Java Interview Questions

# FUNDAMENTALS OF JAVA

1. Which of the following can be operands of arithmetic operators?  
   a) Numeric  
   b) Boolean  
   c) Characters  
   d) Both Numeric & Characters

Ans:  d

**Explanation :The operand of arithmetic operators can be any of numeric or character type, But not boolean.**

1. Modulus operator, %, can be applied to which of these?  
   a) Integers  
   b) Floating – point numbers  
   c) Both Integers and floating – point numbers  
   d) None of the mentioned

Ans :c

**Explanation : Modulus operator can be applied to both integers and floating point numbers.**

1. Decrement operator, −−, decreases the value of variable by what number?  
   a) 1  
   b) 2  
   c) 3  
   d) 4

Ans : a

**Explanation: Decrement operator, âˆ’âˆ’, decreases the value of variable by 1.**

1. Which of these statements are incorrect?  
   a) Assignment operators are more efficiently implemented by Java run-time system than their equivalent long forms  
   b) Assignment operators run faster than their equivalent long forms  
   c) Assignment operators can be used only with numeric and character data type  
   d) None of the mentioned

Ans :d

**Explanation: None of the mentioned.**

1. What will be the output of the following Java program?

**class** increment

   {

**public** **static** **void** main(String args[])

        {

**double** var1 = 1 + 5;

**double** var2 = var1 / 4;

**int** var3 = 1 + 5;

**int** var4 = var3 / 4;

            System.out.print(var2 + " " + var4);

        }

    }

a) 1 1  
b) 0 1  
c) 1.5 1  
d) 1.5 1.0

Ans : C

1. What will be the output of the following Java program?

**class** increment

    {

**public** **static** **void** main(String args[])

        {

**int** g = 3;

             System.out.print(++g \* 8);

        }

    }

a) 25  
b) 24  
c) 32  
d) 33

Ans: C

1. What are the primitive data types in Java ?

There are eight primitive data types.

* byte.
* short.
* int.
* long.
* float.
* double.
* boolean.
* char.

1. **Difference between double and float variables in Java.**

In java, float takes 4 bytes in memory while Double takes 8 bytes in memory. Float is single precision floating point decimal number while Double is double precision decimal number.

1. **Can you compare a boolean with an int variable in Java?**

No. you will get compilation error.

1. **What are primitive Data Types**

The primitive data types include byte, short, int, long, float, double, boolean, and char. The primitive data type specifies the size and type of variable values.

byte

A Byte data type is used to save space in larger arrays in the place of integers. A byte is four times smaller than an integer. The default value of a byte is 0.

byte b = 90 ;

short

A short data type can be used to save memory similar to a byte data type. A short is two times smaller than an integer. The default value of a short is 0.

short s = 1200;

int

An integer is the conventionally used default data type for values unless there is a concern about memory. The default value of an integer is 0.

int var = 5;

long

A long is a type used when a wider range than an int is needed. The default value of a long is 0L.

long l = 1000l;

float

A float is mainly used to save memory in large data structures of floating point numbers or to represent a collection of floating point numbers. The default value of a float is 0.0f.

float f = 3.14f

boolean

A boolean data type is used for flags that examine true or false conditions. There are only two possible values: true and false. The default value is false.

boolean var = true;

double

A double is generally used as the default data type for decimal values. It can be used to represent both floating point and decimal numbers. The default value is 0.0d.

double d = 1000.2;

char

A char is used to store any character.

char ch= 'A';

1. Explain the concept of loops in java :

The basic reason for using loops is when a repetitive action needs to be taken over and over. Usually, this will be for a set of data. It’s better and more efficient to use loops than having to explicitly write the code to process each action. And sometimes the programmer won’t know in advance how many pieces of data the program will encounter so looping will be necessary.

All loops in Java use the same building blocks that we’re going to define here.

* loop initialization – setting the initial value of variable(s) to be used in the loop
* boolean condition – this is a boolean expression that determines whether to continue looping for one more iteration
* step value (update value) – an update to the loop variable(s)
* loop body – execute the main part of the loop that does the processing

**For Loop**

For loops are best used when you know how many times you have to loop over something. For example, when looping over an array of numbers you will need to loop over as many times as there are elements in the array.

For Loop Structure

Java for loops are structured to follow this order of execution:

**1) loop initialization  
2) boolean condition – if true, continue to next step; if false, exit loop  
3) loop body  
4) step value  
5) repeat from step 2 (boolean condition)**

Example – Incrementing Step Value

Here is a simple for loop incrementing values from 0 to 99.\

Notice the settings for the loop:

* loop index is initialized to 0
* boolean condition tests that the loop index is less than 100
* step value increments the loop index by 1 after every loop
* loop exits when i gets to 100 and the boolean condition evaluates to false

**example: print 10 values using for loop statement :**

for (int i-0;i<=10;i++)

{

System.out.println(i);

}

1. Explain if condition using java

The **Java if statement** is the most simple decision-making statement. It is used to decide whether a certain statement or block of statements will be executed or not i.e if a certain condition is true then a block of statement is executed otherwise not

**Syntax:**

if(condition)

{

   // Statements to execute if

   // condition is true

}

1. Control falls into the if block.
2. The flow jumps to Condition.
3. Condition is tested.
   1. If Condition yields true, goto Step 4.
   2. If Condition yields false, goto Step 5.
4. The if-block or the body inside the if is executed.
5. Flow steps out of the if block.

**Example 1:**

* Java

|  |
| --- |
| // Java program to illustrate If statement    **class** IfDemo {  **public** **static** **void** main(String args[])      {  **int** i = 10;    **if** (i < 15)              System.out.println("10 is less than 15");            System.out.println("Outside if-block");          // both statements will be printed      }  } |

1. **What is constructors in java**

**Constructor** is just like a method in [**Java**](https://javainterviewpoint.com/category/core-java/) that is used to initialize the state of an object and will be invoked during the time of object creation.

1. **What are the Rules for defining a constructor?**
2. Constructor **name** should be the same as the class name
3. It **cannot** contain any **return type**
4. It **can** have all **Access Modifiers** are allowed (private , public, protected, default)
5. It **Cannot** have any **Non Access Modifiers** (final ,static, abstract, synchronized)
6. **What is a No-arg constructor?**

Constructor **without arguments** is called no-arg constructor. In Java Default constructor is a no-arg constructor.

class Demo

{

     public Demo()

     {

        //No-arg constructor

     }

}

* **No return** statement is allowed
* It **can** take any number of **parameters**
* Constructor can **throw exception**, we can have**throws clause**

1. **Why constructors cannot be abstract in Java?**

When you set a method as abstract, then “The method doesn’t or cannot have body”. A constructor will be automatically called when object is created. It cannot lack a body moreover an abstract constructor could never be implemented.

|  |  |  |  |
| --- | --- | --- | --- |
| |  |  | | --- | --- | | 1. **Why every object constructor automatically call super() in Object before its own constructors?** | [Core Java](https://javasearch.buggybread.com/InterviewQuestions/questionSearch.php?searchOption=label&keyword=Core%20Java) | | |
| Ans. Derived object carries the body of its class as well as the body of the parent class. Its body ( member elements ) is initialized using its own class constructor whereas the body ( member elements ) carried from the parent class are initialized using super class constructor. So In order to initialize the elements of the parent class before its own elements are even initialized, super is called. | |
| |  |  | | --- | --- | | 1. **Are constructors inherited? Can a subclass call the parent's class constructor? When?** | [Core Java](https://javasearch.buggybread.com/InterviewQuestions/questionSearch.php?searchOption=label&keyword=Core%20Java) | |  |  | |  |
| Ans. You cannot inherit a constructor. That is, you cannot create a instance of a subclass using a constructor of one of it's superclasses. One of the main reasons is because you probably don't want to override the superclasses constructor, which would be possible if they were inherited. By giving the developer the ability to override a superclasses constructor you would erode the encapsulation abilities of the language. |  |

# JAVA STRINGS

**Explain  STRING FUNCTIONS/METHODS in java :**

* A number of methods provided in Java to perform operations in Strings are called String functions. A string is an object that represents a sequence of characters or char values. The java. lang. A string class is used to create a Java string object. There are so many methods available which we will discuss in this blog. Strings in Java are constant, and it is created either using a literal or using a keyword. String literal is used to make Java memory efficient, and the keyword creates a Java string in normal memory. The string represents an array of character values, and the class is implemented by three interfaces Serializable, Comparable, and Char Sequence interfaces. It represents the sequence of characters in a serialized or comparable manner.

**1. Create String**

**2. String length**

**3. Concatenating  of strings**

**4. Creating a format string**

**1) how to Create a  String:**

There are two ways in which a String object can be created in Java:

Using a string literal: A string literal in Java can be created using double quotes. Ex. String s= "Hello NumPy Ninja!";

Using the new keyword: Java String can be created by using the keyword “new”. Ex. String s= new String("Hello NumPy Ninja!");

**2. Find the String length**:

Methods that are u1sed to get information about an object are called accessor methods in Java. One such accessor method related to strings is the length () method. length(): we are using this function to know the length of the string.

**Syntax: variable. Length();**

**class Main {**

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

**String str1 = "Java is fun";**

**System.out.println(str1.length());**

**}**

**}**

**// Output: 11**

3. How to **Concatenating tw0 strings:**

This method returns a new string which is string1 with string2 combined at the end. Concat() method can be used with string literals to get this done. Strings are also commonly concatenated using the + operator.

**public class ExerciseNew {**

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

**String s1="Hello,";**

**s1=s1.concat("What is your good name?");**

**System.out.println(s1);**

**}}**

**Output: Hello,What is your good name?**

**5) Explain the Methods of String functions in Java**

**startsWith & endsWith validation:**

**startsWith**: To verify whether the string is starting with a specific character or not.

**Syntax: variable. startsWith (“expected text”)**

**class Main {**

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

**String str = "JavaScript";**

**// checks if "JavaScript" starts with "Java"**

**System.out.println(str.startsWith("Java"));**

**}**

**}**

**// Output: true**

**endsWith**: To verify whether the variable is ending with a specific character or notSyntax: variable. endsWith (“expected text”)

**class Main {**

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

**String str = "Java Programming";**

**System.out.println(str.endsWith("mming")); // true**

**System.out.println(str.endsWith("g")); // true**

**System.out.println(str.endsWith("a Programming")); // true**

**System.out.println(str.endsWith("programming")); // false**

**System.out.println(str.endsWith("Java")); // false**

**}**

**}**

**ToLowerCase() & toUpperCase():**

**toLowerCase():** To convert string from uppercase into lowercase

**Syntax: variable. toLowerCase ();**

**toUpperCase:** To convert string from lowercase into uppercase

Syntax: variable. toUpperCase ();

**Equals() & equalsIgnoreCase():**

Equals(): To verify whether one string is equal to another string or not.

**Syntax: variable. equals (second/another variable)**

equalsIgnoreCase(): To verify whether a variable is the same as another variable or not irrespective of case.

**Syntax: variable. equalsIgnoreCase (another variable)**

**Contains():** To verify whether the string contains a specific value or string or not.

**Syntax: variable. contains(“value”);**

**toCharArray():** The Java String toCharArray() method converts the string to a char array and returns it.

**Syntax: string.toCharArray()**

Ex:

**class Main {**

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

**String str = "Java Programming";**

**// creating a char array**

**char[] result;**

**result = str.toCharArray();**

**System.out.println(result); // Java Programming**

**}**

**}**

**charAt():** To verify which character is available in which index of the string

Syntax: variable. charAT (index no);

**Note: Usually Length starts with 1 and Index in the string starts with 0**

Ex.:

**public class CharAtExample{**

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

**String name="javatpoint";**

**char ch=name.charAt(4);//returns the char value at the 4th index**

**System.out.println(ch);**

**}}**

**Output: t**

**indexOf():** To verify in which Index which character is available.

Syntax: variable.indexOf (char);

**Trim():** We are using this function to eliminate starting and ending spaces of the string (first and last spaces).

**Syntax: variable. trim ()**

**public class StringTrimExample{**

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

**String s1=" hello string ";**

**System.out.println(s1+"javatpoint");//without trim()**

**System.out.println(s1.trim()+"javatpoint");//with trim()**

**}}**

Output

hello string javatpoint

hello stringjavatpoint

**replace():** To replace old characters or Strings with new characters or strings in a Var.

Syntax: var. replace (“Old character/String”, “New Character/String”);

**Split():** To split variable/String into multiple substrings

**Syntax: variable. Split (“separator”);**

**Note: All split values will be stored in the array. That is the reason the return type for the split function is String [].**

Example:

Prepare a program to split a variable into multiple substrings and print all strings one by one

**isEmpty(): isEmpty()** method checks if the input string is empty or not. Note that here empty means the number of characters contained in a string is zero.public boolean isEmpty()

**public class IsEmptyExample{**

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

**String s1="";**

**String s2="javatpoint";**

**System.out.println(s1.isEmpty());**

**System.out.println(s2.isEmpty());**

**}}**

Output:

True

false

**String valueOf():** valueOf() method converts different types of values into strings. With the help of the string valueOf() method, you can convert int to string, long to string, boolean to string, character to string, float to string, double to string, object to a string, and char array to string.

Example

**public class StringValueOfExample{**

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

**int value=30;**

**String s1=String.valueOf(value);**

**System.out.println(s1+10);//concatenating string with 10**

**}}**

**Output: 3010**

**JAVA STRING METHODS:**

**The java.lang.String class provides many useful methods to perform operations on sequence of char values.**

|  |  |  |
| --- | --- | --- |
| **No.** | **String method** | **Description** |
| 1 | [char charAt(int index)](https://www.javatpoint.com/java-string-charat) | It returns char value for the particular index |
| 2 | [int length()](https://www.javatpoint.com/java-string-length) | It returns string length |
| 3 | [static String format(String format, Object... args)](https://www.javatpoint.com/java-string-format) | It returns a formatted string. |
| 4 | [static String format(Locale l, String format, Object... args)](https://www.javatpoint.com/java-string-format) | It returns formatted string with given locale. |
| 5 | [String substring(int beginIndex)](https://www.javatpoint.com/java-string-substring) | It returns substring for given begin index. |
| 6 | [String substring(int beginIndex, int endIndex)](https://www.javatpoint.com/java-string-substring) | It returns substring for given begin index and end index. |
| 7 | [boolean contains(CharSequence s)](https://www.javatpoint.com/java-string-contains) | It returns true or false after matching the sequence of char value. |
| 8 | [static String join(CharSequence delimiter, CharSequence... elements)](https://www.javatpoint.com/java-string-join) | It returns a joined string. |
| 9 | [static String join(CharSequence delimiter, Iterable<? extends CharSequence> elements)](https://www.javatpoint.com/java-string-join) | It returns a joined string. |
| 10 | [boolean equals(Object another)](https://www.javatpoint.com/java-string-equals) | It checks the equality of string with the given object. |
| 11 | [boolean isEmpty()](https://www.javatpoint.com/java-string-isempty) | It checks if string is empty. |
| 12 | [String concat(String str)](https://www.javatpoint.com/java-string-concat) | It concatenates the specified string. |
| 13 | [String replace(char old, char new)](https://www.javatpoint.com/java-string-replace) | It replaces all occurrences of the specified char value. |
| 14 | [String replace(CharSequence old, CharSequence new)](https://www.javatpoint.com/java-string-replace) | It replaces all occurrences of the specified CharSequence. |
| 15 | [static String equalsIgnoreCase(String another)](https://www.javatpoint.com/java-string-equalsignorecase) | It compares another string. It doesn't check case. |
| 16 | [String[] split(String regex)](https://www.javatpoint.com/java-string-split) | It returns a split string matching regex. |
| 17 | [String[] split(String regex, int limit)](https://www.javatpoint.com/java-string-split) | It returns a split string matching regex and limit. |
| 18 | [String intern()](https://www.javatpoint.com/java-string-intern) | It returns an interned string. |
| 19 | [int indexOf(int ch)](https://www.javatpoint.com/java-string-indexof) | It returns the specified char value index. |
| 20 | [int indexOf(int ch, int fromIndex)](https://www.javatpoint.com/java-string-indexof) | It returns the specified char value index starting with given index. |
| 21 | [int indexOf(String substring)](https://www.javatpoint.com/java-string-indexof) | It returns the specified substring index. |
| 22 | [int indexOf(String substring, int fromIndex)](https://www.javatpoint.com/java-string-indexof) | It returns the specified substring index starting with given index. |
| 23 | [String toLowerCase()](https://www.javatpoint.com/java-string-tolowercase) | It returns a string in lowercase. |
| 24 | [String toLowerCase(Locale l)](https://www.javatpoint.com/java-string-tolowercase) | It returns a string in lowercase using specified locale. |
| 25 | [String toUpperCase()](https://www.javatpoint.com/java-string-touppercase) | It returns a string in uppercase. |
| 26 | [String toUpperCase(Locale l)](https://www.javatpoint.com/java-string-touppercase) | It returns a string in uppercase using specified locale. |
| 27 | [String trim()](https://www.javatpoint.com/java-string-trim) | It removes beginning and ending spaces of this string. |
| 28 | [static String valueOf(int value)](https://www.javatpoint.com/java-string-valueof) | It converts given type into string. It is an overloaded method. |

# JAVA VARIABLES AND METHODS

1. **What is variable in java?**

*Variable*is like a container that is use to hold value of different data type.

You need to specify the data type of the variable while declaring it to make it clear what kind of value it will be storing.

**To use a variable, a programmar needs to:**

* declare a variable
* initialize the variable
* then use it

**For example. if we want to store a int value:**

**public int num;  //declare variable num**

**num=11;  //initialize value to variable num**

1. **How many types of variable are there?**

**We have following *types of variable* available in java:**

* local variables
* instance variable
* static variable
* method parameters

*You can study answers about the****next three questions****following the same link.*

1. **what is local variable?**

**A varia**ble declared inside method, block or loop is known as local variable.

Their scope or visibility is local to the entity in which they are declared. They cannot be used outside these entity.

A local variable should be initialized before use otherwise compiler error will occur.

1. **What is static variable?**

Static variables are also referred to as ***class variable*** as they belongs to a class.

We can declare a static variable using ***static keywords***.

Static variable is ***shared*** among different instances of a class. So static variables requires less storage memory space than instance variables.

1. **What is instance variable?**

An instance variable is used by objects ***to store their states***.

Each instance or object will have *separate value*which cannot be shared by other instances.

**Methods in java :**

1. **Explain the structure of methods in java**

* A Java method is a collection of statements that are grouped together to perform an operation. When you call the System.out.**println()** method, for example, the system actually executes several statements in order to display a message on the console.
* Now you will learn how to create your own methods with or without return values, invoke a method with or without parameters, and apply method abstraction in the program design.

**Example for creating a method in java :**

## Creating Method

Considering the following example to explain the syntax of a method −

**Syntax**

**public static int methodName(int a, int b) {**

**// body**

**}**

Here,

* **public static** − modifier
* **int** − return type
* **methodName** − name of the method
* **a, b** − formal parameters
* **int a, int b** − list of parameters

## Calling methods in java

For using a method, it should be called. There are two ways in which a method is called i.e., method returns a value or returning nothing (no return value).

The process of method calling is simple. When a program invokes a method, the program control gets transferred to the called method. This called method then returns control to the caller in two conditions, when −

* **the return statement is executed.**
* it reaches the method ending closing brace.

The methods returning void is considered as call to a statement. Lets consider an example −

System.out.println("This is tutorialspoint.com!");

The method returning value can be understood by the following example −

int result = sum(6, 9);

Following is the example to demonstrate how to define a method and how to call it −

**Example**

public class ExampleMinNumber {

   public static void main(String[] args) {

      int a = 11;

      int b = 6;

      int c = minFunction(a, b);

      System.out.println("Minimum Value = " + c);

   }

   /\*\* returns the minimum of two numbers \*/

   public static int minFunction(int n1, int n2) {

      int min;

      if (n1 > n2)

         min = n2;

      else

         min = n1;

      return min;

   }

}

**What is void keyword :**

The void keyword allows us to create methods which do not return a value. Here, in the following example we're considering a void method *methodRankPoints*. This method is a void method, which does not return any value. Call to a void method must be a statement i.e. *methodRankPoints(255.7);*. It is a Java statement which ends with a semicolon as shown in the following example.

public class ExampleVoid {

   public static void main(String[] args) {

      methodRankPoints(255.7);

   }

   public static void methodRankPoints(double points) {

      if (points >= 202.5) {

         System.out.println("Rank:A1");

      }else if (points >= 122.4) {

         System.out.println("Rank:A2");

      }else {

         System.out.println("Rank:A3");

      }

   }

}

**Explain passing parameters to value :**

Passing Parameters by Value means calling a method with a parameter. Through this, the argument value is passed to the parameter.

**Example**

The following program shows an example of passing parameter by value. The values of the arguments remains the same even after the method invocation.

public class swappingExample {

   public static void main(String[] args) {

      int a = 30;

      int b = 45;

      System.out.println("Before swapping, a = " + a + " and b = " + b);

      // Invoke the swap method

      swapFunction(a, b);

      System.out.println("\n\*\*Now, Before and After swapping values will be same here\*\*:");

      System.out.println("After swapping, a = " + a + " and b is " + b);

   }

   public static void swapFunction(int a, int b) {

      System.out.println("Before swapping(Inside), a = " + a + " b = " + b);

      // Swap n1 with n2

      int c = a;

      a = b;

      b = c;

      System.out.println("After swapping(Inside), a = " + a + " b = " + b);

   }

}

This will produce the following result −

**Output**

Before swapping, a = 30 and b = 45

Before swapping(Inside), a = 30 b = 45

**What is method overloading :**

When a class has two or more methods by the same name but different parameters, it is known as method overloading. It is different from overriding. In overriding, a method has the same method name, type, number of parameters, etc.

Let’s consider the example discussed earlier for finding minimum numbers of integer type. If, let’s say we want to find the minimum number of double type. Then the concept of overloading will be introduced to create two or more methods with the same name but different parameters.

The following example explains the same −

**Example**

public class ExampleOverloading {

   public static void main(String[] args) {

      int a = 11;

      int b = 6;

      double c = 7.3;

      double d = 9.4;

      int result1 = minFunction(a, b);

      // same function name with different parameters

      double result2 = minFunction(c, d);

      System.out.println("Minimum Value = " + result1);

      System.out.println("Minimum Value = " + result2);

   }

   // for integer

   public static int minFunction(int n1, int n2) {

      int min;

      if (n1 > n2)

         min = n2;

      else

         min = n1;

      return min;

   }

   // for double

   public static double minFunction(double n1, double n2) {

     double min;

      if (n1 > n2)

         min = n2;

      else

         min = n1;

      return min;

   }

}

This will produce the following result −

**Output**

Minimum Value = 6

Minimum Value = 7.3

## Using Command-Line Arguments

Sometimes you will want to pass some information into a program when you run it. This is accomplished by passing command-line arguments to main( ).

A command-line argument is the information that directly follows the program's name on the command line when it is executed. To access the command-line arguments inside a Java program is quite easy. They are stored as strings in the String array passed to main( ).

**Example**

The following program displays all of the command-line arguments that it is called with −

public class CommandLine {

   public static void main(String args[]) {

      for(int i = 0; i<args.length; i++) {

         System.out.println("args[" + i + "]: " +  args[i]);

      }

   }

}

Try executing this program as shown here −

$java CommandLine this is a command line 200 -100

This will produce the following result −

**Output**

args[0]: this

args[1]: is

args[2]: a

args[3]: command

args[4]: line

args[5]: 200

args[6]: -100

**what is finalize method :**

It is possible to define a method that will be called just before an object's final destruction by the garbage collector. This method is called **finalize( )**, and it can be used to ensure that an object terminates cleanly.

**For example, you might use finalize( ) to make sure that an open file owned by that object is closed.**

To add a finalizer to a class, you simply define the finalize( ) method. The Java runtime calls that method whenever it is about to recycle an object of that class.

Inside the finalize( ) method, you will specify those actions that must be performed before an object is destroyed.

The finalize( ) method has this general form −

protected void finalize( ) {

   // finalization code here

}

Here, the keyword protected is a specifier that prevents access to finalize( ) by code defined outside its class.

This means that you cannot know when or even if finalize( ) will be executed. For example, if your program ends before garbage collection occurs, finalize( ) will not execute.

# JAVA ACCESS SPECIFIER AND CONSTRUCTORS

1. what is scope of each access specifier?

**Public** – A class, method, constructor, interface etc declared public can be accessed from any other class. Therefore fields, methods, blocks declared inside a public class can be accessed from any class belonging to the Java . **Protected** – Variables, methods and constructors which are declared protected in a superclass can be accessed only by the subclasses in other package or any class within the package of the protected members’ class. **Default** – without any access modifier (i.e, public private or protected). It means that it is visible to all within a particular package. **Private** – Private variables or methods may be used only by an instance of the same class that declares the variable or method, A private feature may only be accessed by the class that own the feature

## What are access specifiers and access modifiers present in java , What is the difference b/w them?

By using access specifier we define that who one can access our class/method and variable (or whatever with that we use access specifier ). basically java access specifier are four types -

1. **public,**
2. **private,**
3. **protected, and**
4. **default**

**But acce**ss modifiers are properties of a class/method/variable while access modifiers are five types:

1. **final**
2. **static**
3. **Synchronized**
4. **abstract**
5. **transient**
6. **What is constructors in java**

**Constructor** is just like a method in [**Java**](https://javainterviewpoint.com/category/core-java/) that is used to initialize the state of an object and will be invoked during the time of object creation.

1. **What are the Rules for defining a constructor?**
2. Constructor **name** should be the same as the class name
3. It **cannot** contain any **return type**
4. It **can** have all **Access Modifiers** are allowed (private , public, protected, default)
5. It **Cannot** have any **Non Access Modifiers** (final ,static, abstract, synchronized)

### What is a No-arg constructor?

Constructor **without arguments** is called no-arg constructor. In Java Default constructor is a no-arg constructor.

class Demo

{

     public Demo()

     {

        //No-arg constructor

     }

}

* **No return** statement is allowed
* It **can** take any number of **parameters**
* Constructor can **throw exception**, we can have**throws clause**

### Why constructors cannot be abstract in Java?

When you set a method as abstract, then “The method doesn’t or cannot have body”. A constructor will be automatically called when object is created. It cannot lack a body moreover an abstract constructor could never be implemented.

|  |  |  |
| --- | --- | --- |
| |  | | --- | | 1. **Why every object constructor automatically call super() in Object before its own constructors?** | | |
| Ans. Derived object carries the body of its class as well as the body of the parent class. Its body ( member elements ) is initialized using its own class constructor whereas the body ( member elements ) carried from the parent class are initialized using super class constructor. So In order to initialize the elements of the parent class before its own elements are even initialized, super is called. | |
| |  |  | | --- | --- | | 1. **Are constructors inherited? Can a subclass call the parent's class constructor? When?** |  | |  |  | |  |
| Ans. You cannot inherit a constructor. That is, you cannot create a instance of a subclass using a constructor of one of it's superclasses. One of the main reasons is because you probably don't want to override the superclasses constructor, which would be possible if they were inherited. By giving the developer the ability to override a superclasses constructor you would erode the encapsulation abilities of the language. |  |

1. **Which access specifier is used when no access specifier is used with a member of class (java)?**a) Private  
   b) Default  
   c) Protected  
   d) Public

Ans B

Explanation : Default access is used if the programmer doesn’t specify the specifier. This acts in a similar way as that of private. But since nothing is specified we call it to default access.

1. Which specifier allows a programmer to make the private members which can be inherited?  
   a) Private  
   b) Default  
   c) Protected  
   d) Protected and default

Ans C:

Explanation : Protected access is used to make the members private. But those members can be inherited. This gives both security and code reuse capability to a program.

1. Which among the following is false?  
   a) Private members can be accessed using friend functions  
   b) Member functions can be made private  
   c) Default members can’t be inherited  
   d) Public members are accessible from other classes also

Answer: c  
Explanation: The default members can be inherited. Provided that they are in same package. It works in a little different way from private access specifier.

1. If a class has all the private members, which specifier will be used for its implicit constructor?  
   a) Private  
   b) Public  
   c) Protected  
   d) Default

Answer: b  
Explanation: The implicit constructor will always be public. Otherwise the class wouldn’t be able to have instances. In turn, no objects will be created and the class can only be used for inheritance.

1. If class A has add() function with protected access, and few other members in public. Then class B inherits class A privately. Will the user will not be able to call \_\_\_\_\_\_\_\_\_ from the object of class B.  
   a) Any function of class A  
   b) The add() function of class A  
   c) Any member of class A  
   d) Private, protected and public members of class A

Answer: d  
Explanation: Class B object will not be able to call any of the private, protected and public members of class A. It is not only about the function add(), but all the members of class A will become private members of class B.

1. On which specifier’s data, does the size of a class’s object depend?  
   a) All the data members are added  
   b) Only private members are added  
   c) Only public members are added  
   d) Only default data members are added

Answer: a  
Explanation: All the data members are counted to calculate the size of an object of a class. The data member access specifier doesn’t play any role here. Hence all the data size will be added.

1. If an abstract class has all the private members, then \_\_\_\_\_\_\_\_\_  
   a) No class will be able to implement members of abstract class  
   b) Only single inheritance class can implement its members  
   c) Only other enclosing classes will be able to implement those members  
   d) No class will be able to access those members but can implement.

Answer: a  
Explanation: The classes which inherit the abstract class, won’t be able to implement the members of abstract class. The private members will not be inherited. This will restrict the subclasses to implement those members.

1. Which access specifier should be used so that all the parent class members can be inherited and accessed from outside the class?  
   a) Private  
   b) Default or public  
   c) Protected or private  
   d) Public

Answer: d  
Explanation: All the members must be of public access. So that the members can be inherited easily. Also, the members will be available from outside the class.

# JAVA OOPS CONCEPTS\_1

**1. What is Encapsulation in Java? Why is it called Data hiding?**

1. Ans: The process of binding data and corresponding methods (behavior) together into a single unit is called encapsulation in Java.
2. In other words, encapsulation is a programming technique that binds the class members (variables and methods) together and prevents them from being accessed by other classes, thereby we can keep variables and methods safes from outside interference and misuse.
3. If a field is declared private in the class then it cannot be accessed by anyone outside the class and hides the fields within the class. Therefore, Encapsulation is also called data hiding.
4. Abstraction & Encapulation

**2) What is the advantage of Encapsulation?**

Ans: There are the following advantages of encapsulation in Java. They are as follows:

* The encapsulated code is more flexible and easy to change with new requirements.
* It prevents the other classes to access the private fields.
* Encapsulation allows modifying implemented code without breaking other code who have implemented the code.
* It keeps the data and codes safe from external inheritance. Thus, Encapsulation helps to achieve security.
* It improves the maintainability of the application.

**3) What is a Tightly encapsulated class in Java?**

Ans: If each variable is declared as private in the class, it is called tightly encapsulated class in Java. For tightly encapsulated class, we are not required to check whether class contains getter and setter method or not and whether these methods are declared as public or not

**4. What is the difference between Abstraction and Encapsulation?** **Or, how abstraction is different from encapsulation in Java?**

Ans: There are the following differences between Abstraction and Encapsulation:

a) Abstraction solves the problem at the design level whereas encapsulation solves the problem at the implementation level.

b) Abstraction is implemented in Java using Interface and Abstract class whereas encapsulation is implemented using private and protected access modifiers.

c) Abstraction is used to hide the unwanted data and giving relevant data whereas encapsulation is used for hiding data and code in a single unit to prevent access from outside.

d) The real-time example of Abstraction is TV Remote Button whereas the real-time example of Encapsulation is medical medicine.

**5) Explain design pattern based on encapsulation in java?**

Ans: In many design patterns, Java uses the encapsulation technique and one of them is Factory pattern which is used to create the objects.

Factory pattern is a better choice in creating the object of those classes whose creation logic can vary. It is also used for creating different implementations of the same interface.

‘BorderFactory class’ of JDK is a good example of encapsulation in Java which creates different types of ‘border’ and encapsulates creation logic of border.

**6)Find which of the following uses encapsulation?**a) void main(){ int a; void fun( int a=10; cout<<a); fun(); }  
b) class student{ int a; public: int b;};  
c) class student{int a; public: void disp(){ cout<<a;} };  
d) struct topper{ char name[10]; public : int marks; }

**Answer: c**Explanation: It is the class which uses both the data members and member functions being declared inside a single unit. Only data members can be there in structures also. And the encapsulation can only be illustrated if some data/operations are associated within class.

**7) Encapsulation helps in writing \_\_\_\_\_\_\_\_\_\_\_ classes in java.**a) Mutable  
b) Abstract  
c) Wrapper  
d) Immutable

**Answer: d**Explanation: Immutable classes are used for caching purpose generally. And it can be created by making the class as final and making all its members private.

**8) Which among the following should be encapsulated?**a) The data which is prone to change is near future  
b) The data prone to change in long terms  
c) The data which is intended to be changed  
d) The data which belongs to some other class  
**Answer: a**Explanation: The data prone to change in near future is usually encapsulated so that it doesn’t get changed accidentally. We encapsulate the data to hide the critical working of program from outside world.

9) **Which among the following can be a concept against encapsulation rules?  
a)** Using function pointers  
b) Using char\* string pointer to be passed to non-member function  
c) Using object array  
d) Using any kind of pointer/array address in passing to another function

**Answer: d**Explanation: If we use any kind of array or pointer as data member which should not be changed, but in some case its address is passed to some other function or similar variable. There are chances to modify its whole data easily. Hence Against encapsulation

# JAVA OOPS CONCEPTS\_2

**1)** What is Object class in Java programming language?

,

Object class defined in *java.lang* package is the superclass of all other classes defined in Java programming language. Every class extends from the Object class either directly or indirectly. All classes inherit the instance methods defined in the Object class.

### 2) What do you understand by OOP?

OOP stands for object-oriented programming. It is a programming paradigm that revolves around the object rather than function and procedure. In other words, it is an approach for developing applications that emphasize on objects. An object is a real word entity that contains data and code. It allows binding data and code together.

### 3) What is the purpose of using OOPs concepts?

The aim of OOP is to implement real-world entities like inheritance, hiding, polymorphism in programming. The main purpose of OOP is to bind together the data and the functions that operate on them so that no other part of the code can access this data except that function.

4) What are the advantages and disadvantages of OOP?

**Advantages of OOP**

* It follows a bottom-up approach.
* It models the real word well.
* It allows us the reusability of code.
* Avoids unnecessary data exposure to the user by using the abstraction.
* OOP forces the designers to have a long and extensive design phase that results in better design and fewer flaws.
* Decompose a complex problem into smaller chunks.
* Programmer are able to reach their goals faster.
* Minimizes the complexity.
* Easy redesign and extension of code that does not affect the other functionality.

**Disadvantages of OOP**

* Proper planning is required.
* Program design is tricky.
* Programmer should be well skilled.
* Classes tend to be overly generalized.

### 5) What do you understand by class and object? Also, give example.

**Class:** A class is a blueprint or template of an object. It is a user-defined data type. Inside a class, we define variables, constants, member functions, and other functionality. It does not consume memory at run time. Note that classes are not considered as a data structure. It is a logical entity. It is the best example of data binding.

**Object:** An object is a real-world entity that has attributes, behavior, and properties. It is referred to as an instance of the class. It contains member functions, variables that we have defined in the class. It occupies space in the memory. Different objects have different states or attributes, and behaviors.

**6) What are the differences between class and object?**

|  |  |
| --- | --- |
| **Class** | **Object** |
| **It is a logical entity.** | **It is a real-world entity.** |
| **It is conceptual.** | **It is real.** |
| **It binds data and methods together into a single unit.** | **It is just like a variable of a class.** |
| **It does not occupy space in the memory.** | **It occupies space in the memory.** |
| **It is a data type that represents the blueprint of an object.** | **It is an instance of the class.** |
| **It is declared once.** | **Multiple objects can be declared as and when required.** |
| **It uses the keyword class when declared.** | **It uses the new keyword to create an object.** |
| **A class can exist without any object.** | **Objects cannot exist without a class.** |

**7) Which of the following is not OOPS concept in Java?**a) Inheritance  
b) Encapsulation  
c) Polymorphism  
d) Compilation

Ans : D

Explanation: There are 4 OOPS concepts in Java. Inheritance, Encapsulation, Polymorphism and Abstraction.

8) **Which language does not support polymorphism but supports classes?**

**a.** Ada

b. C++

c. Java

d. SmallTalk

**Answer:** (a) Ada

9) **If in case, in multiple inheritances, a class R would inherit the Class Q, while Class Q would inherit the class P, then in which sequence would their destructors be called in case we declare an object of Class R?**

**a.** ~R() then ~P() then ~Q()

b. ~P() then ~Q() then ~R()

c. ~Q() then ~R() then ~P()

d. ~R() then ~Q() then ~P()

**Answer:** (d) ~R() then ~Q() then ~P()

**10) Which of these classes is a specialization of some more general template classes?**

a. Integer

b. Maths

c. String

d. Digit

**Answer:** (c) String

**11) .** A derived class is also called a \_\_\_\_\_\_\_\_\_\_\_\_\_\_.

a. Small class

b. Subclass

c. Noticeable class

d. Big class

**Answer:** (b) Subclass

**12)  If in case a class is called in the main function directly and has a public member function, then \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_.**

a. It will lead to the production of an undeclared function error

b. It will lead to the production of an out of memory error

c. The program will only give warning

d. The program will shut the computer down

**Answer:** (a) It will lead to the production of an undeclared function error

# INHERITANCE AND POLYMORPHISM

**1)What is Polymorphism in Java OOPs?**

ns: Polymorphism in java is one of the core concepts of object-oriented programming system. Polymorphism means “many forms” in Greek. That is one thing that can take many forms.

Polymorphism is a concept by which we can perform a single task in different ways. That is, when a single entity (object) behaves differently in different cases, it is called polymorphism.

In other words, if a single object shows multiple forms or multiple behaviors, it is called polymorphism.

**2) What are different ways to achieve or implement polymorphism in Java?**

Ans: Polymorphism in Java can be primarily achieved by subclassing or by implementing an interface. The subclasses can have their own unique implementation for certain features and can also share some of the functionality through inheritance

**3) What are the differences between Polymorphism and Inheritance in Java?**

Ans: The differences between polymorphism and inheritance in java are as follows:

a. Inheritance represents the parent-child relationship between two classes. On the other hand, polymorphism takes the advantage of that relationship to make the program more dynamic

Inheritance helps in code reusability in child class by inheriting behavior from parent class. On the other hand, polymorphism enables child class to redefine already defined behavior inside parent class.

Without polymorphism, it is not possible for a child class to execute its own behavior.

**4) What is Runtime Polymorphism (Dynamic Polymorphism)?**

Ans: A polymorphism where object binding with methods happens at runtime is called runtime polymorphism. In runtime polymorphism, the behavior of a method is decided at runtime.

JVM (Java Virtual Machine) binds the method call with method definition/body at runtime and invokes the relevant method during runtime when the method is called. This happens because objects are created at runtime and the method is called using an object of the class.

**5) What are the differences between compile-time polymorphism and runtime polymorphism in java?**

Ans: There are three main differences between compile-time polymorphism and runtime polymorphism that are as follows:

a) In the compile-time polymorphism, the behavior of a method is decided at compile-time. Hence, Java compiler binds method calls with method definition/body during compilation.

In runtime polymorphism, the behavior of a method is decided at runtime, JVM binds the method call with method definition at runtime and invokes the relevant method during runtime when the method is called.

b) Compile time polymorphism is also known as early binding because the binding is performed at compile time.

Runtime polymorphism is also known as late binding because the binding is performed at runtime.

c) Compile time polymorphism can be achieved via method overloading.

Runtime polymorphism can be achieved via method overriding.

**6) Which of the following language supports polymorphism but not the classes?**

* C++ programming language
* Java programming language
* Ada programming language
* C# programming language

**Ans: C**

**Explanation:** It is a programming language that disapproves of the concept of polymorphism but supports the concept of classes. It is an object-based language. So, it does not follow the Object-oriented programming concepts.

**7)Define the programming language, which does not support all four types of inheritance?**

* Smalltalk
* Kotlin
* Java
* C++

**Ans C**

**Explanation:** Java is a programming language that disapproves of the concept of 'multiple inheritance'. So, it does not agree with all types of inheritance. But, we can implement 'multiple inheritance' in Java language using the interface concept.

8) Which operator from the following can be used to illustrate the feature of polymorphism?

* Overloading <<
* Overloading &&
* Overloading | |
* Overloading +=

**Ans:** a. Overloading <<

**Explanation:** << is an insertion operator which is used for overloading (polymorphism).

**9) Which of the following definition is incorrect for polymorphism?**

* Polymorphism helps in redefining the same functionality
* Polymorphism concept is the feature of object-oriented programming (OOP)
* It always increases the overhead of function definition
* Ease in the readability of the program

**Ans:** c. It always increases the overhead of function definition

**Explanation:** This concept of OOPS never increases the overhead of function definition.

**10) Which function best describe the concept of polymorphism in programming languages?**

* Class member function
* Virtual function
* Inline function
* Undefined function

**Answer:** b. Virtual function

**Explanation:** Only those functions are used to achieve the polymorphism, which are declared as 'virtual'. These functions let the OOPS programs decide at runtime which function is to be called by the pointer.

# EXCEPTIONAL HANDLING

1. **When does Exceptions in Java arises in code sequence?**a) Run Time  
   b) Compilation Time  
   c) Can Occur Any Time  
   d) None of the mentioned

**ANS : Answer: a**Explanation: Exceptions in Java are run-time errors.

1. **2) Which of these keywords must be used to monitor for exceptions?**a) try  
   b) finally  
   c) throw  
   d) catch

ANS : a

1. **3) Which of these keywords is used to manually throw an exception?**a) try  
   b) finally  
   c) throw  
   d) catch

ANS : C

1. **What will be the output of the following Java program?**

**class** exception\_handling

   {

**public** **static** **void** main(String args[])

        {

**try**

            {

                System.out.print("Hello" + " " + 1 / 0);

            }

**catch**(ArithmeticException e)

            {

         System.out.print("World");

            }

       }

   }

1. **Hello  
   b) World  
   c) HelloWorld  
   d) Hello World**

**ANS: b**Explanation: System.ou.print() function first converts the whole parameters into a string and then prints, before “Hello” goes to output

1. **Which of these keywords must be used to handle the exception thrown by try block in some rational manner?**

[**A.**](about:blank) try

[**B.**](about:blank) finally

[**C.**](about:blank) throw

[**D.**](about:blank) catch

**ANS : Option D**

Explanation: If an exception occurs within the try block, it is thrown and cached by catch block f or processing.

**6) Which statement is true?**

A. An Error that might be thrown in a method must be declared as thrown by that method, or be handled within that method.  
B. Multiple catch statements can catch the same class of exception more than once.  
C. A try statement must have at least one corresponding catch block.  
D. Except in case of VM shutdown, if a try block starts to execute, a corresponding finally block will always start to execute.

**Ans : D**

Explanation: A is wrong. A try statement can exist without catch, but it must have a finally statement. B is wrong. A try statement executes a block. If a value is thrown and the try statement has one or more catch clauses that can catch it, then control will be transferred to the first such catch clause. If that catch block completes normally, then the try statement completes normally. C is wrong. Exceptions of type Error and RuntimeException do not have to be caught, only checked exceptions (java.lan

**7) Which of these keywords is used to manually throw an exception?**

A. finally  
B. throw  
C. catch  
D. try

**Ans : B**

Explanation: Throw keywords is used to manually throw an exception.

**8) Which of these exceptions will occur if we try to access the index of an array beyond its length?**

ArithmeticException  
B. ArrayException  
C. ArrayIndexException  
D. ArrayIndexOutOfBoundsException

**Ans : D**

Explanation: ArrayIndexOutOfBoundsException is a built in exception that is caused when we try to access an index location which is beyond the length of an array.

### 9)  What are the Exception Handling Keywords in Java?

**There are four keywords used in java exception handling.**

1. **throw**: Sometimes we explicitly want to create an exception object and then throw it to halt the normal processing of the program. The **throw** keyword is used to throw exceptions to the runtime to handle it.
2. **throws**: When we are throwing any checked exception in a method and not handling it, then we need to use the throws keyword in the method signature to let the caller program know the exceptions that might be thrown by the method. The caller method might handle these exceptions or propagate them to its caller method using the throws keyword. We can provide multiple exceptions in the throws clause and it can be used with the **main()** method also.
3. **try-catch**: We use try-catch block for exception handling in our code. try is the start of the block and catch is at the end of the try block to handle the exceptions. We can have multiple catch blocks with a try and try-catch blocks can be nested also. catch block requires a parameter that should be of type Exception.
4. **finally**: The finally block is optional and can be used only with a try-catch block. Since exception halts the process of execution, we might have some resources open that will not get closed, so we can use the finally block. The finally block gets executed always, whether an exception occurs or not.

### 10) What is the difference between Checked and Unchecked Exceptions in Java?

1. Checked Exceptions should be handled in the code using try-catch block or else the method should use the throws keyword to let the caller know about the checked exceptions that might be thrown from the method. Unchecked Exceptions are not required to be handled in the program or to mention them in the throws clause of the method.
2. Exception is the superclass of all checked exceptions whereas RuntimeException is the superclass of all unchecked exceptions. Note that RuntimeException is the child class of Exception.
3. Checked exceptions are error scenarios that require to be handled in the code, or else you will get compile time error. For example, if you use FileReader to read a file, it throws FileNotFoundException and we must catch it in the try-catch block or throw it again to the caller method. Unchecked exceptions are mostly caused by poor programming, for example, NullPointerException when invoking a method on an object reference without making sure that it’s not null. For example, I can write a method to remove all the vowels from the string. It’s the caller’s responsibility to make sure not to pass a null string. I might change the method to handle these scenarios but ideally, the caller should take care of this.

### 11)  Can we have an empty catch block?

### We can have an empty catch block but it’s an example of bad programming. We should never have an empty catch block because if the exception is caught by that block, we will have no information about the exception and it wil be a nightmare to debug it. There should be at least a logging statement to log the exception details in console or log files.

### 12) Difference between throw and throws keyword in java?

*throw Keyword:*

* throw keyword is basically used for custom exception

and it is used to explicitly throw an exception.

* throw is used within the method.
* It is followed by an instance.
* By the help of throw keyword, we cannot forward checked exception in calling chain.

*throws Keyword:*

* It is used to declare an exception.
* It is used with the method signature.
* It is followed by a class.
* By the help of throws keyword we can forward checked exception in    calling chain.

# JAVA COLLECTIONS\_1

**1)What is an Array?**

1. Array is a collection of similar data types. It can not have different data type. It can hold both primitive types (int, float, double) and object references.
2. It is fixed in length i.e static in nature.
3. Arrays are created on the heap memory not on the stack.
4. Accessing an invalid index of an Array will cause exception.

2) **How do you declare an Array in java?**  
  
You can declare an Array in java by the following way :  
  
dataType[]    arrayVariableName  = new dataType[arraySize];  
  
for example for int data type, you can declare an int array as :  
  
int[]  temp = new int[256]

3) **What is ArrayStoreException ? When this exception is thrown ?**  
  
ArrayStoreException is a runtime exception. Array must contain the same data type elements.  
  
This exception is thrown to indicate that an attempt has been made to store the wrong type of object into an array of objects. In other words, if you want to store the integer Object in an Array of String you will get ArrayStoreException.  
  
The following code throws ArrayStoreException :  
  
example:

**public** **class** **Javaprogram**{

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

       Object x[] = **new** String[**3**];

         x[**0**] = **new** Integer(**0**);

    }

}

***4) What is an ArrayList?***

An ArrayList is a data structure that allows you to store a collection of items in an ordered fashion. You can access items in the ArrayList by their index, and you can add or remove items from the ArrayList as needed.

***5) Why should we use Array Lists instead of arrays?***

Array Lists offer a number of advantages over arrays, including the ability to dynamically resize the list as needed, the ability to add and remove elements from the list, and the ability to easily sort and search the list.

**Array list in Java declaration**

This is how you can declare an ArrayList of String type:

ArrayList<String> list=new ArrayList<>();

This is how you can declare an ArrayList of Integer type:

Integers :

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

Strings :

ArrayList<String> arrList=new ArrayList<String>();

Example: Adding elements to array list

import java.util.\*;

class JavaExample{

   public static void main(String args[]){

      ArrayList<String> arrList=new ArrayList<String>();

      arrList.add("Steve");

      arrList.add("Tim");

      arrList.add("Lucy");

      arrList.add("Pat");

      arrList.add("Angela");

      arrList.add("Tom");

      //displaying elements

      System.out.println(arrList);

      //Adding "Steve" at the fourth position

      arrList.add(3, "Steve");

      //displaying elements

      System.out.println(arrList);

   }

}

**6) Which of the following classes implements the RandomAccess interface? (More than one options are correct)?**

a) ArrayList  
b) LinkedList  
c) Vector  
d) Stack

Ans : A and C

Explanation : rrayList & Vector classes implements RandomAccess interface. Hence they can access any random element with the same speed. It means retrieving nth element directly without retrieving (n-1) elements.

**7) Find the output of the below program?**

import java.util.\*;

import java.io.\*;

public class Test {

   public static void main(String[] args) {

      List l1 = new Vector();

      List l2 = new ArrayList();

      Stack s1 = new Stack();

      System.out.println(l1 instanceof RandomAccess);

      System.out.println(l2 instanceof AutoCloseable);

      System.out.println(s1 instanceof Serializable);

   }

}

a) true, true, true  
b) true, false, false  
c) true, false, true  
d) false, false, false

**Ans True false false**

**8)  What will be the output of the below ArrayList program?**

import java.util.ArrayList;

import java.util.List;

public class Test {

   public static void main(String[] args) {

      List<String> al = new ArrayList<String>();

      al.add("A");

      al.add("B");

      al.add("A");

      al.add(null);

      System.out.println(al);

   }

}

a) Compile time error  
b) [A, B, null]  
c) [A, B]  
d) [A, B, A, null]

Ans : D (A,B,A,null)

explanation : In ArrayList duplicate objects are allowed and we can also insert null values.\

**9) Find the output of the below Java ArrayList program?**

import java.util.ArrayList;

import java.util.List;

public class Test {

   public static void main(String[] args) {

      List<String> al = new ArrayList<String>();

      al.add("A");

      al.add("B");

      al.add("A");

      al.add("B");

      al.add("A");

      al.remove(3);

      System.out.println(al);

   }

}

a) Compile time error  
b) [A, B, A, B]  
c) [A, B, A, A]  
d) None of these

Answer:- c) [A, B, A, A]  
Explanation : The remove(index) method removes the element at the given index. The index starts from 0.

**10)What are the important features of ArrayList in Java?**

Answer: There are several significant features of ArrayList in Java that are as follows:

a) ArrayList in Java uses an index-based structure.

b) The size of ArrayList can increase or decrease at runtime. Once ArrayList is created, we can add any number of elements.

c) An ArrayList allows adding elements into the middle of collection.

d) It allows to delete elements.

e) Duplicate elements are allowed in the array list.

f) Any number of null elements can be added to ArrayList.

g) ArrayList maintains the insertion order in Java. That is insertion order is preserved.

h) ArrayList is not synchronized. That means multiple threads can use the same ArrayList objects simultaneously.

i) Since ArrayList implements random access interface, we can get, set, insert, and remove elements of the array list from any arbitrary position.

j) The performance of ArrayList is slow because if any element is removed from ArrayList, a lot of shifting takes place.

**11) Why is ArrayList called a dynamically growing array in Java?**

Answer: ArrayList is called a dynamically growing array in java because ArrayList uses a dynamic array internally for storing a group of elements.

If the initial capacity of the array is exceeded, a new array with a larger capacity is created automatically and all the elements from the current array are copied to the new array.

When elements are removed from the array list, the size of array list can be shrunk automatically.

**12) Is it possible to join two or more ArrayLists in Java?**

Answer: Yes, we can join two or more ArrayLists in java. List interface provides a method addAll() to join two or more lists in java.

If we have one list list1 and another list2, we can join them with the help of addAll() like this: list1.addAll(list2);

**13) Suppose we want to add an element in the middle of list. Which list implementation will provide you better performance? ArrayList or LinkedList?**

Answer: For the above scenario, LinkedList is a better choice because in the case of LinkedList, when we add an element at the specified position, internally, a node is created and only two links are changed.

But in the case of ArrayList, a lot of shifting is done in the memory when we add an element in the middle of the list or anywhere, except at the end.

**14) Both ArrayList and LinkedList provide get() method to retrieve an element at the specified position from the list. Which one is faster, ArrayList or LinkedList?**

Answer: ArrayList’s get() method is faster than LinkedList’s get() because LinkedList does not implement Random Access Interface.

Due to which it will traverse from the beginning or ending over the list until it reaches the index specified.

15) **Which collection classes implement List interface in Java?**

Answer: The collection classes that implement List interface, are as:

* ArrayList
* LinkedList
* CopyOnWriteArrayList
* Vector
* Stack

# JAVA COLLECTIONS\_2

1. Which of these is not a interface in the Collections Framework?

A. Set

B. List

C. Group

D. Collection

**Ans : C**

2) Which of these packages contain all the collection classes?

A. java.net

B. java.awt

C. java.lang

D. java.util

**Ans :D**

1. Which of these classes is not part of Java’s collection framework?

A. Maps

B. Stack

C. Array

D. Queue

**Ans D**

1. Can elements of a Set be traversed without using Iterator?

A. Yes

B. No

**Ans A**

1. Which implementation of Iterator can traverse a collection back and forth?

A. Iterator

B. ListIterator

C. SetIterator

D. MapIterator

**Ans B**

**Explanation:- Iterator can only traverse forward while ListIterator traverses both forward and backward. ListIterator can help replace an element, while Iterator cannot.**

1. List, Set and Queue \_\_\_\_\_\_\_\_\_\_ Collection.

A. extends

B. implements

C. both of the above

D. none of the aboven

**Ans :A**

**Explanation:List, Set and Queue are all interfaces and they extends Collection interface. (interface extends interface)**

1. **What is collections in java**

In [**Java**](https://www.interviewbit.com/online-java-compiler/), a collection is a framework that provides an architecture for storing and manipulating a collection of objects. In JDK 1.2, a new framework called "Collection Framework" was created, which contains all of the collection classes and interfaces.

[**Collections in Java**](https://www.interviewbit.com/java-interview-questions/) are capable of doing any data operations such as searching, sorting, insertion, manipulation, and deletion. A single unit of objects in Java is referred to as a collection. The two basic “root” interfaces of Java collection classes are the Collection interface (java.util.Collection) and the Map interface(java.util.Map). Many interfaces (Set, List, Queue, Deque) and classes are available in the Java Collection [**framework**](https://www.interviewbit.com/blog/java-frameworks/) (ArrayList, Vector, LinkedList, PriorityQueue, HashSet,  LinkedHashSet, TreeSet).

1. **Distinguish between ArrayList and Vector in the Java collection framework.**

|  |  |
| --- | --- |
| **ArrayList** | **Vector** |
| ArrayList is cannot be synchronized. | Vector can be is synchronized. |
| It is not a legacy class. | It is a legacy class. |
| It can increase its size by 50% of the size of the array. | It can increase its size by doubling the size of the array. |
| ArrayList is not thread-safe. | Vector is a thread-safe. |

1. **List out benefits of generic collection**

The benefits of using the generic collection are:

* If the programmers are using generic class, they don’t require typecasting.
* It is type-safe and can be checked at the time of compilation.
* It provides the stability of the code by detecting bug at the compilation time.

**10)Explain the basic interfaces of the Java collections framework**

Java collection framework is a root of the collection hierarchy. It represents a group of objects as its elements. The Java programming language does not provide a direct implementation of such interface.

* **Set:** Set is a collection having no duplicate elements. It uses hashtable for storing elements.
* **List:**List is an ordered collection that can contain duplicate elements. It enables developers to access any elements from its inbox. The list is like an array having a dynamic length.
* **MAP:**It is an object which maps keys to values. It cannot contain duplicate keys. Each key can be mapped to at least one value.

### 11)Define emptySet() in the Java collections framework

Method emptySet() that returns the empty immutable set whenever programmers try to remove null elements. The set which is returned by emptySet() is serializable. The syntax of this method is:

**public static final <T> Set<T> emptySet()**

**12)Differentiate between Collection and Collections**

The difference between Collec tion and Collections are:

|  |  |
| --- | --- |
| **Collection** | **Collections** |
| The collection is an interface. | Collections is a class. |
| It represents a group of objects as a single entity. | It defines various utility methods for collection objects. |
| The collection is the root interface of the Java Collection framework. | Collections is a general utility class. |
| This interface is used to derive the collection data structures. | This class contains static methods to manipulate data structure. |

**13)What are the benefits of the Collection Framework in Java?**

The benefits of Collection Framework in Java are:

* Java collection framework offers highly efficient and effective data structures that enhance the accuracy and speed of the program.
* The program developed with the Java collection framework is easy to maintain.
* A developer can mix classes with other types that result in increasing the reusability of code.
* The Java collection framework enables programmers to modify the primitive collection types the way they like.

### 14)What is a good way to sort the Collection objects in Java?

A good way to sort Java collection objects is using Comparable and Comparator interfaces. A developer can use Collections.sort(), the elements are sorted based on the order mention in compareTo().

When a developer uses Collections, sort (Comparator), it sorts the objects depend on compare() of the Comparator interface.

**15)How do you remove an entry from a Collection? and subsequently what is the difference between the remove() method of Collection and remove() method of Iterator, which one you will use while removing elements during iteration?**  
  
Collection interface defines remove(Object obj) method to remove objects from Collection. List interface adds another method remove(int index), which is used to remove objects at a specific index. You can use any of these methods to remove an entry from Collection, while not iterating.

**16)Things change when you iterate. Suppose you are traversing a List and removing only certain elements based on logic, then you need to use Iterator's remove() method. This method removes the current element from Iterator's perspective.**

If you use Collection's or List's remove() method during iteration then your code will throw ConcurrentModificationException. That's why it's advised to use the Iterator remove() method to remove objects from Collection.

**17)why we required collection frameworks in java**

The Java collections framework provides various data structures and algorithms that can be used directly. This has two main advantages:

* **We do not have to write code to implement these data structures and algorithms manually.**
* **Our code will be much more efficient as the collections framework is highly optimized.**

**M**oreover, the collections framework allows us to use a specific data structure for a particular type of data. Here are a few examples,

* **If we want our data to be unique, then we can use the Set interface provided by the collections framework.**
* **To store data in**key/value**pairs, we can use the Map interface.**
* **The ArrayList class provides the functionality of resizable arrays.**

**public** WebDriver ldriver;

**public** Add\_New\_customer\_Page(WebDriver rdriver)

{

ldriver=rdriver;

PageFactory.*initElements*(ldriver, **this**);

}

By lnkCustomers\_menu=By.*xpath*("//a[@href='#']//span[contains(text(),'Customers')]");

By lnkCustomers\_menuitem=By.*xpath*("//span[@class='menu-item-title'][contains(text(),'Customers')]");

By btnAddnew=By.*xpath*("//a[@class='btn bg-blue']"); //Add new

By txtEmail=By.*xpath*("//input[@id='Email']");

By txtPassword=By.*xpath*("//input[@id='Password']");

By txtcustomerRoles=By.*xpath*("//div[@class='k-multiselect-wrap k-floatwrap']");

By lstitemAdministrators=By.*xpath*("//li[contains(text(),'Administrators')]");

By lstitemRegistered=By.*xpath*("//li[contains(text(),'Registered')]");

By lstitemGuests=By.*xpath*("//li[contains(text(),'Guests')]");

By lstitemVendors=By.*xpath*("//li[contains(text(),'Vendors')]");

By drpmgrOfVendor=By.*xpath*("//\*[@id='VendorId']");

By rdMaleGender=By.*id*("Gender\_Male");

By rdFeMaleGender=By.*id*("Gender\_Female");

By txtFirstName=By.*xpath*("//input[@id='FirstName']");

By txtLastName=By.*xpath*("//input[@id='LastName']");

By txtDob=By.*xpath*("//input[@id='DateOfBirth']");

By txtCompanyName=By.*xpath*("//input[@id='Company']");

By txtAdminContent=By.*xpath*("//textarea[@id='AdminComment']");

By btnSave=By.*xpath*("//button[@name='save']");

//Actions Methods

**public** String getPageTitle()

{

**return** ldriver.getTitle();

}

**public** **void** clickOnCustomersMenu() {

ldriver.findElement(lnkCustomers\_menu).click();

}

**public** **void** clickOnCustomersMenuItem() {

ldriver.findElement(lnkCustomers\_menuitem).click();

}

**public** **void** clickOnAddnew() {

ldriver.findElement(btnAddnew).click();

}

**public** **void** setEmail(String email)

{

ldriver.findElement(txtEmail).sendKeys(email);

}

**public** **void** setPassword(String password)

{

ldriver.findElement(txtPassword).sendKeys(password);

}

**public** **void** setCustomerRoles(String role) **throws** InterruptedException

{

**if**(!role.equals("Vendors")) //If role is vendors should not delete Register as per req.

{

ldriver.findElement(By.*xpath*("//\*[@id=\"SelectedCustomerRoleIds\_taglist\"]/li/span[2]")).click();

}

ldriver.findElement(txtcustomerRoles).click();

WebElement listitem;

Thread.*sleep*(3000);

**if**(role.equals("Administrators"))

{

listitem=ldriver.findElement(lstitemAdministrators);

}

**else** **if**(role.equals("Guests"))

{

listitem=ldriver.findElement(lstitemGuests);

}

**else** **if**(role.equals("Registered"))

{

listitem=ldriver.findElement(lstitemRegistered);

}

**else** **if**(role.equals("Vendors"))

{

listitem=ldriver.findElement(lstitemVendors);

}

**else**

{

listitem=ldriver.findElement(lstitemGuests);

}

//listitem.click();

//Thread.sleep(3000);

JavascriptExecutor js = (JavascriptExecutor)ldriver;

js.executeScript("arguments[0].click();", listitem);

}

**public** **void** setManagerOfVendor(String value)

{

Select drp=**new** Select(ldriver.findElement(drpmgrOfVendor));

drp.selectByVisibleText(value);

}

**public** **void** setGender(String gender)

{

**if**(gender.equals("Male"))

{

ldriver.findElement(rdMaleGender).click();

}

**else** **if**(gender.equals("Female"))

{

ldriver.findElement(rdFeMaleGender).click();

}

**else**

{

ldriver.findElement(rdMaleGender).click();//Default

}

}

**public** **void** setFirstName(String fname)

{

ldriver.findElement(txtFirstName).sendKeys(fname);

}

**public** **void** setLastName(String lname)

{

ldriver.findElement(txtLastName).sendKeys(lname);

}

**public** **void** setDob(String dob)

{

ldriver.findElement(txtDob).sendKeys(dob);

}

**public** **void** setCompanyName(String comname)

{

ldriver.findElement(txtCompanyName).sendKeys(comname);

}

**public** **void** setAdminContent(String content)

{

ldriver.findElement(txtAdminContent).sendKeys(content);

}

**public** **void** clickOnSave()

{

ldriver.findElement(btnSave).click();

}

}

**public** WebDriver driver;

**public** Add\_New\_customer\_Page(WebDriver d) {

driver = d;

PageFactory.*initElements*(d, **this**);

}

@FindBy(xpath = "//a[@href='#']//p[contains(text(),'Customers')]")

WebElement lnk\_Customers\_Menu;

@FindBy(xpath = "//a[@href='/Admin/Customer/List']//p[contains(text(),'Customers')]")

WebElement lnk\_Customers\_Menu\_Item;

@FindBy(xpath = "//a[@class='btn btn-primary']")

WebElement btn\_Addnew;

@FindBy(xpath = "//input[@id='Email']")

WebElement txt\_Email;

@FindBy(xpath = "//input[@id='Password']")

WebElement txt\_Password;

@FindBy(xpath = "//input[@id='FirstName']")

WebElement txt\_FirstName;

@FindBy(xpath = "//input[@id='LastName']")

WebElement txt\_LastName;

@FindBy(xpath = "//input[@id='Gender\_Male']")

WebElement RadioButton\_Male;

@FindBy(xpath = "//input[@id='Gender\_Female']")

WebElement RadioButton\_Female;

@FindBy(xpath = "//input[@id='DateOfBirth']")

WebElement txt\_DOB;

@FindBy(xpath = "//input[@id='Company']")

WebElement txt\_CompanyName;

@FindBy(xpath = "//input[@id='IsTaxExempt']")

WebElement checkBox\_TaxExempt;

@FindBy(xpath = "//input[@class='k-input k-readonly']")

WebElement Dropdown\_NewsLetter;

@FindBy(xpath = "//li[normalize-space()='Your store name']")

WebElement Newsletter\_YourStoreName;

@FindBy(xpath = "//option[normalize-space()='Test store 2']")

WebElement Newsletter\_TestStore2;

@FindBy(xpath = "//div[@class='k-widget k-multiselect k-multiselect-clearable k-state-hover k-state-border-up']//div[@role='listbox']")

WebElement txt\_CustomerRoles;

@FindBy(xpath = "//select[@id='VendorId']")

WebElement Dropdown\_Vendors;

@FindBy(xpath = "//span[@title='delete']")

WebElement Dropdown\_OptionDelete;

@FindBy(xpath = "//li[contains(text(),'Vendors')]")

WebElement listItem\_Vendors;

@FindBy(xpath = "//li[text()='Guests']")

WebElement listItem\_Guests;

@FindBy(xpath = "//li[text()='Registered']")

WebElement listItem\_Registered;

@FindBy(xpath = "//li[text()='Forum Moderators']")

WebElement listItem\_Forum\_Moderators;

@FindBy(xpath = "//li[text()='Administrators']")

WebElement listItem\_Administrators;

@FindBy(xpath = "//select[@id='VendorId']")

WebElement ManagerOf\_Vendor;

@FindBy(xpath = "//input[@id='Active']")

WebElement checkBox\_Active;

@FindBy(xpath = "//textarea[@id='AdminComment']")

WebElement txt\_Admin\_Comments;

@FindBy(xpath = "//a[@class='btn btn-primary']")

WebElement btn\_Save;

// Operation Methods.

**public** String getPageTitle() {

**return** driver.getTitle();

}

**public** **void** clickOnCustomerMenu() {

lnk\_Customers\_Menu.click();

}

**public** **void** clickOnCustomerMenuItem() {

lnk\_Customers\_Menu\_Item.click();

}

**public** **void** clickOnAddNew() {

btn\_Addnew.click();

}

**public** **void** setEmail(String email) {

txt\_Email.sendKeys(email);

}

**public** **void** setPassword(String password) {

txt\_Password.sendKeys(password);

}

**public** **void** setFirstName(String firstname) {

txt\_FirstName.sendKeys(firstname);

}

**public** **void** setLastName(String lastname) {

txt\_FirstName.sendKeys(lastname);

}

**public** **void** setGender(String gender) {

**if** (gender.equals("Male")) {

RadioButton\_Male.click();

} **else** **if** (gender.equals("Female")) {

RadioButton\_Female.click();

} **else** {

RadioButton\_Male.click();// Default

}

}

**public** **void** setDob(String dob) {

txt\_DOB.sendKeys(dob);

}

**public** **void** setCompanyName(String comname) {

txt\_CompanyName.sendKeys(comname);

}

**public** **void** isTaxExempt(String yes\_no) {

**if** (yes\_no.equals("yes")) {

checkBox\_TaxExempt.click();

}

}

**public** **void** setNewsletter(String newsletter) {

Dropdown\_NewsLetter.click();

**if** (newsletter.equals("Your store name")) {

Newsletter\_YourStoreName.click();

} **else**

Newsletter\_TestStore2.click();// Default

}

**public** **void** setCustomerRoles(String role) **throws** InterruptedException {

Dropdown\_OptionDelete.click();

WebElement listitem;

Thread.*sleep*(3000);

Dropdown\_Vendors.click();

**if** (role.equals("Administrators")) {

listitem = listItem\_Administrators;

} **else** **if** (role.equals("Guests")) {

listitem = listItem\_Guests;

} **else** **if** (role.equals("Registered")) {

listitem = listItem\_Registered;

} **else** **if** (role.equals("Vendors")) {

listitem = listItem\_Vendors;

} **else** {

listitem = listItem\_Forum\_Moderators;

}

// listitem.click();

// Thread.sleep(3000);

JavascriptExecutor js = (JavascriptExecutor) driver;

js.executeScript("arguments[0].click();", listitem);

}

**public** **void** setManagerOfVendor(String value) {

Select drp = **new** Select(ManagerOf\_Vendor);

drp.selectByVisibleText(value);

}

**public** **void** setAdminComment(String content)

{

txt\_Admin\_Comments.sendKeys(content);

}

**public** **void** clickOnSave()

{

btn\_Save.click();

}

@Then("User can view Dashboard")

**public** **void** user\_can\_view\_dashboard() {

addCustpg = **new** Add\_New\_customer\_Page(driver);

Assert.*assertEquals*(addCustpg.getPageTitle(), "Dashboard / nopCommerce administration");

}

@When("User click on customers Menu")

**public** **void** user\_click\_on\_customers\_menu() {

addCustpg.clickOnCustomerMenu();

}

@When("click on customers Menu Item")

**public** **void** click\_on\_customers\_menu\_item() {

addCustpg.clickOnCustomerMenuItem();

}

@When("Click on Add New button")

**public** **void** click\_on\_add\_new\_button() {

addCustpg.clickOnAddNew();

}

@Then("User can view Add new customer page")

**public** **void** user\_can\_view\_add\_new\_customer\_page() {

Assert.*assertEquals*("Add a new customer / nopCommerce administration", addCustpg.getPageTitle());

}

@When("User enter customer info")

**public** **void** user\_enter\_customer\_info() **throws** InterruptedException {

String email = *randomestring*() + "@gmail.com";

addCustpg.setEmail(email);

addCustpg.setPassword("test123");

addCustpg.setCustomerRoles("Guest");

Thread.*sleep*(3000);

addCustpg.isTaxExempt("yes");

addCustpg.setNewsletter("Your store name");

addCustpg.setManagerOfVendor("Vendor 2");

addCustpg.setGender("Male");

addCustpg.setFirstName("John");

addCustpg.setLastName("Jhonson");

addCustpg.setDob("1/01/2000"); // Format: D/MM/YYY

addCustpg.setCompanyName("QA.pvt.ltd");

addCustpg.setAdminComment("This is for testing.........");

}

@When("Click on Save button")

**public** **void** click\_on\_save\_button() {

addCustpg.clickOnSave();

}

@Then("User can view confirmation message {string}")

**public** **void** user\_can\_view\_confirmation\_message(String string) {

Assert.*assertTrue*(driver.findElement(By.*tagName*("body")).getText()

.contains("The new customer has been added successfully"));

}

}