Java Constructors

A constructor in Java is a **special method** that is used to initialize objects. The constructor is called when an object of a class is created. It can be used to set initial values for object attributes:

```
// Create a MyClass class

public class MyClass {

  int x;  // Create a class attribute


  // Create a class constructor for the MyClass class

  public MyClass() {

    x = 5;  // Set the initial value for the class attribute x

  }


  public static void main(String[] args) {

    MyClass myObj = new MyClass(); // Create an object of class MyClass
(This will call the constructor)

    System.out.println(myObj.x); // Print the value of x

  }

}
```

Constructor Parameters

Constructors can also take parameters, which is used to initialize attributes.

The following example adds an `int y` parameter to the constructor. Inside the constructor we set x to y (x=y). When we call the constructor, we pass a parameter to the constructor (5), which will set the value of x to 5:

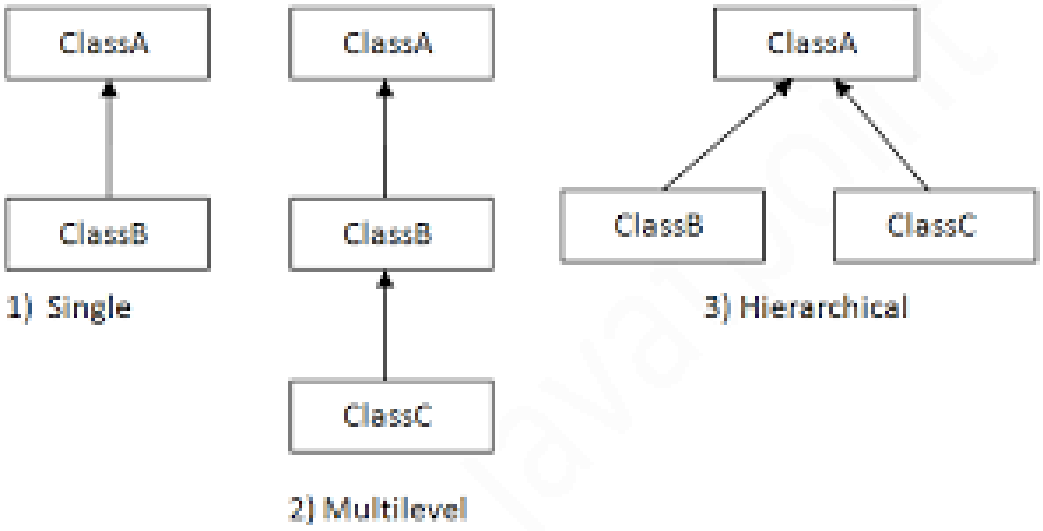
```
public class MyClass {
  int x;

  public MyClass(int y) {
    x = y;
  }

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

// Outputs 5
```

Java Inheritance (Subclass and Superclass)

Java Inheritance (Subclass and Superclass)

In Java, it is possible to inherit attributes and methods from one class to another. We group the "inheritance concept" into two categories:

- **subclass** (child) - the class that inherits from another class
- **superclass** (parent) - the class being inherited from



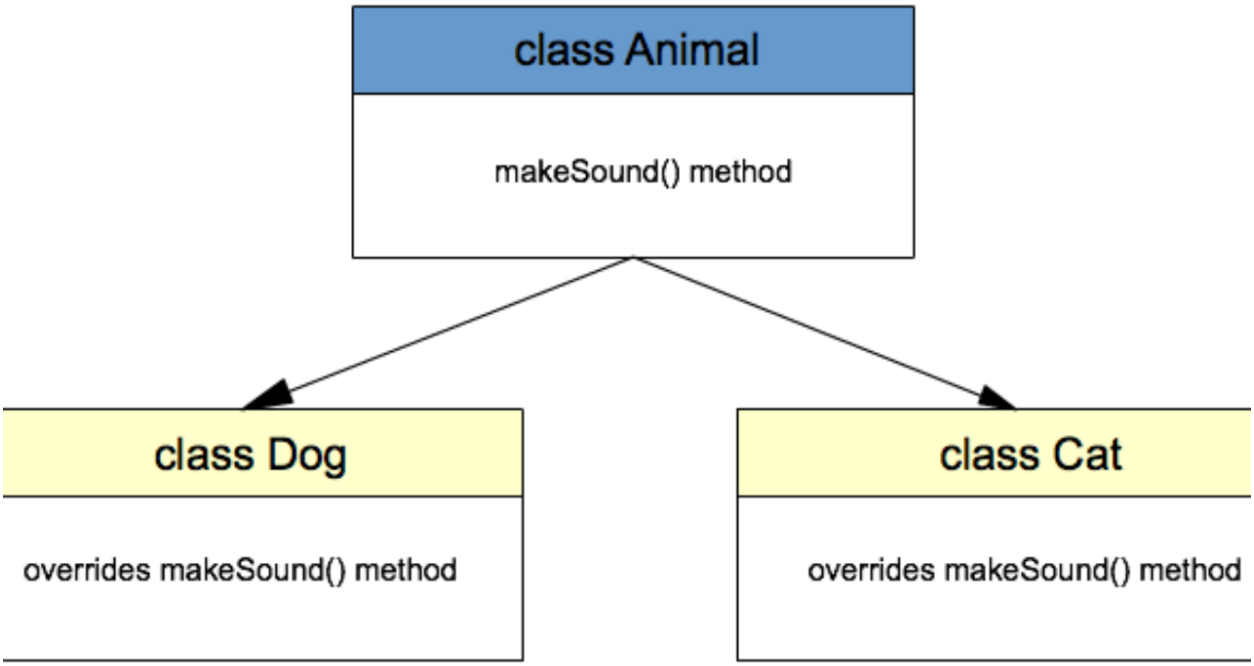
To inherit from a class, use the `extends` keyword.

```
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 attribute
  public static void main(String[] args) {

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

    // Call the honk() method (from the Vehicle class) on the myCar object
```

```
    myCar.honk();


    // Display the value of the brand attribute (from the Vehicle class)
and the value of the modelName from the Car class

    System.out.println(myCar.brand + " " + myCar.modelName);

  }

}




final class Vehicle {

  ...

}


class Car extends Vehicle {

  ...

}
```

Java Polymorphism

Polymorphism means "many forms", and it occurs when we have many classes that are related to each other by inheritance.



Like we specified in the previous chapter; **Inheritance** lets us inherit attributes and methods from another class. **Polymorphism** uses those methods to perform different tasks. This allows us to perform a single action in different ways.

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");
  }
}

class MyMainClass {
  public static void main(String[] args) {
    Animal myAnimal = new Animal();  // Create a Animal object
    Animal myPig = new Pig();  // Create a Pig object
    Animal myDog = new Dog();  // Create a Dog object
    myAnimal.animalSound();
    myPig.animalSound();
    myDog.animalSound();
  }
}
```