14. Inheritance :: Calling a Superclass Constructor

Who has the responsibility for initializing instance variables when an object is being constructed? The constructors, and in particular, the specific constructor that is called:

```
Point pete = new Point();            // Calls Point()
Point patty = new Point(2.3, 4.5);  // Calls Point(double, double)
```

Consider our *Point* class:

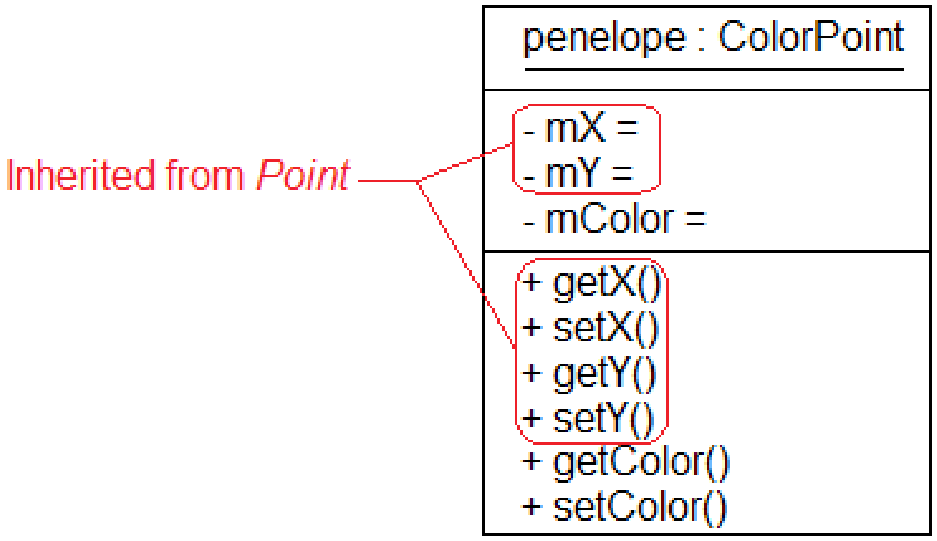
```
public class Point {
  private double mX;
  private double mY;
  public Point() { this(0, 0); }
  public Point(double pX, double pY) { setX(pX); setY(pY); }
  public double getX() { return mX; }
  public double getY() { return mY; }
  public void setX(double pNewX) { mX = pNewX; }
  public void setY(double pNewY) { mY = pNewY; }
}
```

14. Inheritance :: Calling a Superclass Constructor (continued)

Suppose we create a subclass of *Point* named *ColorPoint* which adds a *Color* instance variable named *mColor*. Now, who has responsibility for initializing the instance variables of a *ColorPoint* object when it is being constructed?

```
ColorPoint penelope = new ColorPoint(10, 20, Color.BLUE);
```

Here is an object diagram for *penelope*:



Note that some of the instance variables are inherited from *Point*. Furthermore, note that *Point* already has two constructors to initialize a *Point*: *Point*() and *Point*(**double**, **double**). So who should initialize the *mX* and *mY* instance variables of *penelope*? Why, naturally, *Point*! And who should initialize the *mColor* instance variable of *penelope*? Since *Point* knows nothing about colors, the responsibility for initializing *mColor* falls to *ColorPoint*.

14. Inheritance :: Calling a Superclass Constructor (continued)

We can summarize this discussion with these rules:

1. The responsibility for initializing **inherited instance variables** belongs to the **superclass ctors**.
2. The responsibility for initializing **subclass-declared instance variables** belongs to the **subclass ctors**.

Per 1, the *ColorPoint* constructor clearly has to have a way to call the superclass *Point* constructors. This is done by using the **super** reserved word:

```
public ColorPoint(double pX, double pY, Color pColor) {
  super(pX, pY);     // Calls Point(double, double) to init mX and mY.
  setColor(pColor);  // ColorPoint initializes the mColor instance variable.
}
```

Note that if a superclass ctor is called via **super**() that statement **must** be the first statement in the subclass ctor:

```
public ColorPoint(double pX, double pY, Color pColor) {
  setColor(pColor);  // ColorPoint initializes the mColor instance variable.
  super(pX, pY);     // Syntax error! super(pX, pY) must be first statement.
}
```

14. Inheritance :: Calling Subclass and Superclass Methods Summary

1. To call a superclass constructor from a subclass constructor:

```
public void subclassCtor(T t, U u) {
  super(parameters);  // Initialize inherited instance variables.
  // Now, initialize subclass instance variables.
}
```

2. To call a nonoverloaded and nonoverridden superclass method from a subclass method (the methods will not have the same names):

```
public void someSubclassMethod(T t, U u) {
  someSuperClassMethod(parameters);
}
```

14. Inheritance :: Calling Subclass and Superclass Methods Summary

3. To call an overloaded superclass method from a subclass method (the superclass and subclass methods will have the same name but different signatures). Note: this is not a common operation.

```
public void overloadedMethod(T t, U u) {
  overloadedMethod(different-parameter-data-types-than-T-and-U);
}
```

4. To call an overridden superclass method from a subclass method (the superclass and subclass methods will have the same names and signatures):

```
public void overriddenMethod(T t, U u) {
  super.overriddenMethod(t, u);
  doSomeSubclassSpecificOperationHere();
}
```