Object-Oriented Java

Basic Java Object-Oriented Programming

Declaring a class in Java is done as follows:

```
public class Main {
  int x = 5;
}
```

And to declare an object, we do:

```
public class ExampleObj {
  int x = 5;

public static void main(String[] args) {
    ExampleObj myObj = new ExampleObj();
    System.out.println(myObj.x);
  }
}
```

As you can see we've also creted an attribute in the above program.

To modify an attribute we do the following:

```
public class Main {
  int x = 10;

  public static void main(String[] args) {
    Main myObj = new Main();
    myObj.x = 40;
    System.out.println(myObj.x);
  }
}
```

If we change the declaration of "`int x = 10;`" to "`final int x = 10;`", the code to change X to 40 will produce an error.

Java Constructors

A constructor in Java is a special method that is used to initialize objects. The constructor name must match the class name, and it cannot have a return type (like `void`). Also note that the constructor is called when the object is created. All classes have constructors by default: if you do not create a class constructor yourself, Java creates one for you.

```
public class ExampleObj {
  int x;
  public ExampleObj(int y) {
    x = y;
  }

  public static void main(String[] args) {
    ExampleObj myObj = new ExampleObj(5);
    System.out.println(myObj.x);
  }
}
```

The constructor sets x to y, and in this case we pass a parameter to the constructor (5).

Object-Oriented Java

Java Interitance

Let's look an example, the Car class (subclass) inherits the attributes and methods from the Vehicle class (superclass):

```
class Vehicle {
  protected String brand = "Ford"; // Vehicle attribute
  public void honk() {             // Vehicle method
    System.out.println("Tuut, tuut!");
  }
}
```

```
class Car extends Vehicle {
  private String modelName = "Mustang";    // Car attrib
  public static void main(String[] args) {

    // Create a myCar object
    Car myCar = new Car();

    // Call the honk() method on the myCar object
    myCar.honk();

/* Display the value of the brand attribute (from Vehicle
class) and the value of the modelName from the Car class*/
    System.out.println(myCar.brand + myCar.modelName);
  }
}
```

To inherit from a class, use the extends keyword.

Java Polymorphism

For example, think of a superclass called Animal that has a method called animalSound(). Subclasses of Animals could be Pigs, Cats, Dogs, Birds - And they also have their own implementation of an animal sound (the pig oinks, and the cat meows, etc.):

```
class Animal {
  public void animalSound() {
    System.out.println("The animal makes a sound");
  }
}
```

```
class Pig extends Animal {
  public void animalSound() {
    System.out.println("The pig says: wee wee");
  }
}
```

```
class Dog extends Animal {
  public void animalSound() {
    System.out.println("The dog says: bow wow");
  }
}
```

Instances of `Dog` that call the method `animalSound()` will give a different anwswer than instances of `Pig` that call the method `animalSound()`.