Object Oriented Programming Languages

What are the benefits?

- Systems more adaptive to change
  - Information hiding
  - Encapsulation
- Appeals to human cognition
- Reuse
  - Gamma et al.: Design Patterns
- Methodologies for design and analysis
  - Grady Booch: Object Oriented Design & Analysis
  - CASE tools

What is an OOPL?

- No consensus
- Cardelli + Wegner:
  OO = ADTs + first class objects + types inheriting attributes from a supertype
- Finkel:
  OO = encapsulation + inheritance + overloading

Common features

- Classes
- Objects (instantiation)
- (Single) Inheritance + redefinition
- Type inheritance
- Multiple inheritance
- Method overloading
- Dynamic lookup
- Deferred implementations
- Protection

Other features

- Static typing, Genericity - Unit 8
- Exception handling, Garbage collection, Design by contract - Unit 9
- Multiple polymorphism
- Reflection + Meta-object protocols
- Packages
- Class methods
- Iterators
- Operator overloading

Classes

- A class is “a software element that describes an ADT and its partial or total definition” (Meyer)
- Can be viewed as a set of members (or slots), which can be:
  - Data (attributes or fields)
  - Operations (methods, messages)

C++ Example:

```
class FixedIntStack {
   int index, *arr;
public:
   FixedIntStack(int size) {
    index = 0;
     arr = new int[size];
   }
   ~FixedIntStack() { delete arr; }
   void push(int item){ arr[index++] = item; }
   int pop() { return arr[--index]; }
};
```

Instantiation

- Class instances are called objects, and are first class entities.
- Constructors run at instantiation time.
- Arguments can be passed to instantiations.
- Instantiation is usually handled by an operator
  - ! In Eiffel
  - **new** in C++ and Java
- Deallocation may call a destructor (or finalization method)

Instantiation Example

C++

```
FixedIntStack *stack1 = new FixedIntStack(10);
FixedIntStack stack2(10);
stack1->push(5);
stack2.push(5);
```

Java

```
FixedIntStack myStack = new FixedIntStack(10);
myStack.push(5);
```

A trend: objects PBR, primitive types by value

Inheritance

- Incremental program extension
  `class Integer inherit Number`
- `Number` is the *parent* of `Integer`
- `Integer` is the *child* or *descendant* of `Number`
- multiple inheritance
  `class Mobile_Home inherit Vehicle, House`

Subclassing ≠ subtyping ≠ is a

- Subclassing - code & data sharing
- Subtyping - type sharing
  - *substitutability* - a subtype may stand in for any parent type
  - polymorphism - `void print(Object ob)`
- Specialization (*is a*) - implies subtyping and subclassing

Java Example

```
class Rectangle { // default: extends Object
  float width, height;
  void set_width(float w) { width = w; }
  void set_height(float h) { height = h; }
}
class Square extends Rectangle {
  // These are overridden (redefined) methods:
  void set_width(float w) {
    super.set_width(w);
    super.set_height(w);
  }
  void set_height(float w) {
    set_width(float w);
  }
```

Method lookup (binding)

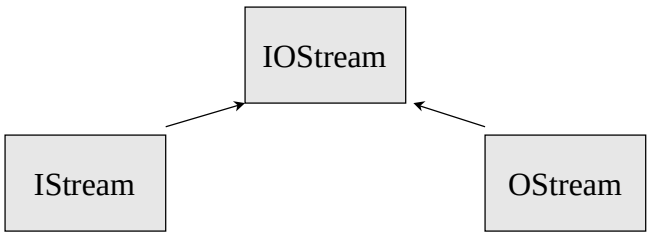
```
Square s = new Square();
s.set_width(12); // Meaning is obvious
Rectangle r = s; // substitutability!
r.set_width(12);
```

- Dynamic lookup: Square's method is called.
- Static lookup: Rectangle's method is called

- Java: Dynamic Lookup
- C++: `virtual` keyword

- Liskov substitution principle

Multiple Inheritance



More from John Viega next Tuesday...

Protection

- A way to enforce *encapsulation.*
- C++:
  - no change by default
  - private, public, protected members
  - private, public, protected inheritance
  - break the rules with `friend`

```
class X : protected Y, private Z {
  friend class Q;
  public:
   X();
  private:
     int datum;
}
```

Protection

- Eiffel:
  - No change by default
  - secret + non-secret members
  - selectively export operations to classes

```
class MySecrets inherit GovtSecrets export
    {NONE} really_big_secret
    {ANY} common_knowledge
feature {NSA, CIA, FBI} big_secret : Secret
end
```