CS 213 : Software Methodology

Spring 2016

Sesh Venugopal

Lecture 1: Jan 19
OOP - Inheritance

Constructor

A constructor creates an object. True or False?

FALSE. A constructor initializes an object.

In the statement `new X()`, the `new X` part creates an X object, and the `X()` part calls the no-arg constructor of X on behalf of the new object, to initialize it

When an object is created with new, its fields are initialized to their intrinsic default values (zero for int, null for object references, etc.). True or False?

TRUE.

Constructor

```
public class Point { }
```

Will this class definition compile? Yes or No.

YES

How to create a new instance of `Point`?

```
new Point();
```

Really? But there's no constructor in the `Point` class ☹

Actually there is. The compiler throws in a default constructor that looks like this:

```
public class Point {
    public Point() { }
}
```

No arguments to constructor, nothing in the body

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;
   }
}
```

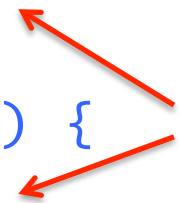
Will this statement compile:

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

NO. There isn't a matching constructor in `Point`.
(Default constructor is thrown in ONLY when there is no defined constructor.)

Constructor

```
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);
    }
}
```



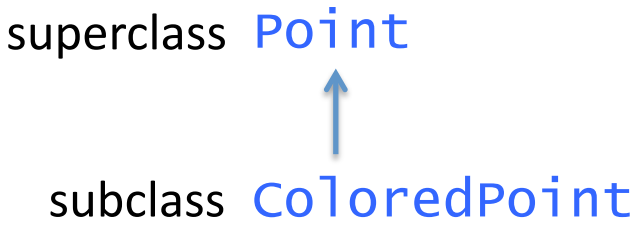
What do these statements do?

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

Inheritance

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

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

Inheritance

```
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 NOT COMPILE

"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();
   }
}
```

Calls superclass's constructor

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

A default constructor will always call a superclass constructor with no arguments (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;
    String color;
    public ColoredPoint(int x, int y, String color) {
        super(x,y);
        this.color = color;
    }
}
```

Calls superclass's constructor

Will the following alternative compile?

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

NO.

Same error as before, a super() call is introduced as the FIRST statement, but Point does not have a no-arg constructor