Object-Oriented Model

Object-oriented analysis (OOA), design (OOD), and programming (OOP)

"Object-oriented modeling and design is a new way of thinking about problems using models organized around real-world concepts. The fundamental construct is the object, which combines both data structure and behavior in a single entity."

James Rumbaugh, Object-Oriented Modeling and Design

= Data Structure (more commonly called an attribute)
  - variable
  - instance field
  - data
  - data field

= Behavior
  - method
  - member function
  - function
  - operation
  - state
  - instance method

Class Code vs. Client Code

An approach to creating and using objects

```java
public class Point {
    Point p1 = new Point(100, 50);
    Point p2 = new Point(25, 75);
    p1.drawLine(p2);
}
```

OO Definitions

Defining characteristics

= Classes
  - Describe "things" with similar attributes and behaviors
  - Template, blueprint, or cookie cutter
  - Implements Abstract Data Types (ADT)

= Objects / Instances
  - Specific instance of a class (i.e., the cookie)
  - Each object has a distinct identity or handle

= Programs are a group of cooperating objects
  - Objects are/have data
  - Have behaviors (i.e., can do things)

UML Class Example

The diagram

```
  stereotype  -->
  protected -->
    private -->

Person
    + name: String
    + height: int
    + weight: int
    + instanceCount: int

  + constructor()
  + Person()
  + Person(name: String)
  + payTaxes()
  + catchBus(direction: int)
  + help()
  - takeLongLunch(): boolean
```

= Class name
  - abstract
  - concrete

= Attributes
  - instance variables
  - class variables
  - static

= Operators
  - behaviors
  - methods
  - functions

UML Class Example

The Java code

```java
public class Person {
    private String name;
    public int height;
    protected int weight;
    final static int instanceCount = 0;

    public Person(String name) {} }

    public void payTaxes() {} }
    public void catchBus(int direction) {} }
    private boolean takeLongLunch() {} }

    public class Client Code {
    public class Person("Fred");
    p.height = 70;
    p.payTaxes(); }

    Data is associated with a specific object (p)
    Methods must be called through an object (p)
Attributes and Behaviors

The fundamental elements of classes and objects

- Attributes (state)
  - Characterize or distinguish an object
  - Are a data values (variables) held by objects
  - Each object maintains a private copy of each attribute value
- A behavior is a method called through an object
  - The object is the default target of the method
- Access methods set or get attributes; some further classify as
  - Accessors, getffer, or selector methods get attributes (gets object's state)
  - Preferred Java style: `getAttribute( )`
  - Mutator or setter methods set attributes (modifies an object's state)
  - Preferred Java style: `void setAttribute(attributeType v)`
- Attributes and operations should represent an atomic concept
  Also called cohesion

Access Methods Example

Java code from UML diagram

```java
public class Person
{
    private int height;
    public int getHeight() // accessor
    {
        return height;
    }

    public void setHeight(int h) // mutator
    {
        if (h > 18 && h < 108) // class invariant
            height = h;
        else
            // ERROR
    }
}
```

Abstraction

Separating use from implementation

- "Focusing on the essential properties rather than the inner details"
- Method
  - User "sees" the method interface or signature (name, return type, and parameters)
  - User does not see how the method does its work (i.e., algorithms or implementation)
- Class
  - User does not see what data an object contains or how it is organized
  - User sees only the public interface (i.e., the public methods)

Abstraction Illustrated

User utilizes an object without knowing how it works

- Data is not visible from the outside
- Method implementations (i.e., the algorithms or bodies) are not visible from the outside
- Only the "public interface" (i.e., the signatures of the public methods) are visible from the outside

The this Reference

The "implicit" parameter

- "Secret" parameter passed to each non-static method
- Points to the calling object; binds it's data to the method
- Used for self reference

```java
public class MyPoint
{
    int x, y;
    void print()
    {
        System.out.println(x + " * " + y);
    }
}
```

```java
class Test
{  
    MyPoint p1 = new MyPoint(100, 17);
    MyPoint p2 = new MyPoint(37, 19);
    MyPoint p3 = new MyPoint(47, 39);
    public void display()
    {
        p2.print();
    }
}
```

Constructors

Special purpose methods

```java
public class MyPoint
{  
    int x, y;
    
    public MyPoint(int aX, int aY)
    {     
        x = aX;
        y = aY;
    }
}
```

- Constructors
  - No return type or value
  - Have the same name as the class
  - Multiple constructors are allowed but each must have a unique signature or argument list (i.e., method overloading)
  - Default constructors do not have arguments
  - Invoked with the new operator (i.e., they initialize instantiated objects' data members or attributes)
**Constructor Example**

```
public class MyPoint {
    int x, y;
    
    public MyPoint() {
        x = 100;
        y = 100;
    }
    
    public MyPoint(int aX, int aY) {
        x = aX;
        y = aY;
    }
}
```

Point P1 = new MyPoint();
Point P2 = new MyPoint(50,25);

---

**More On Constructors and this**

Constructors can call each other

```
public class Circle {
    int radius = 100;
    
    public Circle(int radius) {
        this.radius = radius;
    }
    
    public Circle() {
        this(100);
    }
    
    public void draw() {
    }
}
```

---

**The toString Method**

A method for all classes

- Converts some or all attributes / fields to strings
  - public String toString() 
  - This signature NEVER changes

```
public class Person {
    
    public int height;
    public String name;
    
    public String toString() {
        return name + " " + height;
    }
}
```

- equals an instanceof covered in chap 9

---

**Java Program File Structure**

A good style helps to find and to avoid errors

```
public class Demo {
    
    public void Method1( ) {
        instance variables / data fields
        method body goes here
    }
    
    public void Method2( ) {
        instance variables / data fields
        method body goes here
    }
    
    public class Person {
        public int height;
        public String name;
    }
}
```

Java classes are large because they include method bodies