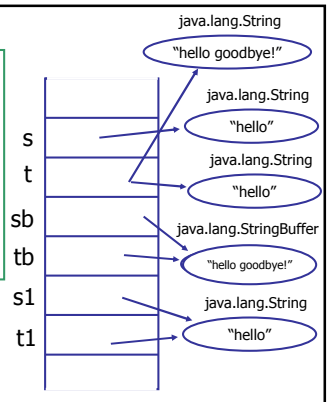
```
public class Strings {
   public static void test () {
      String s = new String ("hello");
      String t = new String ("hello");
      StringBuffer sb = new StringBuffer
                ("he");
      StringBuffer tb = sb;
      String s1 = "hello";
      String t1 = "hello";

      sb.append ("llo");
      tb.append (" goodbye!");
      s.concat (" goodbye!");
      t = s.concat (" goodbye!"); } }
```

java.lang.String
"hello goodbye!"




```
public class Strings {
   public static void test () {
      String s = new String ("hello");
      String t = new String ("hello");
      StringBuffer sb = new StringBuffer
                ("he");
      StringBuffer tb = sb;
      String s1 = "hello";
      String t1 = "hello";

      sb.append ("llo");
      tb.append (" goodbye!");
      s.concat (" goodbye!");
      t = s.concat (" goodbye!"); } }
```

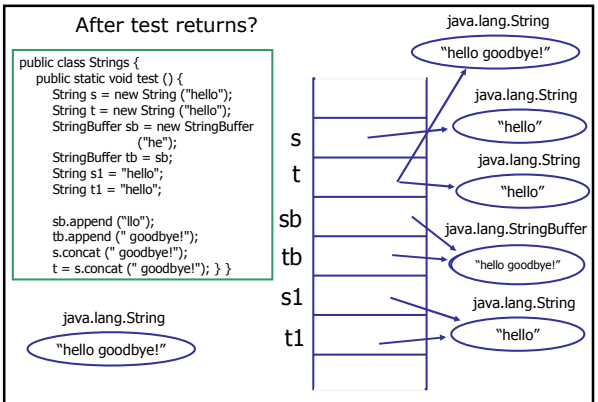
java.lang.String
"hello goodbye!"



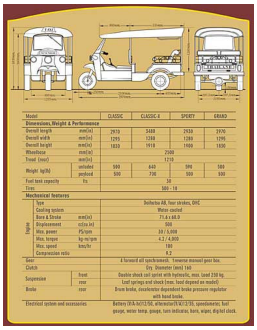
After test returns?

```
public class Strings {
   public static void test () {
      String s = new String ("hello");
      String t = new String ("hello");
      StringBuffer sb = new StringBuffer
                ("he");
      StringBuffer tb = sb;
      String s1 = "hello";
      String t1 = "hello";

      sb.append ("llo");
      tb.append (" goodbye!");
      s.concat (" goodbye!");
      t = s.concat (" goodbye!"); } }
```

java.lang.String
"hello goodbye!"



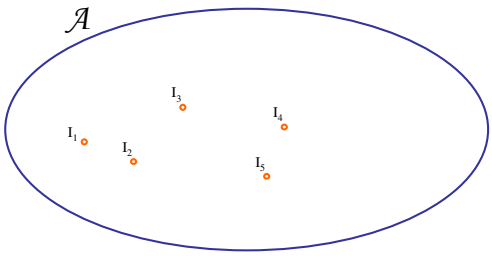
Abstraction by Specification



Managing Complexity

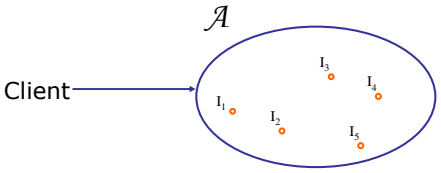
- Divide problem into subproblems that
  - Can be solved independently
  - Can be combined to solve the original problem
- How do we know they can be solved independently?
- How do we know they can be combined to solved the original problem?

Abstraction



An abstraction is a *many-to-one* map.

Using Abstractions



When a client uses an abstraction, it *should work as the client expects it to* no matter with implementation is provided.

How should client know what to expect?

Specification

- Tells the client of an abstraction what the client can expect it to do
- Tells the implementer of an abstraction what the implementation must do to satisfy the client
- Contract between client and implementer:
  - Client will only rely on behavior described by specification
  - Implementer will provide an implementation that satisfies the specification

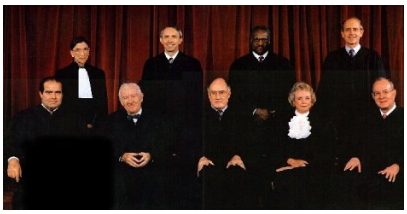
Good Specifications

- Clear, precise and unambiguous
  - Clients and implementers will agree on what they mean
- Complete
  - Describe the behavior of the abstraction in all situations
- Declarative
  - Describe *what* the abstraction should do, not *how* it should do it

Formality of Specifications

- Informal: written in a natural language (e.g., English)
  - People can disagree on what it means
  - Degrees of informality
- Formal: written in a specification language
  - Meaning is defined by specification language (whose meaning is defined precisely, but eventually informally)
  - May be analyzed by machines

What do you call people who decide what informal specifications mean?



Example Informal Specification

Excessive bail shall not be required, nor excessive fines imposed, nor cruel and unusual punishments inflicted.

8th Amendment

Correct Implementation?

```
public static boolean
  violatesEigthAmendment (Punishment p) {
    // EFFECTS: Returns true if p violates the 8th
    //     amendment: cruel and unusual
    //     punishments.
    return (p.isCruel () && p.isUnusual ());
}
```

Or did they mean p.isCruel () || p.isUnusual () ?

Procedural Specifications

- Specification for a procedure describes:
  - What its inputs are
  - What the mapping between inputs and outputs are
  - What it can do the state of the world

Requires and Effects

- Header: name of procedure, types of parameters and return value
  - Java declaration
- Clauses (comments in Java)
  - REQUIRES - **precondition** the client must satisfy before calling
  - EFFECTS – **postcondition** the implementation satisfy at return

Contract

- Client promises to satisfy the precondition in the requires clause
- Implementer promises if client satisfies the precondition, the return value and state when the function returns will satisfy the postcondition.

Specification Contract

f ()
REQUIRES: *precondition*
EFFECTS: *postcondition*

precondition
{ f (); }
postcondition

If the precondition is true, after we call f (), the postcondition is true.

Specification Example

```
public String bestStock ()
    // REQUIRES: false
    // EFFECTS: Returns the name of the
    //       best stock to buy on the NASDAQ
    //             tomorrow.
```

Can we implement a procedure that satisfies this specification?

Yes, any implementation will satisfy this specification! If the precondition in the requires clause is not satisfied, the procedure can do **anything** and still satisfy its specification!

Specification Example

```
public String bestStock ()
    // REQUIRES: true
    // EFFECTS: Returns the name of the
    //       best stock to buy on the NASDAQ
    //             tomorrow.
```

Can we implement a procedure that satisfies this specification?

Requires Clauses

- The *weaker* (more easy to make true) the requires clause:
  - The more useful a procedure is for clients
  - The more difficult it is to implement correctly
- Avoid requires clauses unless there is a good reason to have one
  - Default requires clause is: REQUIRES true
  - Client doesn't need to satisfy anything before calling

Specification Example

```
public static int biggest (int [ ] a)
   // REQUIRES: true
   // EFFECTS: Returns the value of the
   //       biggest element of a.
```

Is this a reasonable specification?

No, what should client expect to happen if a is empty.

Specification Example

```
public static int biggest (int [ ] a)
  // REQUIRES: a has at least one element.
  // EFFECTS: Returns the value of the
  //       biggest element of a.
```

Is this a good specification?

Maybe, depends on the client. Its risky…

Specification Example

```
public static int biggest (int [ ] a)
  // REQUIRES: true
  // EFFECTS: If a has at least one
  //      element, returns the value biggest
  //      element of a. Otherwise, returns
  //      Integer.MIN_VALUE (smallest int
  //      value).
```

Better, but client has to deal with special case now. Best would probably be to use an exception…

Bad Use of Requires Clause

- Bug discovered in Microsoft Outlook that treats messages that start with "begin " as empty attachments (can be exploited by viruses)

To workaround this problem:
- Do not start messages with the word "begin" followed by two spaces.
- Use only one space between the word "begin" and the following data.
- Capitalize the word "begin" so that it is reads "Begin."
- Use a different word such as "start" or "commence".

from http://support.microsoft.com/default.aspx?scid=KB;EN-US;Q265230&
(this is no longer available, was "revoked" by Microsoft)

Modifies

- How does a client know a is the same after biggest returns?

```
public static int biggest (int [ ] a)
  // REQUIRES: true
  // EFFECTS: If a has at least one element,
  //      returns the value biggest element of a.
  //      Otherwise, returns Integer.MIN_VALUE
  //      (smallest int value).
```

Reading the effects clause is enough – if biggest modifies anything, it should describe it. But, that's a lot of work.

Modifies

- Modifies clause: any state not listed in the modifies clause may not be changed by the procedure.

```
public static int biggest (int [ ] a)
  // REQUIRES: true
  // MODIFIES: nothing
  // EFFECTS: If a has at least one element,
  //      returns the value biggest element of a.
  //      Otherwise, returns Integer.MIN_VALUE
  //      (smallest int value).
```


Modifies Example

```
public static int replaceBiggest (int [ ] a, int [] b)
  // REQUIRES: a and b both have at least one
  //    element
  // MODIFIES: a
  // EFFECTS: Replaces the value of the biggest
  //      element in a with the value of the biggest
  //      element in b.
```

Defaults

- What should it mean when there is no requires clause?

  REQUIRES: true

- What should it mean when there is no modifies clause?

  MODIFIES: nothing

- What should it mean when there is no effects clause?

  Meaningless.

Charge

- Specifications in CS205
  - Will be informal: written in English (aided by common mathematical notations)
  - ...but must be precise and clear
  - REQUIRES/MODIFIES/EFFECTS style
- Reading before next class:

  Chapters 3 and 9