Creating Objects From Scratch

Constructor Methods, Instance Variables, and Class Variables

Friday, November 16, 2007

CS111 Computer Programming

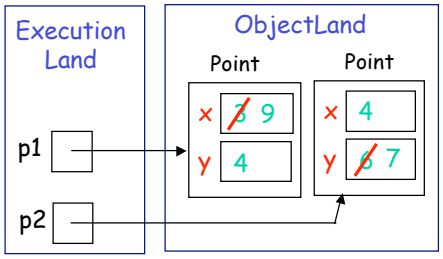
Department of Computer Science
Wellesley College

A Simple Example: Points

Suppose we want to describe Point objects with two instance variables named x and y.

```
Point p1 = new Point(3,4);

System.out.println(
  "p1 = " + p1.toString());

Point p2 = new Point(p1.y, 2*p1.x);

System.out.println("p2 = " + p2);

p1.x = p2.x + 5;

p2.y++;

System.out.println("p1 = " + p1);

System.out.println("p2 = " + p2);
```

Point (unlike Location) components can change over time.



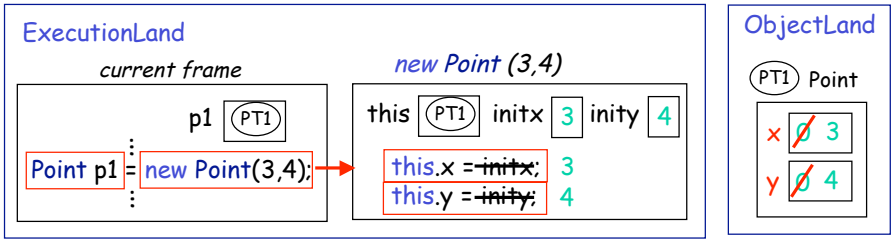
Console

```
p1 = (3,4)
p2 = (4,6)
p1 = (9,4)
p2 = (4,7)
```

Declaring the Point Class

```
public class Point { // by default, extends the Object class

   // Instance Variables
   public int x;  // x coordinate
   public int y;  // y coordinate

   // Constructor Method
   public Point (int initx, int inity) {
      this.x = initx; // set x coordinate to initx
      this.y = inity; // set y coordinate to inity
      // the "this." is optional above and elsewhere
   }

   // Instance Method
   public String toString() {
      return "(" + this.x + "," + this.y + ")";
   }

}
```


Constructor Methods in a JEM



Steps in the invocation of a constructor method:

1. Invoking a constructor method creates an execution frame with parameter variables.
2. A new instance of the class is created in ObjectLand with the instance variables specified in the class initialized to the default values of their type.
3. A reference to the new instance is stored in a variable named **this** in the constructor method execution frame.
4. The body of the constructor method is executed.
5. The value in **this** is returned as the result of the constructor method.

public vs. private Instance Variables

```
public class Color { // Color instances have red, green, and blue
                     // components that cannot be changed.
   // Instance Variables
   // Make these private in order to control access to them.
   private int r, g, b;  // red, green, and blue values (0 to 255)

   // Constructor method
   public Color (int ir, int ig, int ib) {
      if ((0 <= ir) && (ir <= 255) && (0 <= ig) && (ig  <= 255)
            && (0 <= ib) && (ib <= 255)) {
         r = ir; g = ig; b = ib;
      } else { ... indicate an error ... }
   }

   // Instance Methods
   public int getRed () { // we can get red component, but not set it.
      return r;
   }

   // Other instance methods go here
}
```


Buggles from Scratch!

Oh no! You've accidentally deleted your Buggle class! Now you have to create it from scratch. (Don't worry about graphics, walls, and bagels.)

```
public class Buggle {
   // Put your instance variables here




   // Put your constructor method here




   // Define your instance methods on the next few slides
}
```

Some Buggle Instance Methods

```
// Some instance methods in the Buggle class
public Location getPosition () {




}

public Location setPosition (Location loc) {




}
// getHeading(), setHeading(), getColor(), setColor()
// brushUp(), and brushDown() are similar.
```


More Buggle Instance Methods

```
// More instance methods in the Buggle class
// Some methods from the Direction contract are helpful for these!
public Location left () {


}

public Location forward () { // Don't worry about walls



}

public Location forward (int n) { // Don't worry about walls



}
```

Extending Buggles with New State

Let's define a CountingBuggle class with an instance variable named count used in the following instance method:

```
public int distanceFromWall() {
  this.count = 0;
  while (!this.isFacingWall()) { // Assume we can handle walls
     this.count++;
     this.forward();
  }
  this.backward(this.count);
  return this.count;
}
```

For example:

```
CountingBuggle counter = new CountingBuggle();
counter.brushUp();
System.out.println(counter.distanceFromWall());
```

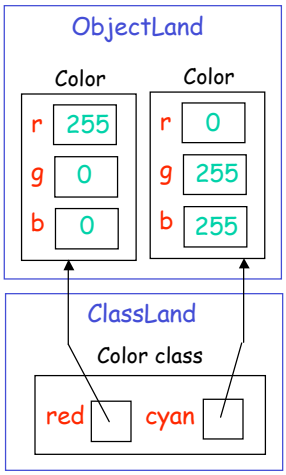

Defining the CountingBuggle Class

```
public class CountingBuggle extends Buggle {
   // New instance variable (in addition to those of Buggles)


   //  Constructor method
   public CountingBuggle() {
      // Implicitly executes the body of the Buggle() constructor
      // and then executes the body below:


    }

    // Instance method
   public int distanceFromWall() {… as defined on previous slide …}

}
```

Class Variables

Class (static) variables belong to the class, not to any instance. We will show them in a new area called ClassLand (where classes live, of course!)

```
public class Color {

  // Class variable declarations
  public static final Color red = new Color(255,0,0);
  public static final Color cyan = new Color(0,255,255);
  // … more color class variables defined here …

  // Other declarations go here.

}
```



The **final** keyword means the variable value cannot be changed by assignment once they are initialized.

SerialBuggles

Suppose we want to create a SerialBuggle class in which each instance has a unique ID. E.g.:

```
SerialBuggle sally = new SerialBuggle();
SerialBuggle sam = new SerialBuggle();
SerialBuggle sarah = new SerialBuggle();
sally.id // evaluates to 1
sam.id // evaluates to 2
sarah.id // evaluates to 3
```

How can we define SerialBuggle?

```
public class SerialBuggle extends Buggle {



}
```

Summary of Class Declarations

There are **five** kinds of declarations in a Java class:

- o **two** kinds of method declarations:
  - Instance variables
  - Class variables
- o **three** kinds of method declarations:
  - Instance methods
  - Class methods
  - Constructor methods