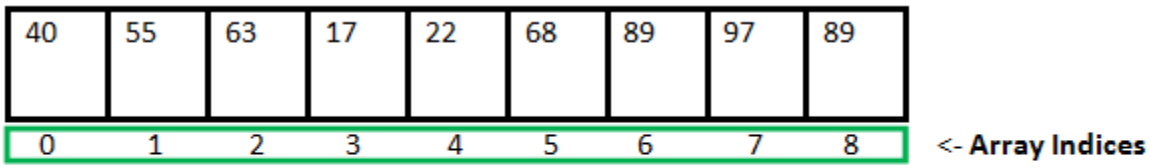
Java Enterprise Edition

ANUDIP FOUNDATION

Assignments

Array Declarations , Constructions , Initialization



Array Length = 9
First Index = 0
Last Index = 8

type var-name[];

OR

type[] var-name;

Example:

```
// both are valid declarations
int intArray[];
or int[] intArray;

byte byteArray[];
short shortsArray[];
boolean booleanArray[];
long longArray[];
float floatArray[];
double doubleArray[];
char charArray[];

// an array of references to objects of
```

```
// the class MyClass (a class created by
// user)
MyClass myClassArray[];

Object[]  ao,        // array of Object
Collection[] ca;  // array of Collection
                  // of unknown type
```

Instantiating an Array in Java

var-name = new type [size];

Example:

```
int intArray[];    //declaring array
intArray = new int[20];  // allocating memory to array
```

OR

```
int[] intArray = new int[20]; // combining both statements in one
```

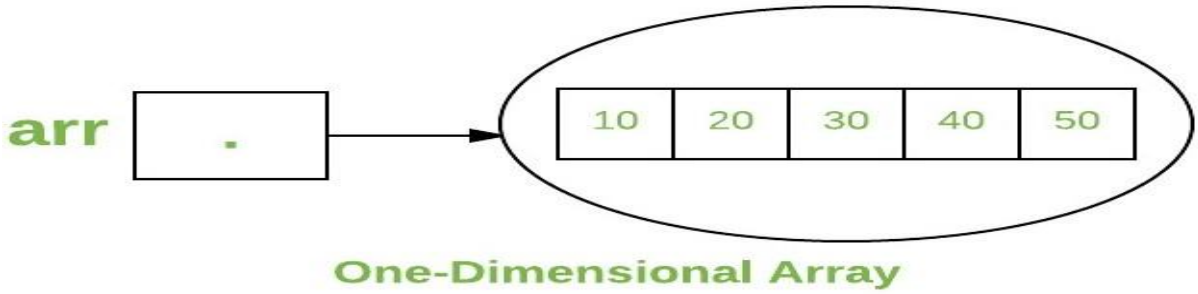
// Java program to illustrate creating an array

```
// of integers, puts some values in the array,
// and prints each value to standard output.

class GFG
{
        public static void main (String[] args)
        {
        // declares an Array of integers.
        int[] arr;

        // allocating memory for 5 integers.
        arr = new int[5];
```

```
        // initialize the first elements of the array
        arr[0] = 10;

        // initialize the second elements of the array
        arr[1] = 20;

        //so on...
        arr[2] = 30;
        arr[3] = 40;
        arr[4] = 50;

        // accessing the elements of the specified array
        for (int i = 0; i < arr.length; i++)
                System.out.println("Element at index " + i +
                                                        " : "+ arr[i]);
        }
}
```

Output:

```
Element at index 0 : 10
Element at index 1 : 20
Element at index 2 : 30
Element at index 3 : 40
Element at index 4 : 50
```



One-Dimensional Array

```
// Java program to illustrate creating an array of
// objects

class Student
{
	public int roll_no;
	public String name;
	Student(int roll_no, String name)
	{
		this.roll_no = roll_no;
		this.name = name;
	}
}

// Elements of the array are objects of a class Student.
public class GFG
{
	public static void main (String[] args)
	{
		// declares an Array of integers.
		Student[] arr;

		// allocating memory for 5 objects of type Student.
```

```
        arr = new Student[5];

        // initialize the first elements of the array
        arr[0] = new Student(1,"aman");

        // initialize the second elements of the array
        arr[1] = new Student(2,"vaibhav");

        // so on...
        arr[2] = new Student(3,"shikar");
        arr[3] = new Student(4,"dharmesh");
        arr[4] = new Student(5,"mohit");

        // accessing the elements of the specified array
        for (int i = 0; i < arr.length; i++)
            System.out.println("Element at " + i + " : " +
                                arr[i].roll_no +" "+ arr[i].name);
    }
}
```

Output:

Element at 0 : 1 aman

Element at 1 : 2 vaibhav

Element at 2 : 3 shikar

Element at 3 : 4 dharmesh

Element at 4 : 5 mohit

MCQ's

1) An Array in Java is a collection of elements of ___ data type.
A) Same

B) Different
C) -
D) -
Answer [=]
A

2) The Java Virtual Machine (JVM) implements arrays as ___ type.
A) Primitive
B) Object
C) -
D) -
Answer [=]
B
Explanation:
That is the reason why Java Array has predefined methods.

3) Unlike C-Arrays, the Java-Arrays have ___.
A) Names
B) Values
C) Methods and Fields
D) None
Answer [=]
C

4) An array declaration in Java without initialization ___ memory.
A) Does not allocate
B) Allocates memory
C) -
D) -
Answer [=]
A
Explanation:
Only initialization causes memory to be allocated.

5) In Java language, an array index starts with ___.
A) -1
B) 0
C) 1
D) Any integer
Answer [=]
B

6) Which are the special symbols used to declare an array in Java?
A) Braces { }
B) Parentheses ()
C) Square Brackets [ ]
D) Angled Brackets < >
Answer [=]
C

7) Which are the special symbols used to initialize an array at the time of the declaration itself?
A) Parentheses ( )

B) Square Brackets [ ]
C) Braces { }
D) Angled Brackets < >
Answer [=]
C
Explanation:

```
int[] nums = {1,3,6};
```

8) It is possible to skip initializing some elements of the array during Shorthand Initialization. (TRUE / FALSE)
A) FALSE
B) TRUE
C) -
D) -
Answer [=]
A
Explanation:
No, you can not skip any elements. All elements need to be initialized in one go or at the same time.

9) In Java, an array can be declared without initialization without mentioning the size. (TRUE / FALSE)
A) TRUE
B) FALSE
C) -
D) -
Answer [=]
A
Explanation:
It is a Lazy initialization of an array.

10) What is the output of the below Java code snippet with arrays?

```
static int[] nums;

public static void main(String args[])

{

  System.out.println(nums.length);

}
```

A) 0
B) null
C) Compiler error
D) Runtime Exception - Null Pointer Exception
Answer [=]
D

Assignment Operators

1: Arithmetic Operators

```
class Main {
  public static void main(String[] args) {

    // declare variables
    int a = 12, b = 5;

    // addition operator
    System.out.println("a + b = " + (a + b));

    // subtraction operator
    System.out.println("a - b = " + (a - b));

    // multiplication operator
    System.out.println("a * b = " + (a * b));

    // division operator
    System.out.println("a / b = " + (a / b));

    // modulo operator
    System.out.println("a % b = " + (a % b));
  }
}
```

Output

```
a + b = 17
a - b = 7
a * b = 60
a / b = 2
a % b = 2
```

2: Assignment Operators

```
class Main {
  public static void main(String[] args) {

    // create variables
    int a = 4;

    int var;

    // assign value using =
    var = a;
    System.out.println("var using =: " + var);

    // assign value using =+
    var += a;
    System.out.println("var using +=: " + var);

    // assign value using =*
    var *= a;
    System.out.println("var using *=: " + var);
  }
}
```

Output

```
var using =: 4
var using +=: 8
var using *=: 32
```

3.

```
public class Test {

  public static void main(String args[]) {
    int a = 10;
    int b = 20;
    int c = 0;

    c = a + b;
```

```
      System.out.println("c = a + b = " + c );

      c += a ;
      System.out.println("c += a  = " + c );

      c -= a ;
      System.out.println("c -= a = " + c );

      c *= a ;
      System.out.println("c *= a = " + c );

      a = 10;
      c = 15;
      c /= a ;
      System.out.println("c /= a = " + c );

      a = 10;
      c = 15;
      c %= a ;
      System.out.println("c %= a  = " + c );

      c <<= 2 ;
      System.out.println("c <<= 2 = " + c );

      c >>= 2 ;
      System.out.println("c >>= 2 = " + c );

      c >>= 2 ;
      System.out.println("c >>= 2 = " + c );

      c &= a ;
      System.out.println("c &= a  = " + c );

      c ^= a ;
      System.out.println("c ^= a   = " + c );

      c |= a ;
      System.out.println("c |= a   = " + c );
   }
}
```

This will produce the following result –

Output

```
c = a + b = 30
c += a  = 40
c -= a = 30
c *= a = 300
c /= a = 1
c %= a  = 5
```

c <<= 2 = 20
c >>= 2 = 5
c >>= 2 = 1
c &= a = 0
c ^= a   = 10
c |= a   = 10

MCQ's

1) An Arithmetic expression in Java involves which Operators or Operations?
A) Addition (+), Subtraction (-)
B) Multiplication (*), Division (/)
C) Modulo Division (%), Increment/Decrement (++/--), Unary Minus (-), Unary Plus (+)
D) All the above
Answer [=]
D

2) Choose the Compound Assignment Arithmetic Operators in Java below.
A) +=, -=
B) *=, /=
C) %=
D) All the above
Answer [=]
D

3)

What is the output of the below Java code snippet?

```
int a = 2 - - 7;
System.out.println(a);
```

A) -5
B) 10
C) 9
D) Compiler Error
Answer [=]
C
Explanation:

Minus of Minus is Plus. So 2 - - 7 becomes 2+7.

4)

What is the output of Java code snippet below?

```
short p = 1;
short k = p + 2;
System.out.println(k);
```

A) 1
B) 2
C) 3
D) Compiler error
Answer [=]
D
Explanation:
Numbers are treated as int type by default. So an int value cannot be assigned to a short variable. You have to type cast the whole expression.

```
short k = (short)(p + 2);
```

5)

What is the output of Java code snippet?

```
short k=1;
k += 2;
System.out.println(k);
```

A) 1
B) 2
C) 3
D) Compiler error about Type Casting
Answer [=]
C
Explanation:
Compound assignment operators automatically convert the expression value to the left-hand side data type.

```
k = k + 1; //Error
k += 1; //Works
k++; //Works
```

6)

What is the output of the Java code snippet?

```
int a=5, b=10, c=15;
a -= 3;
b *= 2;
c /= 5;
System.out.println(a +" " + b + " " + c);
```

A) 2 20 3
B) 2 20 5

C) 2 10 5
D) -2 20 3
Answer [=]
A
Explanation:

```
a = a - 3;
b = b*2;
c = c/5;
```

7)

How do you rewrite the below Java code snippet?

```
int p=10;
p = p%3;
```

A)

```
p=%3;
```

B)

```
p%=3;
```

C)

```
p=3%;
```

D) None of the above
Answer [=]
B

8) Which is the arithmetic operator in Java that gives the Remainder of Division?
A) /
B) @
C) %
D) &
Answer [=]
C
Explanation:

```
//Modulo Division operator
// or simply Modulus Operator
int a = 14%5;
//a holds 4


5)14(2
```

```
 -10
------
  4
```

9) Arithmetic operators +, -, /, * and % have which Associativity?
A) Right to Left
B) Left to Right
C) Right to Right
D) Left to Left
Answer [=]
B

10) Between Postfix and Prefix arithmetic operators in Java, which operators have more priority?
A) Postfix operators have more priority than Prefix operators
B) Prefix operators have more priority than Postfix operators
C) Both Prefix and Postfix operators have equal priority
D) None of the above
Answer [=]
A
Explanation:
op++, op-- have more priority than --op, ++op.

Widening Casting

```
public class Main {
  public static void main(String[] args) {
    int myInt = 9;
    double myDouble = myInt; // Automatic casting: int to double

    System.out.println(myInt);
    System.out.println(myDouble);
  }
}
```

o/p

```
9
9.0
```

Narrowing Casting

```
public class Main {
  public static void main(String[] args) {
    double myDouble = 9.78;
    int myInt = (int) myDouble; // Manual casting: double to int

    System.out.println(myDouble);   // Outputs 9.78
    System.out.println(myInt);      // Outputs 9
  }
}
```

O/P

```
9.78
9
```

```
class Test {
```

```
    public static void main(String[] args) {
        // Casting conversion (5.4) of a float literal to
        // type int. Without the cast operator, this would
        // be a compile-time error, because this is a
        // narrowing conversion (5.1.3):
        int i = (int)12.5f;
        // String conversion (5.4) of i's int value:
        System.out.println("(int)12.5f==" + i);
        // Assignment conversion (5.2) of i's value to type
        // float. This is a widening conversion (5.1.2):
        float f = i;
        // String conversion of f's float value:
        System.out.println("after float widening: " + f);
        // Numeric promotion (5.6) of i's value to type
        // float. This is a binary numeric promotion.
        // After promotion, the operation is float*float:
        System.out.print(f);
        f = f * i;
        // Two string conversions of i and f:
        System.out.println("*" + i + "==" + f);
        // Method invocation conversion (5.3) of f's value
        // to type double, needed because the method Math.sin
        // accepts only a double argument:
        double d = Math.sin(f);
        // Two string conversions of f and d:
        System.out.println("Math.sin(" + f + ")==" + d);
    }
}
```

O/P

```
(int)12.5f==12
after float widening: 12.0
12.0*12==144.0
Math.sin(144.0)==-0.49102159389846934
```

MCQ's

1) What are the Type Conversions available in Java language?
A) Narrowing Type Conversion
B) Widening Type Conversion
C) A and B
D) None of the above
Answer [=]
C

2) What is a higher data type in Java language?
A) A data type which holds more data than other data types
B) A data type whose size is more than other data types
C) A data type which can hold more precision digits than other data types
D) All the above
Answer [=]
D
Explanation:
Float is bigger than short

double is bigger than float

3) What is a Widening Type Conversion in Java?
A) Conversion of data from higher data type to lower data type
B) Conversion of data from lower data type to higher data type
C) Conversion of data from any data type to any data type
D) None of the above
Answer [=]
B

4) What is a Narrowing Type Conversion in Java?
A) Conversion of data from lower data type to higher data type
B) Conversion data from a higher data type to a lower data type
C) Conversion of data from any data type to any data type
D) None of the above
Answer [=]
B

5) What is the result of a Narrowing type conversion?
A) Loss of data
B) Addition of data
C) Corruption of data
D) None of the above
Answer [=]
A

Explanation:

```
int a =(int)1.2f;
//a holds 1
```

6) What is the result of a Widening Type Conversion in Java?
A) Loss of data
B) Gain of data
C) No change
D) None of the above
Answer [=]
C
Explanation:

```
int a=456;
float b = a; //No change of data
//b holds 456.0;
```

7) Type promotion in Java usually refers to _____.
A) Narrowing Type Conversion
B) Widening Type Conversion
C) No Type Conversion
D) None of the above
Answer [=]
B
Explanation:
All integers are promoted to int or long.

All characters are promoted to int from char or long from char.

All float numbers are promoted to double.

8) Type Casting in Java usually refers to _____?
A) Narrowing Type Conversion
B) Widening Type Conversion
C) No Type Conversion
D) None of the above
Answer [=]
A

9) Explicit Type Conversion in Java refers to ___?
A) Narrowing Type Conversion
B) Widening Type Conversion
C) No Type Conversion
D) None of the above
Answer [=]
A

10) Implicit Type Conversion in Java is also called ___?
A) Narrowing Type Conversion
B) Widening Type Conversion

C) No Type Conversion
D) None of the above
Answer [=]
B
Explanation:
Implicit type conversion is an Automatic Type Promotion from a lower data type to a higher data type.

Does Java Use Pass-By-Value Semantics?

Java is always **pass-by-value**.
Unfortunately, we never handle an object at all, instead juggling object-handles called *references* (which are passed by value of course). The chosen terminology and semantics easily confuse many beginners.

It goes like this:

```
public static void main(String[] args) {
    Dog aDog = new Dog("Max");
    Dog oldDog = aDog;

    // we pass the object to foo
    foo(aDog);
    // aDog variable is still pointing to the "Max" dog when foo(...) returns
    aDog.getName().equals("Max"); // true
    aDog.getName().equals("Fifi"); // false
    aDog == oldDog; // true
}

public static void foo(Dog d) {
    d.getName().equals("Max"); // true
    // change d inside of foo() to point to a new Dog instance "Fifi"
    d = new Dog("Fifi");
    d.getName().equals("Fifi"); // true
}
```

In the example above aDog.getName() will still return "Max". The value aDog within main is not changed in the function foo with the Dog "Fifi" as the object reference is passed by value. If it were passed by reference, then the aDog.getName() in main would return "Fifi" after the call to foo.

Likewise:

```
public static void main(String[] args) {
    Dog aDog = new Dog("Max");
    Dog oldDog = aDog;

    foo(aDog);
    // when foo(...) returns, the name of the dog has been changed to "Fifi"
    aDog.getName().equals("Fifi"); // true
    // but it is still the same dog:
    aDog == oldDog; // true
}

public static void foo(Dog d) {
    d.getName().equals("Max"); // true
    // this changes the name of d to be "Fifi"
    d.setName("Fifi");
}
```

In the above example, Fifi is the dog's name after call to foo(aDog) because the object's name was set inside of foo(...). Any operations that foo performs on d are such that, for all practical purposes, they are performed on aDog, but it is **not** possible to change the value of the variable aDog itself.

Garbage Collection in Java

Garbage Collection

- Overview of Memory Management and Garbage Collection

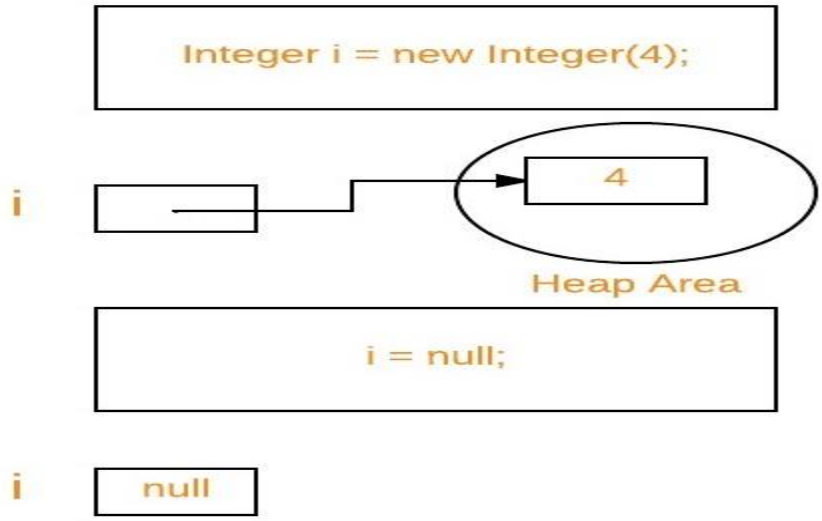
- Overview of Java's Garbage Collector

Writing Code That Explicitly Makes Objects Eligible for Garbage Collection

Important terms :

1. **Unreachable objects :** An object is said to be unreachable iff it doesn't contain any reference to it. Also note that objects which are part of island of isolation are also unreachable.

```
Integer i = new Integer(4);
// the new Integer object is reachable  via the reference in 'i'
i = null;
// the Integer object is no longer reachable.
```



2.Eligibility for garbage collection : An object is said to be eligible for GC(garbage collection) iff it is unreachable. In above image, after *i = null;* integer object 4 in heap area is eligible for garbage collection.

Ways to make an object eligible for GC

- Even though the programmer is not responsible to destroy useless objects but it is highly recommended to make an object unreachable(thus eligible for GC) if it is no longer required.
- There are generally four different ways to make an object eligible for garbage collection.
    1. Nullifying the reference variable
    2. Re-assigning the reference variable
    3. Object created inside method
    4. Island of Isolation

Once we made object eligible for garbage collection, it may not destroy immediately by the garbage collector. Whenever JVM runs the Garbage Collector program, then only the object will be destroyed. But when JVM runs Garbage Collector, we can not expect.

☐ We can also request JVM to run Garbage Collector. There are two ways to do it :

1. **Using *System.gc()* method** : System class contain static method *gc()* for requesting JVM to run Garbage Collector.
2. **Using *Runtime.getRuntime().gc()* method** : Runtime class allows the application to interface with the JVM in which the application is running. Hence by using its gc() method, we can request JVM to run Garbage Collector.

```
// Java program to demonstrate requesting
```

```
// JVM to run Garbage Collector
public class Test
{
	public static void main(String[] args) throws InterruptedException
	{
		Test t1 = new Test();
		Test t2 = new Test();

		// Nullifying the reference variable
		t1 = null;

		// requesting JVM for running Garbage Collector
		System.gc();

		// Nullifying the reference variable
		t2 = null;

		// requesting JVM for running Garbage Collector
		Runtime.getRuntime().gc();

	}

	@Override
	// finalize method is called on object once
	// before garbage collecting it
	protected void finalize() throws Throwable
	{
		System.out.println("Garbage collector called");
		System.out.println("Object garbage collected : " + this);
	}
}
```

Output:

```
Garbage collector called
Object garbage collected : Test@46d08f12
Garbage collector called
Object garbage collected : Test@481779b8
```

- **Note :**
  1. There is no guarantee that any one of above two methods will definitely run Garbage Collector.
  2. The call *System.gc()* is effectively equivalent to the call : *Runtime.getRuntime().gc()*

Finalization

- Just before destroying an object, Garbage Collector calls *finalize()* method on the object to perform cleanup activities. Once *finalize()* method completes, Garbage Collector destroys that object.
- *finalize()* method is present in Object class with following prototype.
- protected void finalize() throws Throwable
  Based on our requirement, we can override *finalize()* method for perform our cleanup activities like closing connection from database.

  Note :
  1. The *finalize()* method called by Garbage Collector not JVM. Although Garbage Collector is one of the module of JVM.

2. Object class *finalize()* method has empty implementation, thus it is recommended to override *finalize()* method to dispose of system resources or to perform other cleanup.
3. The *finalize()* method is never invoked more than once for any given object.
4. If an uncaught exception is thrown by the *finalize()* method, the exception is ignored and finalization of that object terminates.

For examples on *finalize()* method, please see Output of Java programs | Set 10 (Garbage Collection)

Let's take a real-life example, where we use the concept of garbage collector.

Suppose you go for the internship at GeeksForGeeks and their you were told to write a program, to count the number of Employees working in the company(excluding interns).To make this program, you have to use the concept of a garbage collector.

This is the actual task you were given at the company:-

Q.Write a program to create a class called Employee having the following data members.

1.An ID for storing unique id allocated to every employee.

2.Name of employee.

3.age of an employee.

Also, provide the following methods-

1. A parameterized constructor to initialize name and age. The ID should be initialized in this constructor.
2. A method show() to display ID, name, and age.
3. A method showNextId() to display the ID of the next employee.

Now any beginner, who doesn't have knowledge on garbage collector will code like this:

```
//Program to count number
//of employees working
//in a company

class Employee
{
        private int ID;
        private String name;
        private int age;
        private static int nextId=1;
        //it is made static because it
        // is keep common among all and
        // shared by all objects
        public Employee(String name,int age)
        {
                this.name = name;
                this.age = age;
                this.ID = nextId++;
        }
        public void show()
        {
                System.out.println
                ("Id="+ID+"\nName="+name+"\nAge="+age);
        }
        public void showNextId()
        {
                System.out.println
                ("Next employee id will be="+nextId);
```

```
        }
}
class UseEmployee
{
        public static void main(String []args)
        {
                Employee E=new Employee("GFG1",56);
                Employee F=new Employee("GFG2",45);
                Employee G=new Employee("GFG3",25);
                E.show();
                F.show();
                G.show();
                E.showNextId();
                F.showNextId();
                G.showNextId();

                        { //It is sub block to keep
                        // all those interns.
                        Employee X=new Employee("GFG4",23);
                        Employee Y=new Employee("GFG5",21);
                        X.show();
                        Y.show();
                        X.showNextId();
                        Y.showNextId();
                }
                //After countering this brace, X and Y
                //will be removed.Therefore,
                //now it should show nextId as 4.
                E.showNextId();//Output of this line
                //should be 4 but it will give 6 as output.
        }
}
```

Output:

```
Id=1
Name=GFG1
Age=56
Id=2
Name=GFG2
Age=45
Id=3
Name=GFG3
Age=25
Next employee id will be=4
Next employee id will be=4
Next employee id will be=4
Id=4
Name=GFG4
Age=23
Id=5
Name=GFG5
Age=21
```

```
Next employee id will be=6
Next employee id will be=6
Next employee id will be=6
```

Now to get the correct output:

Now garbage collector(gc) will see 2 objects free. Now to decrement nextId,gc(garbage collector) will call method finalize() only when we programmers have override it in our class. And as mentioned previously, we have to request gc(garbage collector) and for this, we have to write the following 3 steps before closing brace of sub-block.

1. Set references to null(i.e X = Y = null;)
2. Call, System.gc();
3. Call, System.runFinalization();

Now the correct code for counting the number of employees(excluding interns)

```
// Correct code to count number
// of employees excluding interns.
class Employee
{
        private int ID;
        private String name;
        private int age;
        private static int nextId=1;
        //it is made static because it
        // is keep common among all and
        // shared by all objects
        public Employee(String name,int age)
        {
                this.name = name;
                this.age = age;
                this.ID = nextId++;
        }
        public void show()
        {
                System.out.println
                ("Id="+ID+"\nName="+name+"\nAge="+age);
        }
        public void showNextId()
        {
                System.out.println
                ("Next employee id will be="+nextId);
        }
        protected void finalize()
        {
                --nextId;
                //In this case,
                //gc will call finalize()
                //for 2 times for 2 objects.
        }
}

// it is closing brace of Employee class
class UseEmployee
{
```

```
        public static void main(String []args)
        {
                Employee E=new Employee("GFG1",56);
                Employee F=new Employee("GFG2",45);
                Employee G=new Employee("GFG3",25);
                E.show();
                F.show();
                G.show();
                E.showNextId();
                F.showNextId();
                G.showNextId();

                {
                        //It is sub block to keep
                        // all those interns.
                        Employee X=new Employee("GFG4",23);
                        Employee Y=new Employee("GFG5",21);
                        X.show();
                        Y.show();
                        X.showNextId();
                        Y.showNextId();
                        X = Y = null;
                        System.gc();
                        System.runFinalization();
                }
        E.showNextId();
        }
}
```

Output:

```
Id=1
Name=GFG1
Age=56
Id=2
Name=GFG2
Age=45
Id=3
Name=GFG3
Age=25
Next employee id will be=4
Next employee id will be=4
Next employee id will be=4
Id=4
Name=GFG4
Age=23
Id=5
Name=GFG5
Age=21
Next employee id will be=6
Next employee id will be=6
Next employee id will be=4
```

MCQ's

1. Which of the following has the highest memory requirement?
   a) Heap
   b) Stack
   c) JVM
   d) Class

Answer:c
Explanation: JVM is the super set which contains heap, stack, objects, pointers, etc.

2. Where is a new object allocated memory?
a)Youngspace
b)Oldspace
c)YoungorOldspacedependingonspaceavailability
d) JVM

Answer:a
Explanation: A new object is always created in young space. Once young space is full, a special young collection is run where objects which have lived long enough are moved to old space and memory is freed up in young space for new objects.

3. Which of the following is a garbage collection technique?
a)Cleanupmodel
b)Markandsweepmodel
c)Spacemanagementmodel
d) Sweep model

Answer:b
Explanation: A mark and sweep garbage collection consists of two phases, the mark phase and the sweep phase. I mark phase all the objects reachable by java threads, native handles and other root sources are marked alive and others are garbage. In sweep phase, the heap is traversed to find gaps between live objects and the gaps are marked free list used for allocating memory to new objects.

4. What is -Xms and -Xmx while starting jvm?
   a)Initial;Maximummemory
   b)Maximum;Initialmemory
   c)Maximummemory
   d) Initial memory
   Answer:a
   Explanation: JVM will be started with Xms amount of memory and will be able to use a maximum of Xmx amount of memory. java -Xmx2048m -Xms256m.
5. Which exception is thrown when java is out of memory?
   a)MemoryFullException
   b)MemoryOutOfBoundsException
   c)OutOfMemoryError
   d) MemoryError

Answer:c
Explanation: The Xms flag has no default value, and Xmx typically has a default value of 256MB. A common use for these flags is when you encounter a java.lang.OutOfMemoryError.

Primitive Data Types

1. boolean type

- The `boolean` data type has two possible values, either `true` or `false`.
- Default value: `false`.
- They are usually used for **true/false** conditions.

Example 1: Java boolean data type

```
class Main {
  public static void main(String[] args) {

    boolean flag = true;
    System.out.println(flag);    // prints true
  }
}
```

2. byte type

- The `byte` data type can have values from **-128** to **127** (8-bit signed two's complement integer).
- If it's certain that the value of a variable will be within -128 to 127, then it is used instead of int to save memory.
- Default value: 0

Example 2: Java byte data type

```
class Main {
  public static void main(String[] args) {

    byte range;
    range = 124;
```

```
    System.out.println(range);    // prints 124
  }
}
```

3. short type

- The short data type in Java can have values from **-32768** to **32767** (16-bit signed two's complement integer).
- If it's certain that the value of a variable will be within -32768 and 32767, then it is used instead of other integer data types (int, long).
- Default value: 0

Example 3: Java short data type

```
class Main {
  public static void main(String[] args) {

    short temperature;
    temperature = -200;
    System.out.println(temperature);  // prints -200
  }
}
```

4. int type

- The int data type can have values from **-2^{31}** to **$2^{31}-1$** (32-bit signed two's complement integer).

- If you are using Java 8 or later, you can use an unsigned 32-bit integer. This will have a minimum value of 0 and a maximum value of 2^{32}-1. To learn more, visit How to use the unsigned integer in java 8?
- Default value: 0

Example 4: Java int data type

```
class Main {
  public static void main(String[] args) {

    int range = -4250000;
    System.out.println(range);  // print -4250000
  }
}
```

5. long type

- The long data type can have values from **-2^{63}** to **$2^{63}-1$** (64-bit signed two's complement integer).
- If you are using Java 8 or later, you can use an unsigned 64-bit integer with a minimum value of **0** and a maximum value of **$2^{64}-1$**.
- Default value: 0

Example 5: Java long data type

```
class LongExample {
  public static void main(String[] args) {

    long range = -42332200000L;
    System.out.println(range);    // prints -42332200000
  }
}
```

Notice, the use of `L` at the end of `-42332200000`. This represents that it's an integral literal of the `long` type. You will learn about integral literals later in this article.

6. double type

- The `double` data type is a double-precision 64-bit floating-point.
- It should never be used for precise values such as currency.
- Default value: 0.0 (0.0d)

Example 6: Java double data type

```
class Main {
  public static void main(String[] args) {

    double number = -42.3;
    System.out.println(number);  // prints -42.3
  }
}
```

7. float type

- The `float` data type is a single-precision 32-bit floating-point.Learn more about single-precision and double-precision floating-point if you are interested.
- It should never be used for precise values such as currency.
- Default value: 0.0 (0.0f)

Example 7: Java float data type

```
class Main {
  public static void main(String[] args) {

    float number = -42.3f;
    System.out.println(number);  // prints -42.3
  }
}
```

Notice that, we have used `-42.3f` instead of `-42.3`in the above program. It's because `-42.3` is a `double` literal.

To tell the compiler to treat `-42.3` as `float` rather than `double`, you need to use `f` or `F`.

If you want to know about single-precision and double-precision, visit Java single-precision and double-precision floating-point.

8. char type

- It's a 16-bit Unicode character.
- The minimum value of the char data type is `'\u0000'` (0) and the maximum value of the is `'\uffff'`.
- Default value: `'\u0000'`

Example 8: Java char data type

```
class Main {
  public static void main(String[] args) {

    char letter = '\u0051';
    System.out.println(letter);  // prints Q
  }
}
```

Here, the Unicode value of `Q` is **`\u0051`**. Hence, we get `Q` as the output.

Here is another example:

```
class Main {
  public static void main(String[] args) {

    char letter1 = '9';
    System.out.println(letter1);  // prints 9

    char letter2 = 65;
    System.out.println(letter2);  // prints A

  }
}
```

Here, we have assigned `9` as a character (specified by single quotes) to the `letter1` variable. However, the `letter2` variable is assigned `65` as an integer number (no single quotes).

Hence, `A` is printed to the output. It is because Java treats characters as integral types and the ASCII value of `A` is 65. To learn more about ASCII, visit What is ASCII Code?.

String type

Java also provides support for character strings via `java.lang.String` class. Strings in Java are not primitive types. Instead, they are objects. For example,

```
String myString = "Java Programming";
```

Here, `myString` is an object of the `String` class.

MCQ's

1) What does a Data Type in Java refers to?
A) The place where data is stored
B) The technique how data is retreived
C) The type or variety of data being handled for reading and writing
D) None of the above
Answer [=]
C
Explanation:
Integers, Real numbers, Boolean, Characters etc

2) Choose the wrong statement about Java programming?
A) Java supports unsigned integers
B) Java supports signed integers
C) Java supports signed char
D) None of the above
Answer [=]
A
Explanation:
Only C language supports unsigned integers. Java does not support.

3) Which data type among the following is an implementation of Objects or OOPs?
A) byte
B) int
C) char
D) None of the above

Answer [=]
D
Explanation:
All primitive data types are implemented in a Non-Object Oriented way.

4) What is a Primitive Data Type in Java?
A) Data type, which is implemented in an Object-oriented way.
B) Data Type which is implemented in a machine-dependent way
C) Data Type which is implemented in a non-object oriented way.
D) None of the above
Answer [=]
C

5) which among the following is not a Data Type in Java?
A) short
B) int
C) long double
D) double
Answer [=]
C
Explanation:
"long double" is present only in C language.

6) Which among the following is not a valid Data Type in Java?
A) long
B) bool
C) double
D) float
Answer [=]
B
Explanation:
It is "boolean" not "bool".

7) Which is the data type used mostly to handle streams and buffers in Java language?
A) short
B) int
C) byte
D) float
Answer [=]
C

8) Which is the data type that is not recommended for numeric applications in Java?
A) byte
B) float
C) int
D) long
Answer [=]
A
Explanation:
Size of a byte is only 8 bits. Also, any arithmetic operation produces output in int, float or double. It cannot handle more data.

9) Choose the number range for byte data type in Java?

A) -127 to +128
B) -128 to +127
C) 0 to 256
D) 0 to 255
Answer [=]
B

10) What is the size of a SHORT integer in Java?
A) 1 byte
B) 2 bytes
C) 4 bytes
D) 8 bytes
Answer [=]

11) What is the size of an INT integer in Java?
A) 2 bytes
B) 4 bytes
C) 6 bytes
D) 8 bytes
Answer [=]
B
Explanation:
Number range is -2147483648 and 2147483647.

12) What is the size of a LONG integer in Java?
A) 2 bytes
B) 4 bytes
C) 8 bytes
D) 16 bytes
Answer [=]
C
Explanation:
Range: -9223372036854775808 and 9223372036854775807.

13) What is the size of a FLOAT floating point number in Java?
A) 2 bytes
B) 4 bytes
C) 6 bytes
D) 8 bytes
Answer [=]
B
Explanation:
Number range is ±3.40282347E+38F

14) What is the size of a DOUBLE floating point number in Java?
A) 4 bytes
B) 6 bytes
C) 8 bytes
D) 16 bytes
Answer [=]
C
Explanation:
Number range is ±1.7976931348623157E+308.

15) What is the size of a CHAR data type constant in Java?
A) 1 byte
B) 2 bytes
C) 4 bytes
D) 6 bytes
Answer [=]
B

Variables and Literals

1.

```
/**
 *  This program demonstrates
 *  how to use variables in a program
 */
public class VariableDemo
{
  public static void main(String[] args)
  {
    // Declare variables to hold data.
    int    rollno;
    String name;
    double marks;

    // Assign values to variables.
    rollno = 19;
    name   = "David";
    marks  = 89.8;

    // Display the message
    System.out.println("Your roll number is " + rollno);
    System.out.println("Your name is " + name);
    System.out.println("Your marks is " + marks);
  }
}
```

o/p

```
Your roll number is 19
Your name is David
Your marks is 89.8
```

2.

```
class Main {
  public static void main(String[] args) {

    double myDouble = 3.4;
    float myFloat = 3.4F;

    // 3.445*10^2
    double myDoubleScientific = 3.445e2;

    System.out.println(myDouble);  // prints 3.4
    System.out.println(myFloat);    // prints 3.4
```

```
    System.out.println(myDoubleScientific);   // prints 344.5
  }
}
```

MCQ's

1) What is Literal in Java?
A) Literal is the value that is given or assigned to a variable.
B) Literal is a data type
C) Literal is similar to String
D) None of the above
Answer [=]
A
Explanation:
Examples: 123, 45.67f, 'C', "abc", false

2) What are the types of Literals available in Java language?
A) Integer and Float
B) Character and String
C) Boolean
D) All the above
Answer [=]
D
Explanation:
Literals are Data assigned to Primitive data type variables.

3) What are the types of Integer Literals in Java?
A) Decimal Literals
B) Octal and Hexadecimal Literals
C) Binary Literals
D) All the above
Answer [=]
D
Explanation:
JDK 7 introduced binary literals to easily set individual bits of a number.

4) Choose correct examples of decimal literals in Java.
A)

```
int a = 12345;
```

B)

```
int a = 12_3__5;
```

C)

```
long a = 987____654_3__21L;
```

D) All the above
Answer [=]
D
Explanation:
To represent big numbers, simply append letter 'l' or 'L' to the number to make it a long integer. This avoids compiler errors saying "out of range"

5) An Octal number is Java is represented with a leading _____?
A) O (Alphabet)
B) 0 (ZERO)
C) 0x
D) 0X
Answer [=]
B
Explanation:

```
Eg. int a=0765;
```

6) Choose correct ranges for Decimal, Octal and Hexadecimal numbers in Java?
A) Decimal: 0 to 9
B) Octal: 0 to 7
C) Hexadecimal: 0 to 9 and A to F / a to f
D) All the above
Answer [=]
D

7) Choose the correct example of Octal Literal in Java?
A)

```
short = 0564;
```

B)

```
int = 076__45_2;
```

C)

```
int = 0_______11;
```

D) All the above
Answer [=]
D
Explanation:

```
int = 0_______11; // 8^1 * 1 + 8^0 * 1 = 9
```

8) What is the prefix used to represent Hexadecimal numbers in Java?
A) 0x
B) 0X
C) A and B

D) None of the above
Answer [=]
D
Explanation:

```
int a=0xFEB5;
int b=0X9876__45;
```

9) Choose correct examples of Hexadecimal literals in Java?
A)

```
long a = 0X987654321L;
```

B)

```
int a = 0x76FE____23;
```

C)

```
byte b = 0X0_____F;
```

D) All the above
Answer [=]
D
10) Binary literals in Java are introduced with which version of Java?
A) JDK 5
B) JDK 6
C) JDK 6
D) JDK 8
Answer [=]
C

Local (Stack, Automatic) Primitives and Objects

```
1. class Person
{
int id;
String name;
 public Person(int id, String name) { this.id = id; this.name = name;
}
}
public class PersonBuilder
{
private static Person buildPerson(int id, String name)
 {
return new Person(id, name);
 }
public static void main(String[] args)
 {
int id = 23;
 String name = "John";
Person person = null;
person = buildPerson(id, name);
}
}
```

MCQ's

What is the size of a boolean data type constant in Java?
A) 1 bit
B) 4 bits
C) 8 bits
D) Not documented well
Answer [=]
D
Explanation:

A boolean value can hold only one bit of information. But the size of a boolean constant in memory is not defined clearly anywhere. It is machine dependent. It may be 4 bytes or so.

17) What is the IEEE standard adopted to represent Floating point numbers in Java?
A) IEEE 9000
B) IEEE 800
C) IEEE 754
D) IEEE 512
Answer [=]
C
Explanation:
IEEE stands for Institute of Electrical and Electronics Engineers. Original specifications were defined in the year 1985. The current version includes improvements or corrections done in the year 2008.

18) What is the character encoding standard used in Java language?
A) ASCII
B) Unicode
C) Hexacode
D) Bytecode
Answer [=]
C
Explanation:
Unicode takes 2 Bytes of memory to represent all characters of all languages.

19) What is the abbreviation of ASCII?
A) American Standard Characters for Information Interchange
B) Australian Standard Code for Information Interchange
C) American Standard Code for Information Interchange
D) None of the above
Answer [=]
C
Explanation:
ASCII can represent only the English Alphabets and a few special symbols.

20) Choose the right statement about Java Data types.
A) Integer data types are short, int and long
B) Real number data types are float and double
C) The character data type is **char**.
D) All the above
Answer [=]
D

Passing Object Reference Variables

What are Pass-by-value and Pass-by-reference?

First, let's understand what are pass-by-value and pass-by-reference.

Pass-by-value:

A copy of the passed-in variable is copied into the argument of the method. Any changes to the argument do not affect the original one.

Pass-by-reference:

The argument is an alias of the passed-in variable. Any changes to the argument will affect the original one.

1.

```
/**
 * Impossible Swap function in Java
 * @author www.codejava.net
 */
public class Swap {
    public static void swap(int x, int y) {
        int temp = x;
        x = y;
        y = temp;
        System.out.println("x(1) = " + x);
        System.out.println("y(1) = " + y);
    }
    public static void main(String[] args) {
        int x = 10;

        int y = 20;
        swap(x, y);
        System.out.println("x(2) = " + x);
        System.out.println("y(2) = " + y);
    }
}
```

O/P

```
x(1) = 20
y(1) = 10
x(2) = 10
y(2) = 20
```

2.

```
// Java program to demonstrate reference

// varibale in java
```

```
import java.io.*;

class Demo {
	int x = 10;
	int display()
	{
		System.out.println("x = " + x);
		return 0;
	}
}

class Main {
	public static void main(String[] args)
	{
		Demo D1 = new Demo(); // point 1

		System.out.println(D1); // point 2

		System.out.println(D1.display()); // point 3
	}
}
```

O/P:

```
Demo@214c265e
x = 10
0
```

3.

```
import java.io.*;
class Demo {
	int x = 10;
```

```
        int display()
        {
                System.out.println("x = " + x);
                return 0;
        }
}

class Main {
        public static void main(String[] args)
        {
                // create instance
                Demo D1 = new Demo();

                // accessing instance(object) variable
                System.out.println(D1.x);

                // point 3
                // accessing instance(object) method
                D1.display();
        }
}
```

O/P:

```
10
x = 10
```

4.

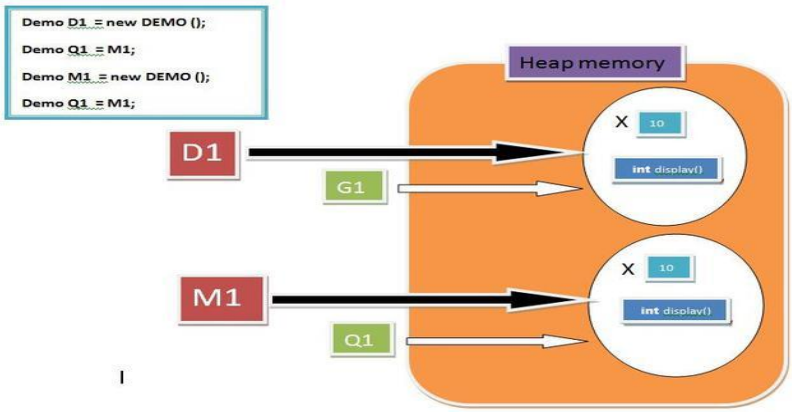
```
// Pointing to same instance memory
```

```
import java.io.*;
class Demo {
	int x = 10;
	int display()
	{
		System.out.println("x = " + x);
		return 0;
	}
}
class Main {
	public static void main(String[] args)
	{
		// create instance
		Demo D1 = new Demo();

		// point to same reference
		Demo G1 = D1;

		Demo M1 = new Demo();

		Demo Q1 = M1;

		// updating the value of x using G!
		// reference variable
		G1.x = 25;

		System.out.println(G1.x); // Point 1

		System.out.println(D1.x); // Point 2
	}
}
```

Output

25

25



5.

// Pass by reference and value

```
import java.io.*;
class Demo {
        int x = 10;
        int y = 20;

        int display(Demo A, Demo B)
        {
                // Updating value using argument
                A.x = 95;

                System.out.println("x = " + x);

                System.out.println("y = " + y);

                return 0;
        }
}
```

```
class Main {
        public static void main(String[] args)
        {
                Demo C = new Demo();

                Demo D = new Demo();

                // updating value using primary reference
                // variable
                D.y = 55;

                C.display(C, D); // POINT 1

                D.display(C, D); // POINT 2
        }
}
```

Output

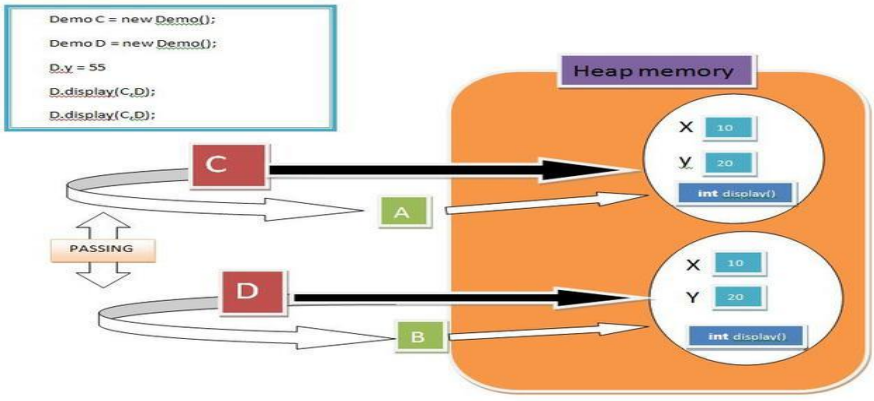
x = 95

y = 20

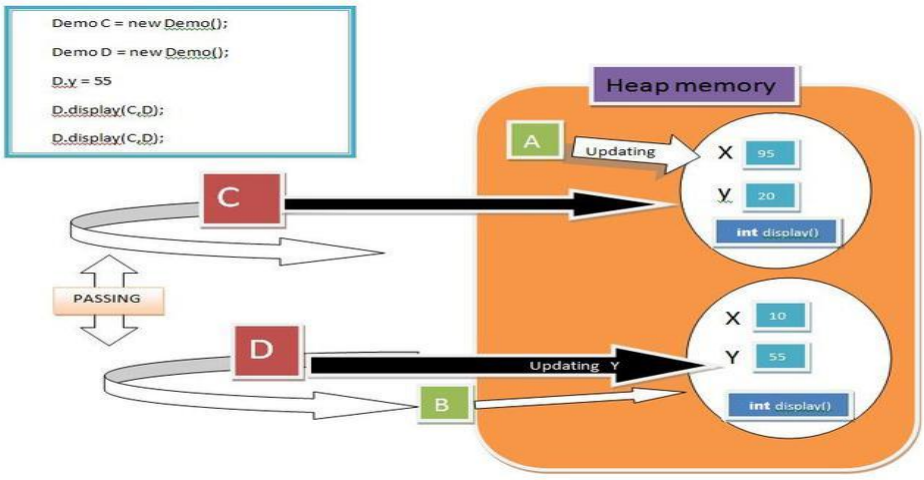
x = 10

y = 55

SCENE 1 :



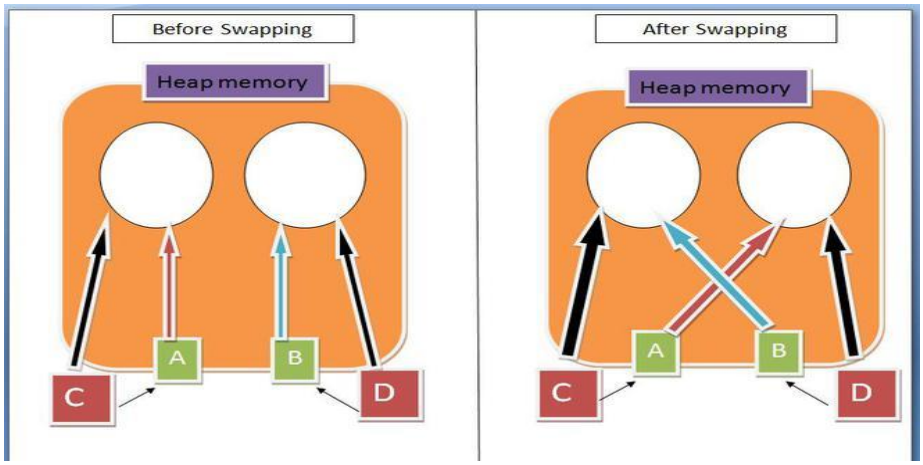
SCENE 2:



6.

```
// Swapping object references

import java.io.*;
class Demo {

        // Swapping Method
        int Swap(Demo A, Demo B)
        {
                Demo temp = A;
                A = B;
                B = temp;
                return 0;
        }
}
class Main {
        public static void main(String[] args)
```

```
        {
                Demo C = new Demo();

                Demo D = new Demo();

                // Passing C and reference variables
                // to Swap method
                C.Swap(C, D);
        }
}
```



MCQ

1) What is the output of the below Java program with two classes?

```
//Testing1.java
public class Example
{

}
public class Testing1
{
  public static void main(String[] args)
```

```
 {
   System.out.println("Hello Boss.!");
 }
}
```

A) Hello Boss.!
B) No Output
C) Compiler error
D) None of the above
Answer [=]
C
Explanation:
There can not be more than one public class declared inside a single java file.

2) What is the output of the below Java program?

```
//bingo.java file
public class Hello
{
  public static void main(String[] args)
  {
    System.out.println("BINGO");
  }
}
```

A) bingo
B) BINGO
C) Compiler error
D) None
Answer [=]
C
Explanation:
The class name and the java file name should be the same. So, change either file name or class name to match.

3) State TRUE or FALSE. A Java class provides encapsulation.
A) TRUE
B) FALSE
C) -
D) -
Answer [=]
A

4) What is the output of the below java class?

```
class Fox
```

```
{
  int legs = 2;
}
class Testing2
{
  public static void main(String[] args)
  {
    Fox t1 = new Fox();
    System.out.println("T1 before: " + t1.legs);
    t1.legs = 4;
    System.out.println("T1 After: " + t1.legs);
  }
}
```

A)

```
T1 before: 4
T1 After: 4
```

B)

```
T1 before: 2
T1 After: 2
```

C)

```
T1 before: 2
T1 After: 4
```

D) Compiler error
Answer [=]
C
Explanation:
There can be any number of classes in a single .java file.

5) The value of one primitive variable is assigned to another primitive variable by ___ in Java.
A) Pass by value
B) Pass by reference
C) -
D) -
Answer [=]

6) A primitive variable is passed from one method to another method by ___ in Java.
A) Pass by value
B) Pass by reference
C) -
D) -
Answer [=]

7) An object or primitive value that is passed from one method to another method is called ___ in Java. (Argument / Parameter)
A) Argument
B) Parameter
C) -
D) -
Answer [=]
B

8) An object or a primitive value that is received in a method from another method is called ___ in Java. (Argument / Parameter)
A) Argument
B) Parameter
C) -
D) -
Answer [=]
A

9) What is the output of the below Java program that passes an object to another method?

```
class Food
{
  int items;
  int show()
  {return items;}
}

class Testing9
{
  public static void main(String[] args)
  {
    Food f = new Food();
    f.items = 5;
    System.out.println("Items Before = " + f.show());
```

```
    change(f);
    System.out.println("Items After = " + f.show());
  }
  static void change(Food foo)
  { foo.items = 10; }
}
```

A)

```
Items Before = 10
Items After = 10
```

B)

```
Items Before = 5
Items After = 5
```

C)

```
Items Before = 5
Items After = 10
```

D)

```
Items Before = 10
Items After = 5
```

Answer [=]
C

10) What is the output of the below Java program that passes primitive values?

```
class Testing10
{
  int rats = 5;

  public static void main(String[] args)
  {
    Testing10 t1 = new Testing10();
    System.out.println("Rats Before = " + t1.rats);
    modify(t1.rats);
```

```
  System.out.println("Rats After = " + t1.rats);
 }
 static void modify(int r)
 { r = 20; }
}
```

A)

```
Rats Before = 5
Rats After = 5
```

B)

```
Rats Before = 20
Rats After = 20
```

C)

```
Rats Before = 5
Rats After = 20
```

D)

```
Rats Before = 20
Rats After = 5
```

Answer [=]
A
Explanation:
The primitive values are passed by value only. So, changes in the method modify does not change the original value.

11) Java object assignment happens by ___.
A) Pass by Value
B) Pass by Reference
C) -
D) -
Answer [=]
B
Explanation:
Yes. That is the reason why you can change the values of variables of the object using another reference.

12) Java object passing from one method to another method happens by ___.
A) Pass by Value
B) Pass by Reference
C) -

D) -
Answer [=]
B
Explanation:
References point to the original objects. So they can change the state of the objects.

13) In Java Pass by reference ___ is passed even if you are passing a reference to an object.
A) Address value
B) Variable value
C) Hash code
D) None of the above
Answer [=]
A
Explanation:
Yes. The address is passed automatically by Java. So, Java pundits argue that it is passing a value (Address).

14) A Java reference is comparable to ___ in C language.
A) Enum
B) Structure
C) Pointer
D) None
Answer [=]
C

15) ___ is the superclass to all Java classes either user-defined or built-in.
A) Class
B) Object
C) Superclass
D) Null
Answer [=]
B
Explanation:
Yes. java.lang.Object is the superclass to all Java classes.

1.Let's look at this in practice and assume that we have the following Person class available to us.

```
public class Person {
    private String name;
    private int birthYear;

    public Person(String name) {
        this.name = name;
        this.birthYear = 1970;
    }

    public int getBirthYear() {
        return this.birthYear;
    }

    public void setBirthYear(int birthYear) {
        this.birthYear = birthYear;
    }

    public String toString() {
        return this.name + " (" + this.birthYear + ")";
    }
}
```

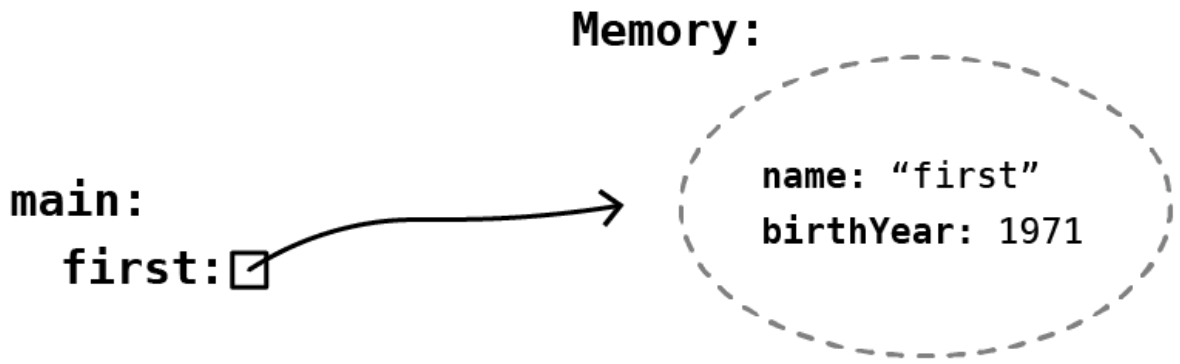
We'll inspect the execution of the program step by step.

```
public class Example {
    public static void main(String[] args) {
        Person first = new Person("First");

        System.out.println(first);
        youthen(first);
        System.out.println(first);

        Person second = first;
        youthen(second);

        System.out.println(first);
```

```
    }

    public static void youthen(Person person) {
        person.setBirthYear(person.getBirthYear() + 1);
    }
}
```

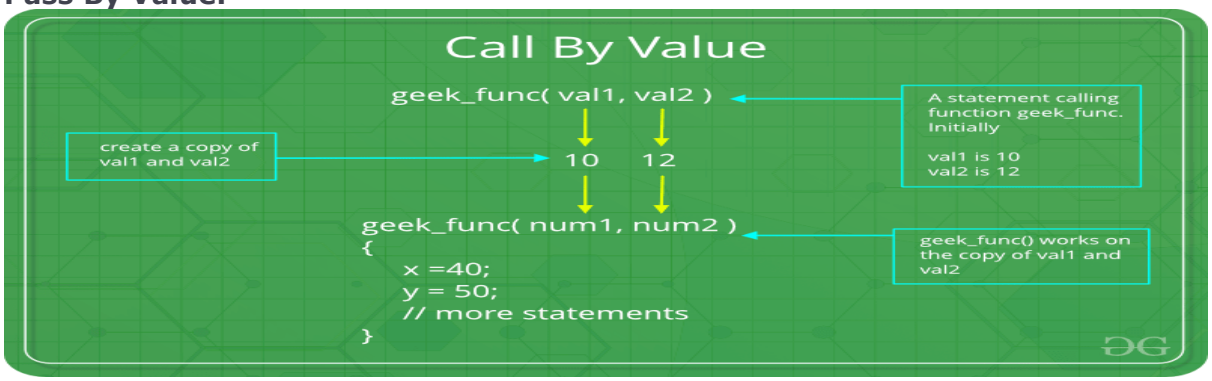
Sample output

```
First (1970) First (1971) First (1972)
```



Passing Variables into Methods

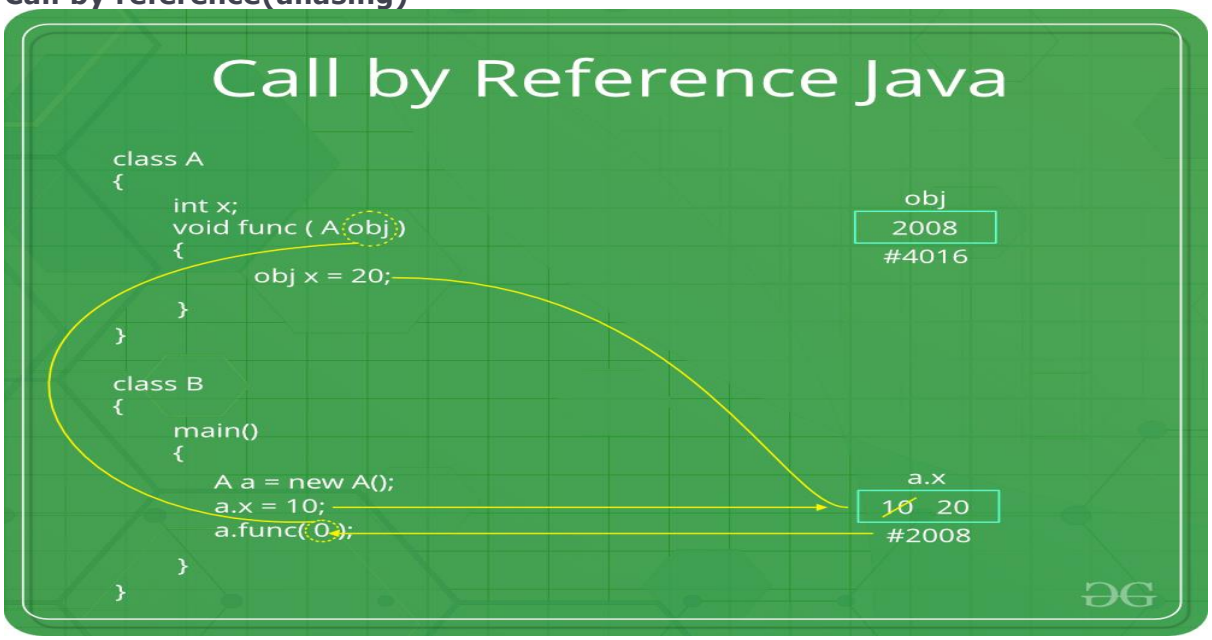
Pass By Value:




```
// Java program to illustrate
// Call by Value

// Callee
class CallByValue {

	// Function to change the value
	// of the parameters
	public static void Example(int x, int y)
	{
		x++;
		y++;
	}
}

// Caller
public class Main {
	public static void main(String[] args)
	{

		int a = 10;
		int b = 20;

		// Instance of class is created
		CallByValue object = new CallByValue();

		System.out.println("Value of a: " + a
						+ " & b: " + b);

		// Passing variables in the class function
		object.Example(a, b);

		// Displaying values after
		// calling the function
		System.out.println("Value of a: "
						+ a + " & b: " + b);
	}
}
```

Output:
Value of a: 10 & b: 20
Value of a: 10 & b: 20

Call by reference(aliasing)




```
// Java program to illustrate
// Call by Reference

// Callee
class CallByReference {

        int a, b;

        // Function to assign the value
        // to the class variables
        CallByReference(int x, int y)
        {
                a = x;
                b = y;
        }

        // Changing the values of class variables
        void ChangeValue(CallByReference obj)
        {
                obj.a += 10;
                obj.b += 20;
        }
}

// Caller
public class Main {
```

```
        public static void main(String[] args)
        {

                // Instance of class is created
                // and value is assigned using constructor
                CallByReference object
                        = new CallByReference(10, 20);

                System.out.println("Value of a: "
                                                + object.a
                                                + " & b: "
                                                + object.b);

                // Changing values in class function
                object.ChangeValue(object);

                // Displaying values
                // after calling the function
                System.out.println("Value of a: "
                                                + object.a
                                                + " & b: "
                                                + object.b);
        }
}
```

Output:

Value of a: 10 & b: 20
Value of a: 20 & b: 40

3. Parameters and Arguments

```
public class Main {
  static void myMethod(String fname) {
    System.out.println(fname + " Refsnes");
  }

  public static void main(String[] args) {
    myMethod("Liam");
    myMethod("Jenny");
    myMethod("Anja");
  }
}
```

O/P:

Liam Refsnes
Jenny Refsnes
Anja Refsnes

4. Multiple Parameters

```
public class Main {
  static void myMethod(String fname) {
    System.out.println(fname + " Refsnes");
  }

  public static void main(String[] args) {
    myMethod("Liam");
    myMethod("Jenny");
    myMethod("Anja");
  }
}
```

O/P:

```
Liam Refsnes
Jenny Refsnes
Anja Refsnes
```

MCQ's

1) What is the output of the below Java program with two classes?

```
//Testing1.java
public class Example
{

}
public class Testing1
```

```
{
  public static void main(String[] args)
  {
    System.out.println("Hello Boss.!");
  }
}
```

A) Hello Boss.!
B) No Output
C) Compiler error
D) None of the above
Answer [=]
C
Explanation:
There can not be more than one public class declared inside a single java file.

2) What is the output of the below Java program?

```
//bingo.java file
public class Hello
{
  public static void main(String[] args)
  {
    System.out.println("BINGO");
  }
}
```

A) bingo
B) BINGO
C) Compiler error
D) None
Answer [=]
C
Explanation:
The class name and the java file name should be the same. So, change either file name or class name to match.

3) State TRUE or FALSE. A Java class provides encapsulation.
A) TRUE
B) FALSE
C) -
D) -
Answer [=]
A

4) What is the output of the below java class?

```
class Fox
{
  int legs = 2;
}
class Testing2
{
  public static void main(String[] args)
  {
    Fox t1 = new Fox();
    System.out.println("T1 before: " + t1.legs);
    t1.legs = 4;
    System.out.println("T1 After: " + t1.legs);
  }
}
```

A)

```
T1 before: 4
T1 After: 4
```

B)

```
T1 before: 2
T1 After: 2
```

C)

```
T1 before: 2
T1 After: 4
```

D) Compiler error
Answer [=]
C
Explanation:
There can be any number of classes in a single .java file.

5) The value of one primitive variable is assigned to another primitive variable by ___ in Java.
A) Pass by value
B) Pass by reference

C) -
D) -
Answer [=]
A

6) A primitive variable is passed from one method to another method by ____ in Java.
A) Pass by value
B) Pass by reference
C) -
D) -
Answer [=]
A

7) An object or primitive value that is passed from one method to another method is called ____ in Java. (Argument / Parameter)
A) Argument
B) Parameter
C) -
D) -
Answer [=]
B

8) An object or a primitive value that is received in a method from another method is called ____ in Java. (Argument / Parameter)
A) Argument
B) Parameter
C) -
D) -
Answer [=]
A

9) What is the output of the below Java program that passes an object to another method?

```
class Food
{
  int items;
  int show()
  {return items;}
}

class Testing9
{
  public static void main(String[] args)
  {
    Food f = new Food();
    f.items = 5;
```

```
  System.out.println("Items Before = " + f.show());
  change(f);
  System.out.println("Items After = " + f.show());
 }
 static void change(Food foo)
 { foo.items = 10; }
}
```

A)

```
Items Before = 10
Items After = 10
```

B)

```
Items Before = 5
Items After = 5
```

C)

```
Items Before = 5
Items After = 10
```

D)

```
Items Before = 10
Items After = 5
```

Answer [=]
C

10) What is the output of the below Java program that passes primitive values?

```
class Testing10
{
 int rats = 5;

 public static void main(String[] args)
 {
  Testing10 t1 = new Testing10();
  System.out.println("Rats Before = " + t1.rats);
```

```
    modify(t1.rats);
    System.out.println("Rats After = " + t1.rats);
  }
  static void modify(int r)
  { r = 20; }
}
```

A)

```
Rats Before = 5
Rats After = 5
```

B)

```
Rats Before = 20
Rats After = 20
```

C)

```
Rats Before = 5
Rats After = 20
```

D)

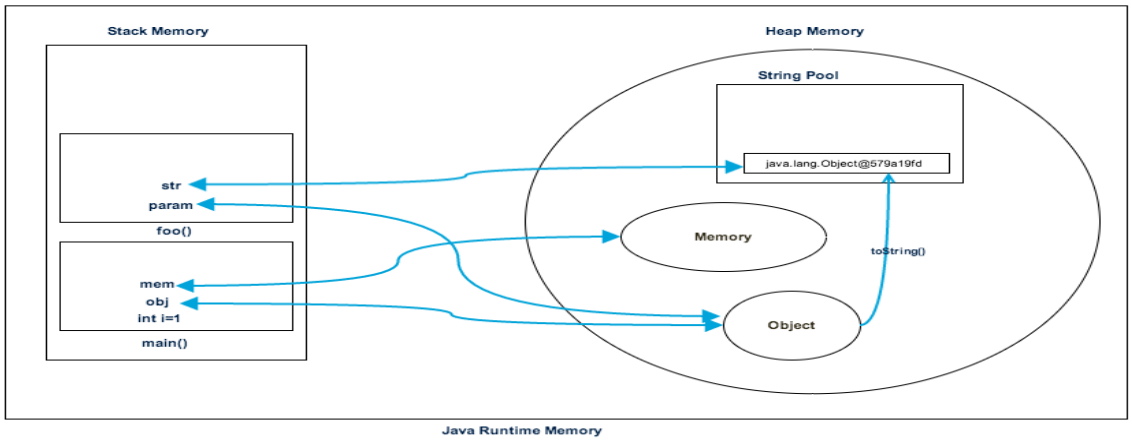
```
Rats Before = 20
Rats After = 5
```

Answer [=]
A
Explanation:
The primitive values are passed by value only. So, changes in the method modify does not change the original value.

Stack and Heap

1.

```
package com.journaldev.test;


public class Memory {


        public static void main(String[] args) { // Line 1
                int i=1; // Line 2
                Object obj = new Object(); // Line 3
                Memory mem = new Memory(); // Line 4
                mem.foo(obj); // Line 5
        } // Line 9


        private void foo(Object param) { // Line 6
                String str = param.toString(); //// Line 7
                System.out.println(str);
        } // Line 8


}
```



2. package java.memoryallocation;

```
// As soon as this program run it loads the runtime classes into the Heap space.
public class Employee {

    // The main() found creates a Stack memory that will be used by the main() method thread.
    public static void main(String[] args) {
        // This local variable is created and stored in the stack memory of the main() method.
        int index = 1;

        // This object creation is created in the Heap space and the Stack memory contains a reference
for it.
        Object myObject = new Object();

        // This object creation is created in the Heap space and the Stack memory contains a reference
for it.
        Employee myEmployee = new Employee();

        // Now calling the "someMethod()", a block in the top of the Stack memory is created and is used
by
        // the "someMethod()" method. Since Java is pass-by-value in nature, a new reference to Object
is created
        // in the "someMethod" stack block.
        myEmployee.someMethod(myObject);

    }   // At this point the "main()" terminates as it has done it job and the Stack space created for
        // "main()" method is destroyed. Also, the program ends and hence the JRE frees all the memory
        // and ends the program execution.

    private void someMethod(Object object) {
        // The string created goes to the "String Pool" that residers in the heap space and the reference of
it
        // is created in the "someMethod()" stack space.
        String name = object.toString();
        System.out.println("Name= " + name);

    }   // At this point the "someMethod()" terminates and the memory block allocated for
"someMethod()"
        // in the Stack space becomes free.
}
```

MCQ's

1. Which of the following has the highest memory requirement?
   a) Heap
   b) Stack
   c) JVM
   d) Class
   Answer:c
   Explanation: JVM is the super set which contains heap, stack, objects, pointers, etc.

2. Where is a new object allocated memory?
   a)Youngspace
   b)Oldspace
   c)Young or Old space depending on space availability
   d) JVM
   Answer:a
   Explanation: A new object is always created in young space. Once young space is full, a special young collection is run where objects which have lived long enough are moved to old space and memory is freed up in young space for new objects.

3. Which of the following is a garbage collection technique?
   a)Cleanupmodel
   b)Markandsweepmodel
   c)Spacemanagementmodel
   d) Sweep model
   Answer:b
   Explanation: A mark and sweep garbage collection consists of two phases, the mark phase and the sweep phase. I mark phase all the objects reachable by java threads, native handles and other root sources are marked alive and others are garbage. In sweep phase, the heap is traversed to find gaps between live objects and the gaps are marked free list used for allocating memory to new objects.

4. What is -Xms and -Xmx while starting jvm?
   a)Initial;Maximummemory
   b)Maximum;Initialmemory
   c)Maximummemory
   d) Initial memory
   Answer:a
   Explanation: JVM will be started with Xms amount of memory and will be able to use a maximum of Xmx amount of memory. java -Xmx2048m -Xms256m.

5. Which exception is thrown when java is out of memory?
   a)MemoryFullException
   b)MemoryOutOfBoundsException

c)OutOfMemoryError
d) MemoryError

Answer:c
Explanation: The Xms flag has no default value, and Xmx typically has a default value of 256MB. A common use for these flags is when you encounter a java.lang.OutOfMemoryError.

5. How to get prints of shared object memory maps or heap memory maps for a given process?
   a)jmap
   b)memorymap
   c)memorypath
   d) jvmmap
   Answer:a
   Explanation: We can use jmap as jmap -J-d64 -heap pid.
6. What happens to the thread when garbage collection kicks off?
   a)Thethread continues its operation
   b) Garbage collection cannot happen until the thread is running
   c) The thread is paused while garbage collection runs
   d) The thread and garbage collection do not interfere with each other

Answer:c
Explanation: The thread is paused when garbage collection runs which slows the application performance.

8.Whichof the below is not a Java Profiler?
a)JVM
b)JConsole
c)JProfiler
d) Eclipse Profiler

Answer:a
Explanation: Memory leak is like holding a strong reference to an object although it would never be needed anymore. Objects that are reachable but not live are considered memory leaks. Various tools help us to identify memory leaks.

9. Which of the below is not a memory leak solution?
a)Codechanges
b)JVMparametertuning
c)Processrestart
d) GC parameter tuning

Answer:c
Explanation: Process restart is not a permanent fix to memory leak problem. The problem will resurge again.

10.Garbage Collection can be controlled by a program?
a)True
b) False

Answer:b
Explanation: Garbage Collection cannot be controlled by a program.

Using a Variable or Array Element That Is Uninitialized and Unassigned

1.Given the following code what will element *b[5]* contain?

```
public class MyVal{
public static void main(String argv[]){
	MyVal m = new MyVal();
	m.amethod();
}


public void amethod(){
boolean b[] = new boolean[5];
	}
```

```
}
```

1) 0
2) null
3) ""
4) none of these options

Ans:

4) none of these options

Sneaky one here. Array element numbering starts at 0, therefore there is no element 5 for this array. If you were to attempt to perform

System.out.println(b[5])

You would get an exception.

2)

Given the following constructor what will element 1 of *mycon* contain?

```
MyCon(){
      int[] mycon= new int[5];
}
```

1) 0
2) null
3) ""
4) None of these options
Ans:

1) 0

A constructor acts no different to any other method for this purpose and an array of integers will be initialised to contain zeros wherever it is created.

3)

What will happen when you attempt to compile and run the following code?

```
public class MyField{
int i=99;
public static void main(String argv[]){
      MyField m = new MyField();
      m.amethod();
      }
```

```
void amethod(){
        int i;
        System.out.println(i);
        }

}
```

1) The value 99 will be output
2) The value 0 will be output
3) Compile time error
4) Run time error

Ans

3) Compile time error

You will get a compile time error indicating that variable *i* may not have been initialised. The classs level variable *i* is a red herring, as it will be shadowed by the method level version. Method level variables do not get any default initialisation.

4)

What will happen when you attempt to compile and run the following code?

```
public class MyField{
String s;
public static void main(String argv[]){
        MyField m = new MyField();
        m.amethod();
        }
void amethod(){
        System.out.println(s);
        }
}
```

1) Compile time error s has not been initialised
2) Runtime error s has not been initialised
3) Blank output
4) Output of null

Ans

4) Output of null

A variable created at class level will always be given a default value. The default value of an object reference is *null* and the *toString* method implicitly called via System.out.println will output *null*

```
5. // Java program to demonstrate default values of array
// elements
class ArrayDemo
{
	public static void main(String[] args)
	{
		System.out.println("String array default values:");
		String str[] = new String[5];
		for (String s : str)
			System.out.print(s + " ");

		System.out.println("\n\nInteger array default values:");
		int num[] = new int[5];
		for (int val : num)
			System.out.print(val + " ");

		System.out.println("\n\nDouble array default values:");
		double dnum[] = new double[5];
		for (double val : dnum)
			System.out.print(val + " ");

		System.out.println("\n\nBoolean array default values:");
		boolean bnum[] = new boolean[5];
		for (boolean val : bnum)
```

```
                System.out.print(val + " ");


        System.out.println("\n\nReference Array default values:");
        ArrayDemo ademo[] = new ArrayDemo[5];
        for (ArrayDemo val : ademo)
                System.out.print(val + " ");
    }
}
```

Output:

String array default values:

null null null null null

Integer array default values:

0 0 0 0 0

Double array default values:

0.0 0.0 0.0 0.0 0.0

Boolean array default values:

false false false false false

Reference Array default values:

null null null null null

MCQ's

1) An Array in Java is a collection of elements of ___ data type.
A) Same
B) Different
C) -
D) -
Answer [=]
A

2) The Java Virtual Machine (JVM) implements arrays as ___ type.
A) Primitive
B) Object
C) -
D) -
Answer [=]
B
Explanation:
That is the reason why Java Array has predefined methods.

3) Unlike C-Arrays, the Java-Arrays have ___.
A) Names
B) Values
C) Methods and Fields
D) None
Answer [=]
C

4) An array declaration in Java without initialization ___ memory.
A) Does not allocate
B) Allocates memory
C) -
D) -
Answer [=]
A
Explanation:
Only initialization causes memory to be allocated.

5) In Java language, an array index starts with ___.
A) -1
B) 0
C) 1
D) Any integer
Answer [=]
B

6) Which are the special symbols used to declare an array in Java?
A) Braces { }
B) Parentheses ()
C) Square Brackets [ ]
D) Angled Brackets < >
Answer [=]
C

7) Which are the special symbols used to initialize an array at the time of the declaration itself?
A) Parentheses ( )
B) Square Brackets [ ]
C) Braces { }
D) Angled Brackets < >
Answer [=]
C
Explanation:

```
int[] nums = {1,3,6};
```

8) It is possible to skip initializing some elements of the array during Shorthand Initialization. (TRUE / FALSE)
A) FALSE
B) TRUE
C) -
D) -
Answer [=]
A
Explanation:
No, you can not skip any elements. All elements need to be initialized in one go or at the same time.

9) In Java, an array can be declared without initialization without mentioning the size. (TRUE / FALSE)
A) TRUE
B) FALSE
C) -
D) -
Answer [=]
A
Explanation:
It is a Lazy initialization of an array.

10) What is the output of the below Java code snippet with arrays?

```
static int[] nums;

public static void main(String args[])

{

  System.out.println(nums.length);

}
```

A) 0
B) null
C) Compiler error
D) Runtime Exception - Null Pointer Exception
Answer [=]

D

Wrapper Class

Using Wrapper Classes and Boxing

- o An Overview of the Wrapper Classes
- o Creating Wrapper Objects
- Using Wrapper Conversion Utilities
- Autoboxing

Wrapper

A Wrapper class is a class whose object wraps or contains primitive data types. When we create an object to a wrapper class, it contains a field and in this field, we can store primitive data types. In other words, we can wrap a primitive value into a wrapper class object.

Need of Wrapper Classes

1. They convert primitive data types into objects. Objects are needed if we wish to modify the arguments passed into a method (because primitive types are passed by value).

2. The classes in java.util package handles only objects and hence wrapper classes help in this case also.
3. Data structures in the Collection framework, such as ArrayList and Vector, store only objects (reference types) and not primitive types.
4. An object is needed to support synchronization in multithreading.

Primitive Data types and their Corresponding Wrapper class

Primitive Data Type	Wrapper Class
char	Character
byte	Byte
short	Short
int	Integer
long	Long
float	Float
double	Double
boolean	Boolean

Autoboxing: Automatic conversion of primitive types to the object of their corresponding wrapper classes is known as autoboxing. For example – conversion of int to Integer, long to Long, double to Double etc.

Example:

```
// Java program to demonstrate Autoboxing

import java.util.ArrayList;
class Autoboxing
{
	public static void main(String[] args)
	{
		char ch = 'a';

		// Autoboxing- primitive to Character object conversion
		Character a = ch;

		ArrayList<Integer> arrayList = new ArrayList<Integer>();

		// Autoboxing because ArrayList stores only objects
		arrayList.add(25);

		// printing the values from object
		System.out.println(arrayList.get(0));
	}
}
```

Output:

```
25
```

Unboxing: It is just the reverse process of autoboxing. Automatically converting an object of a wrapper class to its corresponding primitive type is known as unboxing. For example – conversion of Integer to int, Long to long, Double to double, etc.

```
// Java program to demonstrate Unboxing
import java.util.ArrayList;
```

```
class Unboxing
{
	public static void main(String[] args)
	{
		Character ch = 'a';

		// unboxing - Character object to primitive conversion
		char a = ch;

		ArrayList<Integer> arrayList = new ArrayList<Integer>();
		arrayList.add(24);

		// unboxing because get method returns an Integer object
		int num = arrayList.get(0);

		// printing the values from primitive data types
		System.out.println(num);
	}
}
```

Output:

```
24
```

```
// Java program to demonstrate Wrapping and UnWrapping
// in Java Classes
class WrappingUnwrapping
{
	public static void main(String args[])
	{
		// byte data type
		byte a = 1;

		// wrapping around Byte object
		Byte byteobj = new Byte(a);

		// int data type
		int b = 10;

		//wrapping around Integer object
		Integer intobj = new Integer(b);

		// float data type
		float c = 18.6f;

		// wrapping around Float object
		Float floatobj = new Float(c);

		// double data type
		double d = 250.5;

		// Wrapping around Double object
		Double doubleobj = new Double(d);
```

```
        // char data type
        char e='a';

        // wrapping around Character object
        Character charobj=e;

        // printing the values from objects
        System.out.println("Values of Wrapper objects (printing as objects)");
        System.out.println("Byte object byteobj: " + byteobj);
        System.out.println("Integer object intobj: " + intobj);
        System.out.println("Float object floatobj: " + floatobj);
        System.out.println("Double object doubleobj: " + doubleobj);
        System.out.println("Character object charobj: " + charobj);

        // objects to data types (retrieving data types from objects)
        // unwrapping objects to primitive data types
        byte bv = byteobj;
        int iv = intobj;
        float fv = floatobj;
        double dv = doubleobj;
        char cv = charobj;

        // printing the values from data types
        System.out.println("Unwrapped values (printing as data types)");
        System.out.println("byte value, bv: " + bv);
        System.out.println("int value, iv: " + iv);
        System.out.println("float value, fv: " + fv);
        System.out.println("double value, dv: " + dv);
        System.out.println("char value, cv: " + cv);
    }
}
```

Output:

```
Values of Wrapper objects (printing as objects)
Byte object byteobj:  1
Integer object intobj:  10
Float object floatobj:  18.6
Double object doubleobj:  250.5
Character object charobj: a
Unwrapped values (printing as data types)
byte value, bv: 1
int value, iv: 10
float value, fv: 18.6
double value, dv: 250.5
char value, cv: a
```

MCQs

1. Which of these is a wrapper for data type int?
   a) Integer
   b) Long

c) Byte
d) Double

Answer: a
Explanation: None.

2. Which of the following methods is a method of wrapper Integer for obtaining hash code for the invoking object?
   a) int hash()
   b) int hashcode()
   c) int hashCode()
   d) Integer hashcode()

Answer: c
Explanation: None.

3. Which of these is a super class of wrappers Long, Character & Integer?
   a) Long
   b) Digits
   c) Float
   d) Number
   Answer:d
   Explanation: Number is an abstract class containing subclasses Double, Float, Byte, Short, Integer and Long.
   4. Which of these is a wrapper for simple data type char?
   a)Float
   b)Character
   c)String
   d) Integer

Answer:b
Explanation: None.

5. Which of the following is method of wrapper Integer for converting the value of an objectintoint?
   a)bytevalue()
   b)intintValue();
   c)Bytevalue()
   d) Byte Bytevalue()
   Answer: b
   Explanation: None.