Java Inheritance example

```
class A {

//Members and methods declarations.

}

class B extends A {

//Members and methods from A are inherited.

//Members and methods declarations of B.

}
```

The most fundamental element of Java is the class. A class represents an entity and also, defines and implements its functionality. In Java, classes can be **derived** from other classes, in order to create more complex relationships.

A class that is derived from another class is called *subclass* and inherits all fields and methods of its *superclass*. In Java, only single inheritance is allowed and thus, every class can have at most one direct superclass. A class can be derived from another class that is derived from another class and so on. Finally, we must mention that each class in Java is implicitly a subclass of the `Object` class.

Suppose we have declared and implemented a class *A*. In order to declare a class *B* that is derived from *A*, Java offers the *extend* keyword that is used as shown below:

```
class A {
//Members and methods declarations.
}

class B extends A {
//Members and methods from A are inherited.
//Members and methods declarations of B.
}
```

Java supports only public inheritance and thus, all fields and methods of the superclass are inherited and can be used by the subclass. **The only exception** are the **private members** of the superclass that **cannot be accessed directly** from the subclass. Also, constructors are not members, thus they are not inherited by subclasses, but the constructor of the superclass can be invoked from the subclass. In order to call the constructor of the superclass Java provides the keyword super, as shown below:

```
class A {
   public A() {
      System.out.println("New A");
   }
}

class B extends A {
   public B() {
      super();
      System.out.println("New B");
   }
}
```

o/p:
?

A sample example to present the inheritance in Java is shown below:

Animal.java:

```
public class Animal {
public Animal ()
{
System.out.println("A new animal has been created!");
}
public void sleep() {
System.out.println("An animal  sleeps...")
```

```
                        }

public void eat() {
System.out.println("An animal eats...");
                        }
}
```

Bird.java:

```
public class Bird extends Animal {

                public Bird() {
                        super();
                System.out.println("A new bird has been created!");
                }

        @Override
                public void sleep() {
                                System.out.println("A bird sleeps...");
                                        }

        @Override
                public void eat() {
                                System.out.println("A bird eats...");
                                }

                                        }
```

Dog.java:

```
public class Dog extends Animal {
   public Dog() {
      super();
      System.out.println("A new dog has been created!");
   }
   @Override
   public void sleep() {
      System.out.println("A dog sleeps...");
```

```
    }

    @Override
    public void eat() {
        System.out.println("A dog eats...");
    }
}
```

MainClass.java:

```
public class MainClass {
    public static void main(String[] args) {
        Animal animal = new Animal();
        Bird bird = new Bird();
        Dog dog = new Dog();

        System.out.println();

        animal.sleep();
        animal.eat();

        bird.sleep();
        bird.eat();

        dog.sleep();
        dog.eat();
    }
}
```

o/p

```
A new animal has been created!
A new animal has been created!
A new bird has been created!
```

```
A new animal has been created!
A new dog has been created!

An animal sleeps...
An animal eats...
A bird sleeps...
A bird eats...
A dog sleeps...
A dog eats
```

In this example we created three distinct classes, `Animal`, `Dog` and `Bird`. Both `Dog` and `Bird` classes extend the `Animal` class and thus, they inherit its members and methods. Moreover, as we can see below, each class overrides the methods of `Animal` and thus, both the Dog and Bird classes redefine the functionality of `Animal's` methods.