CSCI 112
Friday, July 2, 2010
Instructor: James Church

Classes

The Class and the Object

Superhero – example of a class
Superman – example of an object

Plato's Theory of Forms

1. Everything is based on a form.
2. That form does not exist in this world.
3. The best we can do is study examples of the form.

Object-Oriented Design Principles

In object-oriented programming we have the opportunity to design our own forms (aka classes).

1. **Modularity –** we can make multiple versions of an object
2. **Abstraction** – there is a layer of abstraction between you and the class. The methods and fields should represent a generic version of the class.
3. **Encapsulation –** possibly the most important part. When trying to design our classes, we want to try and keep how the class works private. The user knows as little about the underlying algorithms and data structures as possible. In other worlds, *keep details private.*

Object-Oriented Goals

1. **Robustness –** software should be able to handle exact inputs, as well as inputs that may not be exactly how we inteneded but are still correct. An example would be how a user might input the date:

   • July 2, 2007
   • Jul 2, 2007
   • 07/02/2007
   • 07/02/07
   • 2 July 2007
   • 07-02-2007
   • 07022007
   • 2007-07-02
   • 20070702

2. **Adaptability** – Ability to quickly change when needed.
3. **Reusability –** the idea of code reuse. Our code should work in as many paridigms as possible.

Designing a Class

- **Class Name** should be a noun
- **Method Names** should be verbs

Public and Private

Private fields – keeps data encapsulated
Publc fields – fields are open. No encapsulation.

Class Example (Bicycle)

Class

```
public class Bicycle {
        public class Bicycle {
                private int gear;
                private int speed;

                // Constructor: first method called in a class
                // Constructor has no return type, is also named same as class
                public Bicycle(){
                        gear = 1;
                        speed = 0;
                }

                // Getter (Accessor) Methods: get data from field
                public int getGear(){
                        return gear;
                }
                public int getSpeed(){
                        return speed;
                }

                // Setter (Mutator) Methods: change data in field
                public void setGear( int gear ){
                        if( gear >= 1 && gear <= 7 ){
                                this.gear = gear;
                        }
                }
                public void setSpeed( int speed ){
                        if( speed >= 0 ){
                                this.speed = speed;
                        }
                }

                // toString method
                public String toString(){
                        return "Gear: " + gear + " Speed: " + speed;
                }
}
```

Test Class

```
import java.util.*;

public class BikeTest{
```

```
        public static void main( String[] args ){
                Bicycle a = new Bicycle();
                Bicycle b = new Bicycle();

                a.setSpeed( 10 );
                a.setGear( 2 );

                b.setSpeed( -10 );  //silent failure, no error message, speed stays where it was
                b.setGear( 3 );

                System.out.println( a );
                System.out.println( b );
        }
}
```

Example Code (Coin.java)

```
public class Coin{
   private final int HEADS = 0;
   private int face;

   //constructor
   public Coin(){
      flip();
   }

   public void flip(){
      face = ( int )( Math.random() * 2 );
   }

   public boolean isHeads(){
      return ( face == HEADS );
   }

   public String toString(){
      return ( face == HEADS )? "Heads" : "Tails"
   }
}
```

CoinTest.java

```
import java.util.*;

public class CoinTest{
   public static void main( String[] args ){
      int headCount = 0;
      int tailCount = 0;
      Coin penny = new Coin();
      final int FLIPS = 100;

      for( int i = 0; i < FLIPS; i++ ){
```

```
            penny.flip();
            System.out.println( i + ": " + penny );

            if( penny.isHeads() ){
               headCount++;
            }
            else{
               tailCount++;
            }
         }
         System.out.println( "Heads: " + headCount );
         System.out.println( "Tails: " + tailCount );
      }
}
```

Static vs. Non-Static

Static is associated with the class.
Non-Static is associated with the object.

Java requires that we always start out in a static context.

You can modify static from non-static, but you can't do things the other way around.

Example Code (MyClass.java)

```
public class MyClass{
   private static int classf;
   private int objectf;

   public MyClass(){
      class f = 10;
      objectf = 10;
   }

   public void reset(){
      classf = 10;
      objectf = 10;
   }

   //class method
   public static void classm(){
      classf = 20;
   }

   //object method
   public void objectm(){
      objectf = 20;
   }

   public void both(){
```

```
        classf = 30;
        objectf = 30;
    }

    public String toString(){
        return "classf: " + classf + " objectf: " + objectf;
    }
}
```

snstest

```
public class snstest{
    public static void main( String[] args ){
        MyClass a = new MyClass();
        MyClass b = new MyClass();

        a.objectm();
        b.classm();

        System.out.println( "a: " + a );
        System.out.println( "b: " + b );

        MyClass.classm();  //also allowed because classm() only affects the static field
    }
}
```