Intermediate OOP

Stefan Zorcic

In this tutorial we will cover inheritance, polymorphism, abstract classes and interfaces.

Inheritance is the process in which class inherit attributes and behaviours from another class. The class that is inheriting the attributes and behaviours is called the subclass while the class to who the behaviours and attributes belong is called the superclass. This relationship can be thought of as a parent/child relationship where the child(subclass) can gain attributes such as blue eyes from the parent(superclass).

The superclass subclass relationship uses the keyword **extends** to indicate inheritance.

Super-class:

```
public class Parent{
   String eyeColor = "Blue";

   public void eat(){
      /* implementation not shown */
   }
}
```

Sub-class:

```
public class Child extends Parent{
}
```

Upon creating an instance of the subclass it becomes evident we can access behaviours and attributes from the superclass. In this case after creating an instance of the Child class we can call the eyeColor variable from the Parent class.

Code:

```
public static void main(String [] args){
   Child bob = new Child();
   System.out.println(bob.eyeColor);
}
```

Output:

```
Blue
```

Polymorphism is an integral part of more advanced OOP projects. Polymorphism expands the concept of inheritance to multiple classes. Polymorphism allows for one superclass to have multiple subclasses to truly form a class tree. An example of a use of polymorphism can be for animals classes. We can create a class with behaviours common to all animals (such as breathing and eating), then create individual subclasses for each animal for specific behaviours. Below is an implementation of what is described. **Super** calls the superclass constructor and must be in the first line of subclass constructor.

Superclass:

```
public class Animal{
   int energyLevel;

   public Animal(int energy){
      energyLevel=energy;
   }

   public void eat(){
      energyLevel+=10;
   }

   public void breath(){
      System.out.println("AIR!");
   }
}
```

Subclass #1:

```
 public class Robin extends Animal{
    int altitude;

    public Robin(int energy, int alt){
       super(energy);
       altitude=alt;
    }

    public void fly(){
       altitude+=50;
    }
 }
```

Subclass #2:

```
public class Pig(){
   public Pig(int energy){
      super(energy);
   }

   public void oink(){
      System.out.println("OINK!!");
   }
}
```

Abstraction is another important part of OOP that is used for programmers to only give essential pieces of information. Abstraction in Java OOP can be done by using the abstract non-access modifier or the interface non-access modifier.

The abstract modifier can be used in place of Public when declaring a class, however abstract classes have the caveat of not being able to make an object and are only used for inheritance of a subclass. Abstract classes also can have abstract method which do not have a code body, the Cody body is to be implemented in the subclass. An example of an abstract class is below.

*Note, when extending an abstract class you must override all abstract methods, otherwise the code will not compile.

Abstract-class:

```
abstract class phone{
   String phoneName;

   public abstract void call();
}
```

Sub-class:

```
public class iPhone extends phone{

   @Override
   public void call(){
      System.out.println("Calling ...");
   }
}
```

Interface is another method of abstraction is Java OOP, interfaces do not have any code bodies in their method and like abstract classes must be used through a subclass. However interfaces use the keyword **implements** instead of **extends**, **implements** has the same function as **extends** and have very small differences in practice.

*Note, when extending an interface you must override all methods, otherwise the code will not compile.

Interface:

```
interface phone{
   public void call();
   public void charge();
}
```

Sub-class:

```
public class iPhone implements phone{
   int batteryPercent=50;

   @Override
   public void call(){
      System.out.println("Calling ...");
   }

   @Override
   public void charge(){
      batteryPercent+=10;
   }
}
```