1)

// Main.java

public class Main

{

    public static void gfg(String s)

    {

        System.out.println("String");

    }

    public static void gfg(Object o)

    {

        System.out.println("Object");

    }

    public static void main(String args[])

    {

        gfg(null);

    }

} //end class

**Output**:

String

**Explanation** : In case of [method overloading](https://www.geeksforgeeks.org/overloading-in-java/), the most specific method is chosen at compile time. As ‘java.lang.String’ is a more specific type than ‘java.lang.Object’. In this case the method which takes ‘String’ as a parameter is choosen.

2)

// Main.java

public class Main

{

    public static void gfg(String s)

    {

        System.out.println("String");

    }

    public static void gfg(Object o)

    {

        System.out.println("Object");

    }

    public static void gfg(Integer i)

    {

        System.out.println("Integer");

    }

    public static void main(String args[])

    {

        gfg(null);

    }

} //end class

**Output:**

Compile Error at line 19.

**Explanation:** In this case of [method Overloading](https://www.geeksforgeeks.org/overloading-in-java/), the most specific method is choosen at compile time.  
As ‘java.lang.String’ and ‘java.lang.Integer’ is a more specific type than ‘java.lang.Object’,but between ‘java.lang.String’ and ‘java.lang.Integer’ none is more specific.  
In this case the Java is unable to decide which method to call.

|  |
| --- |
| **3)Important**  // Main.java  public class Main  {      public static void main(String args[])      {          String s1 = "abc";          String s2 = s1;          s1 += "d";          System.out.println(s1 + " " + s2 + " " + (s1 == s2));            StringBuffer sb1 = new StringBuffer("abc");          StringBuffer sb2 = sb1;          sb1.append("d");          System.out.println(sb1 + " " + sb2 + " " + (sb1 == sb2));      }  } //end class |

Output:

abcd abc false

abcd abcd true

**Explanation :**In Java, String is immutable and string buffer is mutable.  
So string s2 and s1 both pointing to the same string abc. And, after making the changes the string s1 points to abcd and s2 points to abc, hence false. While in string buffer, both sb1 and sb2 both point to the same object. Since string buffer are mutable, making changes in one string also make changes to the other string. So both string still pointing to the same object after making the changes to the object (here sb2).

4)

|  |
| --- |
| class First  {      public First() {  System.out.println("a"); }  }    class Second extends First  {      public Second()  {  System.out.println("b"); }  }    class Third extends Second  {      public Third()   {  System.out.println("c"); }  }    public class MainClass  {      public static void main(String[] args)      {          Third c = new Third();      }  } |

Output:

a

b

c

**Explanation:**  
While creating a new object of ‘Third’ type, before calling the default constructor of Third class, the default constructor of super class is called i.e, Second class and then again before the default constructor of super class, default constructor of First class is called. And hence gives such output.

|  |
| --- |
| 5) class First  {      int i = 10;        public First(int j)      {          System.out.println(i);          this.i = j \* 10;      }  }    class Second extends First  {      public Second(int j)      {          super(j);          System.out.println(i);          this.i = j \* 20;      }  }    public class MainClass  {      public static void main(String[] args)      {          Second n = new Second(20);          System.out.println(n.i);      }  } |
|  |

Output:

10

200

400

**Explanation:**  
Since in ‘Second’ class it doesn’t have its own ‘i’, the variable is inherited from the super class. Also, the constructor of parent is called when we create an object of Second.

|  |
| --- |
| 6)  import java.util.\*;  class I  {      public static void main (String[] args)      {          Object i = new ArrayList().iterator();          System.out.print((i instanceof List) + ", ");          System.out.print((i instanceof Iterator) + ", ");          System.out.print(i instanceof ListIterator);      }  } |

Output:

false, true, false

**Explanation:**  
The iterator() method returns an iterator over the elements in the list in proper sequence, it doesn’t return a List or a ListIterator object. A ListIterator can be obtained by invoking the listIterator method.

7)

|  |
| --- |
| class ThreadEx extends Thread  {      public void run()      {          System.out.print("Hello...");      }      public static void main(String args[])      {          ThreadEx T1 = new ThreadEx();          T1.start();          T1.stop();          T1.start();      }  } |

Output:

Run Time Exception

**Explanation:**  
Exception in thread “main” java.lang.IllegalThreadStateException at java.lang.Thread.start  
Thread cannot be started twice.

|  |
| --- |
| 8)  public class Calculator  {      int num = 100;      public void calc(int num)  { this.num = num \* 10;  }      public void printNum()     { System.out.println(num); }        public static void main(String[] args)      {          Calculator obj = new Calculator();          obj.calc(2);          obj.printNum();      }  } |

**Answer : 20**  
**Explanation :** Here the class instance variable name(num) is same as calc() method local variable name(num). So for referencing class instance variable from calc() method, [**this**](http://quiz.geeksforgeeks.org/this-reference-in-java/) keyword is used. So in statement **this.num = num \* 10**, num represents local variable of the method whose value is 2 and this.num represents class instance variable whose initial value is 100. Now in printNum() method, as it has no local variable whose name is same as class instance variable, so we can directly use num to reference instance variable, although this.num can be used.