CS 213 : Software Methodology
Spring 2023

Sesh Venugopal

Jan 23

OOP – Constructors/Inheritance

Default Constructor, Multiple Constructors

Default Constructor

Given this definition of a Point class:

```
public class Point {
   int x,y;
   public Point(int x, int y) {
      this.x = x; this.y = y;
   }
}
```

Will this statement compile:

```
Point p = new Point();
```

NO. There isn't a matching no-arg constructor in `Point`.
Default constructor is written in by the compiler
ONLY when there is *no* programmer defined constructor!!

Default and no-arg constructors

A no-arg constructor is a constructor that does not take any arguments

The default constructor is a no-arg constructor that is *written in by the compiler*

A no-arg constructor can be written *explicitly by the programmer*, in which case it is not a default constructor.

Multiple constructors and `this()`

```
public class Point {
    int x,y;
    public Point(int x, int y) {
        this.x = x; this.y = y;
    }
    public Point(int x) {
        this(x,0);
    }
    public Point() {
        this(0,0);
    }
}
```

What do these statements do?

They call another matching (in argument sequence/types) constructor in the class – in this case the first constructor

Multiple constructors and `this()`

```
public class Point {

   public static final int X_MAX=800, Y_MAX=800;
   int x,y;

   // most general constructor with params for all fields (x and y)
   public Point(int x, int y) {
      if (x < 0 || x > X_MAX || y < 0 || y > Y_MAX) {
          throw new IllegalArgumentException("invalid x or y");
      }
      this.x = x; this.y = y;
   }
   public Point(int x) {
      this(x,0);
   }
   public Point(int y) {
      this(0,y);
   }
   public Point() {
      this(0,0);
   }
}
```

This code won't compile!

Duplicate constructor Point(int)

Can either allow default for x or default for y but not both

Inheritance and Constructors

Inheritance, Superclass and Subclass

```
public class Point {
   int x,y;
}
```

$$\begin{tikzcd}\text{superclass Point} \\ \text{subclass ColoredPoint} \arrow[u]\end{tikzcd}$$

```
public class ColoredPoint
extends Point {
   int x,y;
}
```

subclass `ColoredPoint` inherits `x` and `y` from superclass `Point`

What this means is `x` and `y` are fields in `ColoredPoint`, without the programmer having to write them in (CODE REUSE)

```
  Point p = new Point(); // OK, x and y in instance p are zero

  ColoredPoint cp =          // OK, x and y in instance cp are zero
     new ColoredPoint();
```

Inheritance and super/sub constructors

```
public class Point {
   int x,y;
   public Point(int x, int y) {
      this.x = x; this.y = y;
   }
}
Point p = new Point(3,4); // OK, p is (3,4)
```

```
public class ColoredPoint
extends Point {

}
```

Will this class compile? NO

Eclipse gives the following error message:

"Implicit super constructor Point() is undefined for default constructor. Must define an explicit constructor."

Inheritance – Subclass constructor

```
public class ColoredPoint
extends Point {
   int x,y;
   public ColoredPoint() {
      super();
   }
}
```

Default constructor

Calls superclass's constructor

The FIRST statement in a subclass constructor should invoke a superclass constructor. (Or it should invoke another constructor in the class, with `this(…)`).

A default constructor will ALWAYS CALL the superclass no-arg constructor

Problem: the `Point` class does not have a no-arg constructor!

Inheritance – Subclass constructor

```
public class ColoredPoint
extends Point {
    int x,y;
    public ColoredPoint() {
        super();
    }
}
```

"Implicit super constructor Point() is undefined for default constructor. Must define an explicit constructor."

The FIRST statement in a subclass constructor - ANY constructor, not just the default - should invoke a superclass constructor. (Or it should invoke another constructor in the class, with `this(…)`).

Inheritance – Subclass constructor

```
public class ColoredPoint
extends Point {
   int x,y;
   String color;
   public ColoredPoint(int x, int y, String color) {
      super(x,y);
      this.color = color;
   }
```

We will need to write a constructor that calls superclass's constructor with arguments

Will the following alternative compile? NO

```
public ColoredPoint(int x, int y, String color) {
      super();
      this.x = x; this.y = y;
      this.color = color;
   }
}
```

Compiler will write in super() as the FIRST statement, but Point does not have a no-arg constructor

Inheritance – Why call super(…)?

Why does the compiler throw in a `super()` call?

There’s nothing wrong with this:

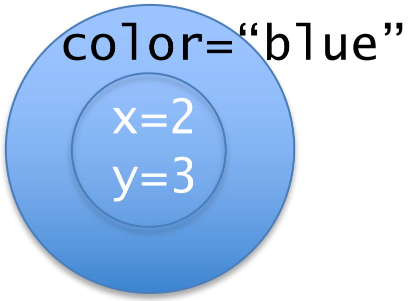
```
public ColoredPoint(int x, int y, String color) {
      this.x = x; this.y = y;
      this.color = color;
   }
}
```

But design-wise it’s not a good approach

Inheritance – Why call super(…)?

Think of a subclass instance having two parts: the superclass part (inherited), and the additional subclass part

```
ColoredPoint cp =
   new ColoredPoint(
             2,3,"blue");
```



Initialization of the superclass part is best done by a superclass constructor, no point in reinventing the wheel (Code REUSE)

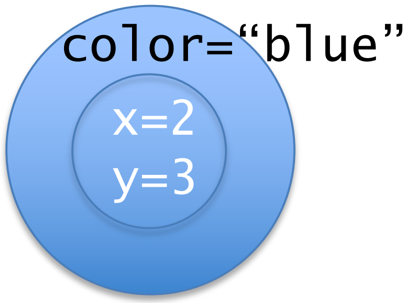
Thus the call to the superclass constructor, to FIRST initialize the superclass part:

```
super(x,y);
```

then code to initialize the subclass part:

```
this.color = color;
```

Inheritance – Why call super(…)?



Q. When a `ColoredPoint` instance is created, is an inner `Point` instance created as well?

NO.
It's CODE reuse,
not instance reuse