Object-Oriented Programming

CSCI 334
Stephen Freund

Operator Overloading

```java
class Vector {
    int data[10];
    int size;
    // overload * to be dot-product on vectors
    int operator*(Vector other) {
        int result = 0;
        for (int i = 0; i < size; i++) {
            result += data[i] * other.data[i];
        }
        return result;
    }
}
Vector v1;
Vector v2;
... int dot_prod = v1 * v2;
```

Object-oriented programming

- Programming methodology
  - organize concepts into objects and classes
  - build extensible systems
- Language concepts
  - encapsulate data and functions into objects
  - subtyping allows extensions of data types
  - inheritance allows reuse of implementation

Object-oriented Method

(Bosch)

- Four steps
  - Identify the objects at a given level of abstraction
  - Identify the semantics (intended behavior) of objects
  - Identify the relationships among the objects
  - Implement these objects
- Iterative process
  - Implement objects by repeating these steps
- Not necessarily top-down
  - “Level of abstraction” could start anywhere

Growing A Language

• Guy Steele

This Method

- Based on associating objects with components or concepts in a system
- Why iterative?
  - An object is typically implemented using a number of constituent objects
  - Apply same methodology to subsystems, underlying concepts
Comparison to top-down design

- **Similarity:**
  - Iterative process of refinement

- **Differences:**
  - Focus of top-down design is on data structure
  - OO methods are based on modeling ideas
  - Combining functions and data into objects makes data refinement more natural

Subtyping and Substitutivity

```java
class Rectangle {
    private int x,y,w,h;
    void moveTo(int x, int y);
    void setSize(int width, int height);
    void show();
    void hide();
}

class FilledRectangle {
    private int x,y,w,h;
    private Color c;
    void moveTo(int x, int y);
    void setSize(int width, int height);
    void show();
    void hide();
    void setFillColor(Color color);
    Color getFillColor();
}
```

Rectangles Revisited

```java
class Rectangle {
    int x,y,w,h;
    void moveTo(int x, int y);
    void setSize(int width, int height);
    void show();
    void hide();
}

class FilledRectangle extends Rectangle {
    Color c;
    void setFillColor(Color color);
    Color getFillColor();
}
```

Java Subtyping (Sneak Preview...)

```java
interface MouseListener {
    void onMousePress(MouseEvent e);
    void onMouseRelease(MouseEvent e);
    void onMouseMove(MouseEvent e);
    void onMouseClick(MouseEvent e);
    ...
}

class StockTicker extends Applet implements MouseListener {
    ...
}
```

OO Program Structure

- **Group data and functions**
- **Class**
  - Defines behavior of all objects that are instances of the class
- **Subtyping**
  - Place similar data in related classes
- **Inheritance**
  - Avoid reimplementing functions that are already defined
Example: Geometry Library
• Define general concept: Shape
• Implement two shapes: Circle, Square
• Functions on implemented shapes: center, move, rotate, draw
• Anticipate additions to library

Shapes
• Interface of every Shape must include:
  center, move, rotate, draw

• Different kinds of shapes are implemented differently:
  - Square: two points, representing corners
  - Circle: center point and radius

Subtype Hierarchy
- General interface defined in the Shape class
- Implementations defined in Circle, Square
- Extend hierarchy with additional shapes

Example Use: Processing Loop
Vector<Shape> shapes = new Vector<Shape>();
...
for (i = 0; i < shapes.size(); i++) {
  Shape s = shapes.get(i);
  s.move(10, 10);
}
Control loop does not know the type of each shape

Code Placed In Classes

<table>
<thead>
<tr>
<th></th>
<th>center</th>
<th>move</th>
<th>rotate</th>
<th>print</th>
</tr>
</thead>
<tbody>
<tr>
<td>Circle</td>
<td><code>c_center</code></td>
<td><code>c_move</code></td>
<td><code>c_rotate</code></td>
<td><code>c_draw</code></td>
</tr>
<tr>
<td>Square</td>
<td><code>s_center</code></td>
<td><code>s_move</code></td>
<td><code>s_rotate</code></td>
<td><code>s_draw</code></td>
</tr>
</tbody>
</table>

• Dynamic lookup:
  - `circle.move(x, y)` calls function `c_move`