**Inheritance:**

**In object-oriented programming**, **inheritance** enables new objects to take on the properties of existing objects. A class that is used as the basis for **inheritance** is called a superclass or base class. A class that **inherits** from a superclass is called a subclass or derived class.

To inherit from a class, use the “:” symbol. In the example below, the Car class (child) inherits the attributes and methods from the Vehicle class (parent):

|  |
| --- |
| Code Example |
| package javaapplication4;  class Vehicle  {  protected String brand = "Ford";  String get\_brand()  {  return this.brand;  }  }  class Car extends Vehicle  {  private String model = "Mustang";  String get\_model()  {  return this.model;  }  }  public class JavaApplication4  {  public static void main(String[] args)  {  Car myCar=new Car();  System.out.println(myCar.get\_brand());  System.out.println(myCar.get\_model());  }  } |

A class can also be derived from one class, which is already derived from another class. In the following example, MyGrandChild is derived from class MyChild (which is derived from MyClass).

|  |
| --- |
| package inheritance\_in\_java;  class Myclass  {  void myFunction()  {  System.out.println("Some content in parent class.");  }  }  class MyChild extends Myclass  {  }  class MyGrandChild extends MyChild  {  }  public class Inheritance\_in\_Java  {  public static void main(String[] args)  {  MyGrandChild myObj= new MyGrandChild();  myObj.myFunction();  }  } |

**Member Access and Inheritance**

Although a subclass includes all of the members of its superclass, **it cannot access those members of the superclass that have been declared as *private***. A class member that has been declared as private will remain private to its class. **It is not accessible by any code outside its class, including subclasses.** **To avoid this we use protectedaccess specifier. *Protected* enforces t**he type or member **can only be accessed by code in the same class, or in a derived class.**

**Methods Overloading:**

If a class has multiple methods having same name but different in parameters, it is known as **Method Overloading**. Method overloading allows programmers to write two or more than two methods with the same name, but different parameter lists.

|  |
| --- |
| class Adder  {  int add(int a,int b)  {  return a+b;  }  int add(int a,int b,int c)  {  return a+b+c;  }  }  public class JavaApplication4  {  public static void main(String[] args)  {  Adder a1=new Adder();  System.out.println(a1.add(20, 30));  System.out.println(a1.add(20,10, 30));  }  } |