Inheritance and Constructors

Remember: classes are abstractions. To get down to business with your classes and really do some work, you're going to need to instantiate some objects. Let's talk about how inheritance and constructors work with subclasses and superclasses, and how we can use them to make things happen in our code.

Subclasses inherit all the private instance variables in a superclass that they extend, but they cannot directly access them since they are private. Constructors, on the other hand, are not inherited at all, although we can use them by making the appropriate calls.

So how do you initialize inherited private variables if you don't have direct access to them in the subclass? In Java you can put a call to the parent constructor as the first line in a subclass constructor. In Java, the superclass constructor can be called from the first line of a subclass constructor by using the keyword `super` and passing appropriate parameters, for example `super();` or `super(variable);` as shown in the code below. The actual parameters given to `super()` are used to initialize the inherited instance variables, for example the `name` instance variable in a `Person` superclass.

```
public class Employee extends Person {
public Employee() {
super(); // calls the Person() constructor
}
public Employee(String theName) {
super(theName); // calls Person(theName) constructor
}
}
```

#Coding Exercise

The `super(theName)` in the `Employee` constructor will call the constructor that takes a `String` object in the `Person` class to set the name.

Use the getter methods to print each attribute out on a new line.

Run

Employee.java Person.java

```
public class Employee extends Person {

   private int id;

   public Employee(String theName, int id) {
      super(theName);
      this.id = id;
   }

   public int getId() {
      return id;
   }

   public static void main(String[] args) {

      Employee emp = new Employee("Dani", 10);

      System.out.println(emp.getName());
      System.out.println(emp.getId());
   }
}
class Person {
   private String name;

   public Person(String theName) {
      this.name = theName;
   }

   public String getName() {
      return name;
   }

   public boolean setName(String theNewName) {
      if (theNewName != null) {
         this.name = theNewName;
         return true;
      }
      return false;
   }
}
```

In the `Employee` class code above, which class constructor sets the name? Which class constructor sets the `id`? How do you know?
In the Employee class, there is a constructor Employee(String theName, int id) which takes two parameters, a String for the name and an int for the id. This constructor calls super(theName) which means it calls the constructor of the parent class (Person in this case) to set the name. It also sets the id attribute using this.id = id;.

If a class has no constructor in Java, the compiler will add a no-argument constructor automatically. A no-argument constructor is one that doesn't have any parameters, for example `public Person()`.

If a subclass has no call to a superclass constructor using `super` as the first line in a subclass constructor, the compiler will automatically add a `super()` call as the first line in its constructor. Be sure to provide no-argument constructors in parent classes or use an explicit call to `super()` as the first line in the constructors of subclasses.

Regardless of whether the superclass constructor is called implicitly or explicitly, the process of calling superclass constructors continues through all of the parent classes until the Object constructor is called. Again, this happens because every class inherits from the Object class, and it's the last stop in terms of any object's ancestor tree.

InfoWarningTip
If you use a `super()` call, but it isn't the first line of the constructor, you'll get an error. Pay attention to these errors and learn from them. Working in an IDE like Eclipse is a great way to learn about the quirks of a language, and you should get used to reading any and all messages that result from minor missteps such as these. You can always look them up online, too. We recommend reading the official documentation whenever possible.

#Check your understanding

Given the class definitions of `MPoint` and `NamedPoint` below, which of the constructors that follow (labeled I, II, and III) would be valid in the `NamedPoint` class?

```
class MPoint {
private int myX; // coordinates
private int myY;

public MPoint( ) {
myX = 0;
myY = 0;
}

public MPoint(int a, int b) {
myX = a;
myY = b;
}

// ... other methods not shown

}

public class NamedPoint extends MPoint {
private String myName;
// constructors go here
// ... other methods not shown
}

// Proposed constructors for this class:
I. public NamedPoint() {
myName = "";
}
II. public NamedPoint(int d1, int d2, String name) {
myX = d1;
myY = d2;
myName = name;
}
III. public NamedPoint(int d1, int d2, String name) {
super(d1, d2);
myName = name;
}
```

You can step through this code using the Java Visualizer by clicking the following link: Named Point.

#Knowledge Check

Fill In The Blank

Fill In The Blank. Hint: be careful to write this hyphenated word properly.

#Summary

- Subclasses inherit all the private instance variables in a superclass that they extend, but they cannot directly access them since they are private.
- Constructors are not inherited.
- The superclass constructor can be called from the first line of a subclass constructor by using the keyword `super` and passing appropriate parameters to set the private instance variables of the superclass.
- The actual parameters passed in the call to the superclass constructor provide values that the constructor can use to initialize the object's instance variables.
- When a subclass's constructor does not explicitly call a superclass's constructor using `super`, Java inserts a call to the superclass's no-argument constructor.
- Regardless of whether the superclass constructor is called implicitly or explicitly, the process of calling superclass constructors continues until the `Object` constructor is called. At this point, all of the constructors within the hierarchy execute beginning with the Object constructor.