# Output of Java Program | Set 1

**Difficulty Level:** Rookie

Predict the output of following Java Programs.

**Program 1**

filter\_none

edit

play\_arrow

brightness\_4

|  |
| --- |
| // filename Main.java  class Test {      protected int x, y;  }    class Main {      public static void main(String args[]) {          Test t = new Test();          System.out.println(t.x + " " + t.y);      }  } |

Output

0 0

In Java, a protected member is accessible in all classes of same package and in inherited classes of other packages. Since Test and Main are in same package, no access related problem in the above program. Also, the default constructors initialize integral variables as 0 in Java (See [this](https://www.geeksforgeeks.org/g-fact-50/)GFact for more details). That is why we get output as 0 0.

**Program 2**

filter\_none

edit

play\_arrow

brightness\_4

|  |
| --- |
| // filename Test.java  class Test {      public static void main(String[] args) {          for(int i = 0; 1; i++) {              System.out.println("Hello");              break;          }      }  } |

Output: Compiler Error  
There is an error in condition check expression of for loop. Java differs from C++(or C) here. C++ considers all non-zero values as true and 0 as false. Unlike C++, an integer value expression cannot be placed where a boolean is expected in Java. Following is the corrected program.

// filename Test.java

class Test {

    public static void main(String[] args) {

        for(int i = 0; true; i++) {

            System.out.println("Hello");

            break;

        }

    }

}

// Output: Hello

**Program 3**

filter\_none

edit

play\_arrow

brightness\_4

|  |
| --- |
| // filename Main.java  class Main {      public static void main(String args[]) {          System.out.println(fun());      }      int fun() {          return 20;      }  } |

Output: Compiler Error  
Like C++, in Java, non-static methods cannot be called in a static method. If we make fun() static, then the program compiles fine without any compiler error. Following is the corrected program.

filter\_none

edit

play\_arrow

brightness\_4

|  |
| --- |
| // filename Main.java  class Main {      public static void main(String args[]) {          System.out.println(fun());      }      static int fun() {          return 20;      }  }  // Output: 20 |

**Program 4**

filter\_none

edit

play\_arrow

brightness\_4

|  |
| --- |
| // filename Test.java  class Test {     public static void main(String args[]) {         System.out.println(fun());     }     static int fun() {         static int x= 0;         return ++x;     }  } |

Output: Compiler Error  
Unlike C++, static local variables are not allowed in Java. See [this](https://www.geeksforgeeks.org/g-fact-47/)GFact for details. We can have class static members to count number of function calls and other purposes that C++ local static variables serve. Following is the corrected program.

filter\_none

edit

play\_arrow

brightness\_4

|  |
| --- |
| class Test {     private static int x;     public static void main(String args[]) {         System.out.println(fun());     }     static int fun() {         return ++x;     }  }  // Output: 1 |

Output of Java Program | Set 2

Predict the output of the following Java programs.  
**Question 1:**

filter\_none

edit

play\_arrow

brightness\_4

|  |
| --- |
| package main;    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);          DoPrint(y);          DoPrint(z);      }  } |

**Output:**

Base

Derived

Derived

Predicting the first line of output is easy. We create an object of type Base and call DoPrint(). DoPrint calls the print function and we get the first line.

DoPrint(y) causes the second line of output. Like C++, assigning a derived class reference to a base class reference is allowed in Java. Therefore, the expression Base y = new Derived() is a valid statement in Java. In DoPrint(), o starts referring to the same object as referred by y. Also, unlike C++, functions are virtual by default in Java. So, when we call o.print(), the print() method of Derived class is called due to run time polymorphism present by default in Java.  
DoPrint(z) causes the third line of output, we pass a reference of Derived type and the print() method of Derived class is called again. The point to note here is: unlike C++, [object slicing](http://en.wikipedia.org/wiki/Object_slicing) doesn’t happen in Java. Because non-primitive types are always assigned by reference.

**Question 2:**

filter\_none

edit

play\_arrow

brightness\_4

|  |
| --- |
| package main;    // filename Main.java  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);      }  } |

**Output:**

Compiler Error

In the above program, there are no access permission issues because the Point and Main are in the same package and protected members of a class can be accessed in other classes of the same package. The problem with the code is: *there is no default constructor in Point.*  
[Like C++](https://www.geeksforgeeks.org/does-c-compiler-create-default-constructor-when-we-write-our-own/), if we write our own parametrized constructor then Java compiler doesn’t create the default constructor. So there are following two changes to Point class that can fix the above program.

1. Remove the parameterized constructor.
2. Add a constructor without any parameter.

Output of Java Program | Set 3

Predict the output of following Java Programs:

filter\_none

edit

play\_arrow

brightness\_4

|  |
| --- |
| // filename: Test.java  class Test {      int x  = 10;      public static void main(String[] args) {           Test t = new Test();           System.out.println(t.x);      }  } |

The program works fine and prints 10. Unlike C++, in Java, members can initialized with declaration of class. This initialization works well when the initialization value is available and the initialization can be put on one line (See [this](http://docs.oracle.com/javase/tutorial/java/javaOO/initial.html)for more details). For example, the following program also works fine.

filter\_none

edit

play\_arrow

brightness\_4

|  |
| --- |
| // filename: Test.java  class Test {      int y = 2;      int x  = y+2;      public static void main(String[] args) {           Test m = new Test();           System.out.println("x = " + m.x + ", y = " + m.y);      }  } |

Output of the above program is “x = 4, y = 2”. y is initialized first, then x is initialized as y + 2. So the value of x becomes 4.

What happen when a member is initialized in class declaration and constructor both? Consider the following program.

|  |
| --- |
| // filename: Test.java  public class Test  {      int x = 2;      Test(int i) { x = i; }      public static void main(String[] args) {           Test t = new Test(5);           System.out.println("x = " + t.x);      }  } |

Output of the above program is “x = 5”. The initialization with class declaration in Java is like initialization using [Initializer List](https://www.geeksforgeeks.org/when-do-we-use-initializer-list-in-c/) in C++. So, in the above program, the value assigned inside the constructor overwrites the previous value of x which is 2, and x becomes 5.

As an exercise, predict the output of following program.

filter\_none

edit

play\_arrow

brightness\_4

|  |
| --- |
| // filename: Test2.java  class Test1 {      Test1(int x) {          System.out.println("Constructor called " + x);      }  }    // This class contains an instance of Test1  class Test2 {      Test1 t1 = new Test1(10);        Test2(int i) { t1 = new Test1(i); }        public static void main(String[] args) {           Test2 t2 = new Test2(5);      }  } |

The output of the program is Constructor called 10 Constructor called 5.  
First t2 object is instantiated in the main method. As the order of initialization of local variables comes first than the constructor,first the instance variable (t1), in the class Test2 is allocated to the memory. In this line a new Test1 object is created, constructor is called in class Test1 and ‘Constructor called 10’ is printed. Next the constructor of Test2 is called and again a new object of the class Test1 is created and ‘Constructor called 5’ is printed.

Output of Java Program | Set 4

Predict the output of following Java Programs:

**Question 1**

filter\_none

edit

play\_arrow

brightness\_4

|  |
| --- |
| // file name: Main.java    class Base {      protected void foo() {}  }  class Derived extends Base {      void foo() {}  }  public class Main {      public static void main(String args[]) {          Derived d = new Derived();          d.foo();      }  } |

Output: Compiler Error  
foo() is protected in Base and default in Derived. Default access is more restrictive. When a derived class overrides a base class function,[more restrictive access can’t be given to the overridden function](https://www.geeksforgeeks.org/more-restrictive-access-is-given-to-a-derived-class-method-in-java/). If we make foo() public, then the program works fine without any error. The behavior in C++ is different. [C++ allows to give more restrictive access to derived class methods.](https://www.geeksforgeeks.org/what-happens-when-more-restrictive-access-is-given-in-a-derived-class-method-in-c/)

**Question 2**

|  |
| --- |
| // file name: Main.java    class Complex {      private double re, im;      public String toString() {          return "(" + re + " + " + im + "i)";      }      Complex(Complex c) {          re = c.re;          im = c.im;      }  }    public class Main {      public static void main(String[] args) {          Complex c1 = new Complex();          Complex c2 = new Complex(c1);          System.out.println(c2);      }  } |

Output: Compiler Error in line “Complex c1 = new Complex();”  
In Java, if we write our own [copy constructor](https://www.geeksforgeeks.org/copy-constructor-in-java/) or parameterized constructor, then compiler doesn’t create the default constructor. This behavior is same as C++.

Output of Java program | Set 5

Predict the output of following Java Programs.

**Program 1:**

filter\_none

edit

play\_arrow

brightness\_4

|  |
| --- |
| // 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.

**Program 2:**

filter\_none

edit

play\_arrow

brightness\_4

|  |
| --- |
| // 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.

**Program 3:**

filter\_none

edit

play\_arrow

brightness\_4

|  |
| --- |
| // 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).

**Program 4:**

filter\_none

edit

play\_arrow

brightness\_4

|  |
| --- |
| // Main.java  public class Main  {      public static void main(String args[])      {          short s = 0;          int x = 07;          int y = 08;          int z = 112345;            s += z;          System.out.println("" + x + y + s);      }  } //end class |

**Output:**

Compile Error at line 8

**Explanation:**  
1. In Line 12 The “” in the println causes the numbers to be automatically cast as strings. So it doesn’t do addition, but appends together as string.  
2. In Line11 the += does an automatic cast to a short. However the number 123456 can’t be contained within a short, so you end up with a negative value (-7616).  
(NOTE – short 2 bytes -32,768 to 32,767), Here the number 123456 doesn’t mean the Value of int z,it shows the length of the int value  
3. Those other two are red herrings however as the code will never compile due to line 8.  
Any number beginning with zero is treated as an octal number (which is 0-7).

Output of Java Program | Set 6

**Difficulty level :**Intermediate

Predict the output of following Java Programs.

**Program 1:**

filter\_none

edit

play\_arrow

brightness\_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.

**Program 2:**

filter\_none

edit

play\_arrow

brightness\_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);      }  } |

**chevron\_right**

10

200

400

filter\_none

https://ide.geeksforgeeks.org/undefined

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.

**Program 3:**

filter\_none

edit

play\_arrow

brightness\_5

|  |
| --- |
| 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.

**Program 4:**

filter\_none

edit

play\_arrow

brightness\_5

|  |
| --- |
| 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.

# Output of Java Program | Set 7

**Difficulty level :**Intermediate

Predict the output of following Java Programs.

**Program 1 :**

filter\_none

edit

play\_arrow

brightness\_5

|  |
| --- |
| 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();      }  } |

**Options :**  
A) 20  
B) 100  
C) 1000  
D) 2  
**Answer : A) 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.

**Program 2 :**

filter\_none

edit

play\_arrow

brightness\_5

|  |
| --- |
| public class MyStuff  {      String name;        MyStuff(String n) {  name = n;  }        public static void main(String[] args)      {          MyStuff m1 = new MyStuff("guitar");          MyStuff m2 = new MyStuff("tv");          System.out.println(m2.equals(m1));      }        @Override      public boolean equals(Object obj)      {          MyStuff m = (MyStuff) obj;          if (m.name != null)  { return true;  }          return false;      }  } |

**Options :**  
A) The output is true and MyStuff fulfills the Object.equals() contract.  
B) The output is false and MyStuff fulfills the Object.equals() contract.  
C) The output is true and MyStuff does NOT fulfill the Object.equals() contract.  
D) The output is false and MyStuff does NOT fulfill the Object.equals() contract.

**Answer :** C) The output is true and MyStuff does NOT fulfill the Object.equals() contract.  
**Explanation :** As equals(Object obj) method in Object class, compares two objects on the basis of equivalence relation. But here we are just confirming that the object is null or not, So it doesn’t fulfill [Object.equals()](https://www.geeksforgeeks.org/overriding-equals-method-in-java/) contract. As m1 is not null, true will be printed.

**Program 3 :**

filter\_none

edit

play\_arrow

brightness\_5

|  |
| --- |
| class Alpha  {      public String type = "a ";      public Alpha() {  System.out.print("alpha "); }  }    public class Beta extends Alpha  {      public Beta()  {  System.out.print("beta ");  }        void go()      {          type = "b ";          System.out.print(this.type + super.type);      }        public static void main(String[] args)      {          new Beta().go();      }  } |

**Options :**  
A) alpha beta b b  
B) alpha beta a b  
C) beta alpha b b  
D) beta alpha a b

**Answer :** A) alpha beta b b  
**Explanation :** The statement **new Beta().go()**executes in two phases. In first phase Beta class constructor is called. There is no instance member present in Beta class. So now Beta class constructor is executed. As Beta class extends Alpha class, so call goes to Alpha class constructor as first statement by default(Put by the compiler) is **super()** in the Beta class constructor. Now as one instance variable(type) is present in Alpha class, so it will get memory and now Alpha class constructor is executed, then call return to Beta class constructor next statement. So alpha beta is printed.  
In second phase go() method is called on this object. As there is only one variable(type) in the object whose value is a. So it will be changed to b and printed two times. The [super keyword](http://quiz.geeksforgeeks.org/super-keyword/) here is of no use.

**Program 4 :**

filter\_none

edit

play\_arrow

brightness\_5

|  |
| --- |
| public class Test  {      public static void main(String[] args)      {          StringBuilder s1 = new StringBuilder("Java");          String s2 = "Love";          s1.append(s2);          s1.substring(4);          int foundAt = s1.indexOf(s2);          System.out.println(foundAt);      }  } |

**Options :**  
A) -1  
B) 3  
C) 4  
D) A **StringIndexOutOfBoundsException** is thrown at runtime.  
**Answer :** C) 4  
**Explanation :** *append(String str)* method,concatenate the str to *s1*. The *substring(int index)* method return the String from the given index to the end. But as there is no any String variable to store the returned string,so it will be destroyed.Now *indexOf(String s2)* method return the index of first occurrence of *s2*. So 4 is printed as s1=”JavaLove”.

**Program 5 :**

filter\_none

edit

play\_arrow

brightness\_5

|  |
| --- |
| class Writer  {      public  static void write()      {          System.out.println("Writing...");      }  }  class Author extends Writer  {      public  static void write()      {          System.out.println("Writing book");      }  }    public class Programmer extends Author  {      public  static void write()      {          System.out.println("Writing code");      }        public static void main(String[] args)      {          Author a = new Programmer();          a.write();      }  } |

**Options :**  
A) Writing…  
B) Writing book  
C) Writing code  
D) Compilation fails

**Answer :** B) Writing book  
**Explanation :** Since static methods can’t be overridden, it doesn’t matter which class object is created. As *a* is a*Author* referenced type, so always *Author* class method is called. If we remove *write()* method from *Author*class then *Writer*class method is called, as *Author*class extends *Writer* class.

# Output of Java Program | Set 8

**Difficulty level** : Intermediate

Predict the output of following Java Programs.

**Program 1:**

filter\_none

edit

play\_arrow

brightness\_5

|  |
| --- |
| class GfG  {      public static void main(String args[])      {          String s1 = new String("geeksforgeeks");          String s2 = new String("geeksforgeeks");          if (s1 == s2)              System.out.println("Equal");          else              System.out.println("Not equal");      }  } |

Output:

Not equal

**Explanation:** Since, s1 and s2 are two different objects the references are not the same, and the == operator compares object reference. So it prints “Not equal”, to compare the actual characters in the string .equals() method must be used.

**Program 2:**

filter\_none

edit

play\_arrow

brightness\_5

|  |
| --- |
| class Person  {      private void who()      {          System.out.println("Inside private method Person(who)");      }        public static void whoAmI()      {          System.out.println("Inside static method, Person(whoAmI)");      }        public void whoAreYou()      {          who();          System.out.println("Inside virtual method, Person(whoAreYou)");      }  }    class Kid extends Person  {      private void who()      {          System.out.println("Kid(who)");      }        public static void whoAmI()      {          System.out.println("Kid(whoAmI)");      }        public void whoAreYou()      {          who();          System.out.println("Kid(whoAreYou)");      }  }  public class Gfg  {      public static void main(String args[])      {          Person p = new Kid();          p.whoAmI();          p.whoAreYou();      }  } |

Output:

Inside static method, People(whoAmI)

Kid(who)

Kid(whoAreYou)

**Explanation:** Static binding (or compile time) happens for static methods. Here *p.whoAmI()* calls the static method so it is called during compile time hence results in static binding and prints the method in *People* class.  
Whereas *p.whoAreYou()* calls the method in *Kid* class since by default Java takes it as a virual method i.e, dynamic binding.

**Program 3:**

filter\_none

edit

play\_arrow

brightness\_5

|  |
| --- |
| class GfG  {      public static void main(String args[])      {          try          {              System.out.println("First statement of try block");              int num=45/3;              System.out.println(num);          }          catch(Exception e)          {              System.out.println("Gfg caught Exception");          }          finally          {              System.out.println("finally block");          }          System.out.println("Main method");      }  } |

Output:

First statement of try block

15

finally block

Main method

**Explanation:**  
Since there is no exception, the catch block is not called, but the *finally* block is always executed after a try block whether the exception is handled or not.

**Program 4:**

filter\_none

edit

play\_arrow

brightness\_5

|  |
| --- |
| class One implements Runnable  {      public void run()      {          System.out.print(Thread.currentThread().getName());      }  }  class Two implements Runnable  {      public void run()      {          new One().run();          new Thread(new One(),"gfg2").run();          new Thread(new One(),"gfg3").start();      }  }  class Three  {      public static void main (String[] args)      {          new Thread(new Two(),"gfg1").start();      }  } |

Output:

gfg1gfg1gfg3

**Explanation :**Initially new Thread is started with name *gfg1* then in class Two the first run method runs the thread with the name *gfg1*, then after that a new thread is created calling run method but since a new thread can be created by calling start method only so the previous thread does the action and again *gfg1* is printed.Now a new thread is created by calling the start method so a new thread starts with *gfg3* name and hence prints *gfg3*.

# Output of Java Program | Set 9

**Difficulty level** : Intermediate

Predict the output of following Java Programs.

**Program 1:**

filter\_none

edit

play\_arrow

brightness\_5

|  |
| --- |
| class Gfg  {      // constructor      Gfg()      {          System.out.println("Geeksforgeeks");      }        static Gfg a = new Gfg(); //line 8        public static void main(String args[])      {          Gfg b; //line 12          b = new Gfg();      }  } |

Output:

Geeksforgeeks

Geeksforgeeks

**Explanation:**  
We know that static variables are called when a class loads and static variables are called only once. Now line 13 results to creation of object which inturn calls the constructor and “Geeksforgeeks” is printed second time.  
If in line 8 static variable would not have been used the object would have been called recursively infinitely leading to StackOverFlow error. See [this](https://ide.geeksforgeeks.org/wtntd4) for a sample run.

**Program 2:**

filter\_none

edit

play\_arrow

brightness\_5

|  |
| --- |
| class Gfg  {      static int num;      static String mystr;        // constructor      Gfg()      {          num = 100;          mystr = "Constructor";      }        // First Static block      static      {          System.out.println("Static Block 1");          num = 68;          mystr = "Block1";      }        // Second static block      static      {          System.out.println("Static Block 2");          num = 98;          mystr = "Block2";      }        public static void main(String args[])      {          Gfg a = new Gfg();          System.out.println("Value of num = " + a.num);          System.out.println("Value of mystr = " + a.mystr);      }  } |

Output:

Static Block 1

Static Block 2

Value of num = 100

Value of mystr = Constructor

**Explanation:**  
Static block gets executed when the class is loaded in the memory. A class can have multiple Static blocks, which are executed in the same sequence in which they have been written into the program.  
**Note**: Static Methods can access class variables without using object of the class. Since constructor is called when a new instance is created so firstly the static blocks are called and after that the constructor is called. If we would have run the same program without using object, the constructor would not have been called.

**Program 3:**

filter\_none

edit

play\_arrow

brightness\_5

|  |
| --- |
| class superClass  {      final public int calc(int a, int b)      {          return 0;      }  }  class subClass extends superClass  {      public int calc(int a, int b)      {          return 1;      }  }  public class Gfg  {      public static void main(String args[])      {          subClass get = new subClass();          System.out.println("x = " + get.calc(0, 1));      }  } |

Output:

Compilation fails.

**Explanation:**  
The method calc() in class superClass is final and so cannot be overridden.

**Program 4:**

filter\_none

edit

play\_arrow

brightness\_5

|  |
| --- |
| public class Gfg  {      public static void main(String[] args)      {          Integer a = 128, b = 128;          System.out.println(a == b);            Integer c = 100, d = 100;          System.out.println(c == d);      }  } |

Output:

false

true

**Explanation:** In the source code of Integer object we will find a method ‘valueOf’ in which we can see that the range of the Integer object lies from IntegerCache.low(-128) to IntegerCache.high(127). Therefore the numbers above 127 will not give the expected output. The range of IntegerCache can be observed from the source code of the IntegerCache class. Please refer [this](https://blogs.oracle.com/darcy/entry/boxing_and_caches_integer_valueof) for details.

# Output of Java programs | Set 10 (Garbage Collection)

Prerequisite – [Garbage Collection in Java](https://www.geeksforgeeks.org/garbage-collection-java/)

**Difficulty level :**Intermediate  
In Java, object destruction is taken care by the [Garbage Collector](https://www.geeksforgeeks.org/garbage-collection-java/) module and the objects which do not have any references to them are eligible for garbage collection. Below are some important output questions on Garbage collection.  
Predict the output of following Java Programs:

* **Program 1 :**

filter\_none

edit

play\_arrow

brightness\_5

|  |
| --- |
| public class Test  {      public static void main(String[] args) throws InterruptedException      {          String str = new String("GeeksForGeeks");            // making str eligible for gc          str = null;            // calling garbage collector          System.gc();            // waiting for gc to complete          Thread.sleep(1000);            System.out.println("end of main");      }        @Override      protected void finalize()      {          System.out.println("finalize method called");      }  } |

Output:

end of main

**Explanation :** We know that [finalize()](https://www.geeksforgeeks.org/g-fact-24-finalfinally-and-finalize-in-java/) method is called by Garbage Collector on an object before destroying it. But here, the trick is that the str is String class object, not the Test class. Therefore, finalize() method of String class(if overridden in String class) is called on str. If a class doesn’t override finalize method, then by default Object class finalize() method is called.

**Program 2 :**

|  |
| --- |
| public class Test  {      public static void main(String[] args) throws InterruptedException      {          Test t = new Test();            // making t eligible for garbage collection          t = null;            // calling garbage collector          System.gc();            // waiting for gc to complete          Thread.sleep(1000);            System.out.println("end main");      }        @Override      protected void finalize()      {          System.out.println("finalize method called");          System.out.println(10/0);      }    } |

Output:

finalize method called

end main

**Explanation :**  
When Garbage Collector calls finalize() method on an object, it **ignores** all the exceptions raised in the method and program will terminate normally.

**Program 3 :**

filter\_none

edit

play\_arrow

brightness\_5

|  |
| --- |
| public class Test  {      static Test t ;        static int count =0;        public static void main(String[] args) throws InterruptedException      {          Test t1 = new Test();            // making t1 eligible for garbage collection          t1 = null; // line 12            // calling garbage collector          System.gc(); // line 15            // waiting for gc to complete          Thread.sleep(1000);            // making t eligible for garbage collection,          t = null; // line 21            // calling garbage collector          System.gc(); // line 24            // waiting for gc to complete          Thread.sleep(1000);            System.out.println("finalize method called "+count+" times");        }        @Override      protected void finalize()      {          count++;            t = this; // line 38        }    } |

**Output:**

finalize method called 1 times

**Explanation :**  
After execution of line 12, t1 becomes eligible for garbage collection. So when we call garbage collector at line 15, Garbage Collector will call finalize() method on t1 before destroying it. But in finalize method, in line 38, we are again referencing the same object by t, so after execution of line 38,[this](http://quiz.geeksforgeeks.org/this-reference-in-java/) object is no longer eligible for garbage collection. Hence, Garbage Collector will not destroy the object.

Now again in line 21, we are making same object eligible for garbage collection one more time. Here, we have to clear about one fact about Garbage Collector i.e. it will call finalize() method on a particular object exactly **one** time. Since on this object, finalize() method is already called, so now Garbage Collector will destroy it without calling finalize() method again.

* **Program 4 :**

filter\_none

edit

play\_arrow

brightness\_5

|  |
| --- |
| public class Test  {      public static void main(String[] args)      {          // How many objects are eligible for          // garbage collection after this line?          m1();  // Line 5      }        static void m1()      {          Test t1 = new Test();          Test t2 = new Test();      }  } |

**Question :**  
How many objects are eligible for garbage collection after execution of line 5 ?  
**Answer :**

2

**Explanation :**  
Since t1 and t2 are local objects of m1() method, so they become eligible for garbage collection after complete execution of method unless any of them is returned.

* **Program 5 :**

filter\_none

edit

play\_arrow

brightness\_5

|  |
| --- |
| public class Test  {      public static void main(String [] args)      {          Test t1 = new Test();          Test t2 = m1(t1); // line 6          Test t3 = new Test();          t2 = t3; // line 8        }        static Test m1(Test temp)      {          temp = new Test();          return temp;      }  } |

**Question :**  
How many objects are eligible for garbage collection after execution of line 8?  
**Answer :**

1

**Explanation :**  
By the time line 8 has executed, the only object without a reference is the one generated i.e as a result of line 6. Remember that “[Java is strictly pass by value](https://www.geeksforgeeks.org/passing-and-returning-objects-in-java/)” so the reference variable t1 is not affected by the m1() method. We can check it using finalize() method. The statement “System.out.println(this.hashcode())” in finalize() method print the object hashcode value on which finalize() method is called,and then just compare the value with other objects hashcode values created in main method.