|  |
| --- |
| **class Base {**  **public void Print() {**  **System.out.println("Base");**  **}**  **}**  **class Derived extends Base {**  **public void Print() {**  **System.out.println("Derived");**  **}**  **}**  **class Main{**  **public static void DoPrint( Base o ) {**  **o.Print();**  **}**  **public static void main(String[] args) {**  **Base x = new Base();**  **Base y = new Derived();**  **Derived z = new Derived();**  **DoPrint(x); // 1**  **DoPrint(y); // 2**  **DoPrint(z); //3**  **}**  **}**  **Explanation:** **1.** **DoPrint(x); // call of Doprint() method with base class object so base class print() method will be invoked : Will print : Base**  **2.** **DoPrint(y); // here method DoPrint() is called using “y” which is base class object but instantiated using derived class so y is referring to derived class and hence print() method of Derived class will be called: Will print : Derived**  **3.** **DoPrint(z); // here method DoPrint() is called using “z which is derived class object so z is referring to derived class and hence print() method of Derived class will be called: Will print : Derived** |
| **class Point {**  **protected int x, y;**  **public Point(int \_x, int \_y) {**  **x = \_x;**  **y = \_y;**  **}**  **}**  **public class Main {**  **public static void main(String args[]) {**  **Point p = new Point();**  **System.out.println("x = " + p.x + ", y = " + p.y);**  **}**  **}**  **Explanation:** **Point p = new Point(); // *Compile time error* in calling constructor Point() because Point class has only one parameterized constructor *Point p = new Point();* and we are not passing the parameters.**  **If a class has only parameterized constructor then compiler cannot call its own default constructor. So we are forcing the user to pass two parameters.**  **Please Note: I told in the class that protected members of a class can’t be used in other classes of same package but I checked I was wrong.**  **Protected members of one class can be used in all other classes belong to same package.**  **So there is no error as such in this code.** |
| **class Base**  **{**  **protected void getDetails()**  **{**  **System.out.println("Base class");**  **}**  **}**  **public class Test1 extends Base**  **{**  **public void getDetails()**  **{**  **System.out.println("Test class");**  **}**  **public static void main(String[] args)**  **{**  **Base obj = new Test1();**  **obj.getDetails();**  **}}**  **Explanation: this example is of Method overriding in Base and Derived class we have a method void getDetails() but the access specifier of base class is protected void getDetails() and access specifier of derived class is public void getDetails().**  **And this is allowed in java so no error.**  **Output will be : Test class** |
| **class Alpha**  **{**  **private void flip()**  **{**  **System.out.println("Alpha");**  **}**  **}**    **public class Beta extends Alpha**  **{**  **public void flip()**  **{**  **System.out.println("Beta");**  **}**  **public static void main(String[] args)**  **{**  **Alpha al = new Beta();**  **al.flip();**  **}**  **}**  **Explanation: Compile time error : private void flip() // Method is private in Base class so not accessible to derived class. It can be protected or public or default.** |
| **public class Test**  **{ public int getData() //getdata() 1**  **{ return 0;**  **}**  **public long getData() //getdata 2**  **{ return 1;**  **}**  **public static void main(String[] args)**  **{**  **Test obj = new Test();**  **System.out.println(obj.getData());**  **}}**  **Explanation: Compile time error: This is an example of Method overloading two methods can have same name but they should have different number or type of receiving parameters. Here both getdata() have same receiving type** |
| **class Grandparent**  **{**  **public void Print()**  **{**  **System.out.println("Grandparent's Print()");**  **}**  **}**    **class Parent extends Grandparent**  **{**  **public void Print()**  **{**  **System.out.println("Parent's Print()");**  **}**  **}**    **class Child extends Parent**  **{**  **public void Print()**  **{**  **super.super.Print(); // Compile time error**  **System.out.println("Child's Print()");**  **}**  **}**    **public class Main**  **{**  **public static void main(String[] args)**  **{**  **Child c = new Child();**  **c.Print();**  **}}**  **Explanation: Compile time error : in super.super.Print();**  **If we want to call grandparent class method then we have to go through the parent class method.**  **So call to Print() of GrandParent class should be placed in Parent class.**  Print() //GrandParent class  {  }  Print() // Parent class  { super.print() // call to Grandparent class  }      Print() // Child class  { super.Print() // call to Parent class  } |
| **public class Test**  **{**  **public int getData(String temp)**  **{**  **return 0;**  **}**  **public int getData(String temp)**  **{**  **return 1;**  **}**  **public static void main(String[] args)**  **{**  **Test obj = new Test();**  **System.out.println(obj.getData("GFG"));**  **}**  **}**  **Explanation: Compile time error: This is an example of Method overloading two methods can have same name but they should have different number or type of receiving parameters. Here both getdata() have same receiving type.**  **Both the method are receiving String as parameter getData(String temp)** |
| **public class Test**  **{**  **private String function()**  **{ return ("GFG");**  **}**  **public final static String function(int data)**  **{ return ("GeeksforGeeks");**  **}**  **public static void main(String[] args)**  **{**  **Test obj = new Test();**  **System.out.println(obj.function());**  **}}**  **Explanation: Output : GFG**  **Because in the call to System.out.println(obj.function());   we are not passing any parameter . So matching function is**  **private String function()**  **{ return ("GFG");**  **}**  **Another point to mention is that we may have static method with the same name in a class. final keyword is to deny overriding of this method.**  **public final static String function(int data)**  **{ return ("GeeksforGeeks");**  **}** |