**Variables -1**

1. **Variable in Java** is a piece of memory that stores the values during Java program execution.
2. Variables are stored in RAM memory. RAM stands for Random Access Memory.
3. Variable value can be changed during program execution. Here 'vary' word represent variation, it means we can change the value of a variable during program execution.
4. Every variable has a assigned data type, which tells that variables holds which kind of value i.e. int, float, char, String, etc.
5. **Variable Declaration:** To declare a variable, you must specify the data type & give the variable a unique name.
6. It is mandatory to specify a data type of a variable before variable name. in java. It is used by compiler to help programmer avoid any mistakes such as storing String values to integer variables.
7. **Variable Initialization:** When we assign a value to a variable first time, it is known as variable initialization. Before initialization objects have null value and primitive types have default values such as 0 or false.
8. **We can declare and initialize the variable at same time.**
9. **Rules for naming a java variables**
10. Java Variable names are case sensitive. Variable declared as name is different than Name or NAME.
11. Variable name must start with a letter, or $ or \_ character. No other special character or digits are allowed as first character of variable name.
12. After the first character, java variable name can have digits as well along with letter, $ and \_.
13. Variable name cannot be equal to reserved keyword in java. For example, you cannot give your variable name as int, if or boolean as they are reserved keyword in java.

**Variables - 2**

1. **There are 3 types of variable in java**

* local variable
* Instance variable
* static variable

1. **Local Variable / Block Variable**

A variable declared inside the body of the method is called local variable. You can use this variable only within that method and the other methods don’t have access to that variable. A local variable cannot be defined with "static" keyword.

#### Global Variable / Instance Variable

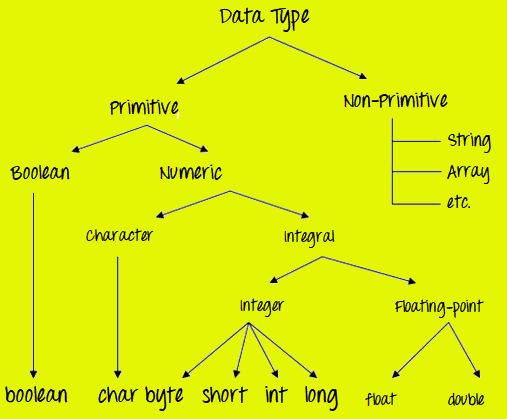
A variable declared inside the class but outside the body of the method, is called an instance variable. It is not declared as [static](https://www.javatpoint.com/static-keyword-in-java). It is called an instance variable because its value is instance-specific / Object Specific and is not shared among instances.

#### Static variable

A variable that is declared as static is called a static variable. It cannot be local. You can create a single copy of the static variable and share it among all the instances of the class. Memory allocation for static variables happens only once when the class is loaded in the memory.

Static variables are initialized only once, at the start of the program execution. These variables should be initialized first, before the initialization of any instance variables.

**Data Types**



* 1. Data type of a variable is an attribute which tells what kind of data that variable can have. Every java variable takes up a certain amount of space in memory. How much memory a variable takes is depends on its data type.
  2. There are two subtypes of data type In java :
* **Primitive Data Types** :- integer, character, Boolean, and float
* **Non-primitive Data Types**: - which include classes, arrays and interfaces.

### Primitive Data Types

Primitive Data Types are predefined and available within the Java language. Primitive values do not share state with other primitive values. There are 8 primitive types: byte, short, int, long, char, float, double, and Boolean.

* **Non-primitive data types:**

 The non-primitive data types include [Classes](https://www.javatpoint.com/object-and-class-in-java), [Interfaces](https://www.javatpoint.com/interface-in-java), and [Arrays](https://www.javatpoint.com/array-in-java).

* **Wrapper Classes**

There are many cases where we cannot directly use primitive data types. For example, We cannot put primitives into Java collections since Java collections (Lists, Sets etc.) can only store objects.  
  
Wrapper classes are classes provided by java programming language that enable us to wrap primitive data in Objects. Each of the eight primitive data types has a corresponding Wrapper class as listed in below table.

### What is Autoboxing and Unboxing?

 🡺

* 1. Autoboxing is the automatic conversion of primitive data types into their corresponding wrapper classes by Java compiler. Java compiler applies autoboxing when a primitive data type is assigned to a variable of the corresponding wrapper class, or a primitive data type is passed as an argument to a method which expects an object of the corresponding wrapper class.
  2. Unboxing is the automatic conversion of an object of a wrapper type to its corresponding primitive value by Java compiler. Java compiler applies urnboxing when a wrapper object is assigned to a variable of corresponding primitive type, or when a wrapper object is passed as an argument to a method which expects corresponding primitive type.

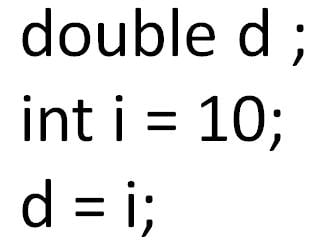
======================================================================

Casting

Converting one type of information into another type

1. **Implicit Casting**
   1. converting lower type of info into higher type of data.
   2. no data loose.

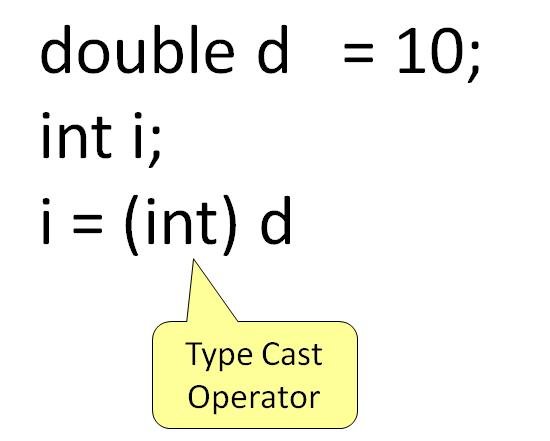
**Case 1)**Variable of smaller capacity is be assigned to another variable of bigger capacity.



This process is Automatic, and non-explicit is known as **Conversion.**

***---------------------------------------------------------------------------------------------------------------------------------*Explicit Casting**

**Case 2)**Variable of larger capacity is be assigned to another variable of smaller capacity



In such cases, you have to explicitly specify the **type cast operator. This process is known as Type Casting.**

In case, you do not specify a type cast operator; the compiler gives an error. Since this rule is enforced by the compiler, it makes the programmer aware that the conversion he is about to do may cause some loss in data and prevents **accidental losses.**

**Final**

1. In Java, the final keyword is used to denote constants. It can be used with variables, methods, and classes.
2. A **final keyword in java** can be used with a class, with a variable and with a method. final keyword restricts the further modification. When you use final keyword with an entity (class or variable or method), it gets the meaning that entity is complete and cannot be modified further.
3. A final variable that have no value it is called blank final variable or uninitialized final variable.

Once any entity (variable, method or class) is declared final, it can be assigned only once. That is,

* A. the final variable cannot be reinitialized with another value.
* B. the final method cannot be overridden.
* C. the final class cannot be extended.

### static blank final variable

A static final variable that is not initialized at the time of declaration is known as static blank final variable. It can be initialized only in static block.

* Q & A

### Is final method inherited?

**🡺Yes, final method is inherited but you cannot override it. (we can just overload it). i.e we can only call that method we can’t change the implementation of that method.**

### Can we initialize blank final variable?

**🡺Yes, but only in constructor.**

### Can we declare a constructor final?

### 🡺No, because constructor is never inherited.

* + - 1. **What is the use of final keyword in java?**

**🡺 final keyword in java is used to make any class or a method or a field as unchangeable. You can’t extend a final class, you can’t override a final method and you can’t change the value of a final field. final keyword is used to achieve high level of security while coding.**

* + - 1. **Can we change the state of an object to which a final reference variable is pointing?**

**🡺Yes, we can change the state of an object to which a final reference variable is pointing, but we can’t re-assign a new object to this final reference variable.**

* + - 1. **How do you change the state of an object in Java?**

**🡺 Generally, the state of an object (once it is created) can be changed using the setter (or mutator) methods exposed by the class of that object. The state can also be changed using the data members of the class if they are not declared private.**

* + - 1. **What is the main difference between abstract methods and final methods?**

**🡺Abstract methods must be overridden in the sub classes and final methods are not at all eligible for overriding.**

* + - 1. **What is the use of final class?**

**🡺A final class is very useful when you want a high level of security in your application. If you don’t want inheritance of a particular class, due to security reasons, then you can declare that class as a final.**

* + - 1. **Why must a final variable be initialized before constructor completes?**

**🡺A property of the final keyword is that it ensures that a variable cannot change state after it has been initialized. It forces the user to initialize when it is declared or in a constructor.**

* + - 1. **Can we change the value of an interface field? If not, why?**

**🡺No, we can’t change the value of an interface field. Because interface fields, by default, are final and static. They remain constant for whole execution of a program.**

* + - 1. **Where all we can initialize a final non-static global variable if it is not initialized at the time of declaration?**

**🡺In all constructors or in any one of instance initialization blocks.**

* + - 1. **Where all we can initialize a final static global variable if it is not initialized at the time of declaration?**

**🡺 In any one of static initialization blocks.**

* + - 1. **Can we use non-final local variables inside a local inner class?**

**🡺No. Only final local variables can be used inside a local inner class.**

* + - 1. **Can we declare constructors as final?**

**🡺No, constructors can not be final.**

* + - 1. **why constructor cannot be declared as final?**

**🡺The child class inherits all the members of the superclass except the constructors. In other words, constructors cannot be inherited in Java therefore you cannot override constructors. So, writing final before constructors makes no sense. Therefore, java does not allow final keyword before a constructor.**

## 10 Points Every Java Programmer Should Know About final Keyword In Java :

**1)** Any class or any method can be either **abstract or final** but not both. abstract and final are totally opposite. Because, abstract class or abstract method must be implemented or modified in the sub classes but final does not allow this. This creates an ambiguity.

    //The following class gives compile time error

final abstract class AnyClass

{

    //Any class can not be final and abstract

    final abstract void methodOne();

    //method can not be final and abstract at a time

}

**2)** final method can be overloaded and that overloaded method can be overridden in the sub class.

class SuperClass

{

    final void methodOne()

    {

        //final method

    }

    void methodOne(int i)

    {

        //final method can be overloaded

    }

}

class SubClass extends SuperClass

{

    @Override

    void methodOne(int i)

    {

        //Overloaded method can be overridden

    }

}

**3)** final variable cannot be re-initialized but final variable can be used to initialize other variables.

    class AnyClassOne

{

    final int i = 10;

    void methodOne()

    {

        i++;

        //above statement gives Compile time error.

        //value of final variable can not be changed

        int j = i;        //final variable can be used to initialize other variables.

        System.out.println(i);  //final variable can be used

    }

}

**4)** When an array reference variable is declared as final, only variable itself is final but not the array elements.

public class UseOfFinalKeyword

{

    public static void main(String[] args)

    {

        final int X[] = new int[10];     //final array variable

        X[2] = 10;

        X[2] = 20;     //Array element can be re-assigned

        X = new int[30];  //compile time error

        //can't re-assign new array object to final array variable

    }

}

**5)** When a reference variable is declared as final, you can’t re-assign a new object to it once it is referring to an object. But, you can change the state of an object to which final reference variable is referring.

class A

{

    int i = 10;

}

public class UseOfFinalKeyword

{

    public static void main(String[] args)

    {

        final A a = new A();  //final reference variable

        a.i = 50;

        //you can change the state of an object to which final reference variable is pointing

        a = new A();  //compile time error

        //you can't re-assign a new object to final reference variable

    }

}

**6)** Static variables, non-static variables and local variables all can be final. once the final variables are initialized, even you can’t re-assign the same value.

class A

{

    static final int i = 10;   //final static variable

    final int j = 20;          //final non-static variable

    void methodOne(final int k)

    {

        //k is final local variable

        k = 20;   //compile time error

    }

}

public class UseOfFinalKeyword

{

    public static void main(String[] args)

    {

        A a = new ();

        a.i = 10;     //Compile time error

        a.j = 20;     //even you can't assign same value to final variables

        a.methodOne(20);

    }

}

**7)** If the global variables are not initialized explicitly, they get default value at the time of object creation. But final global variables don’t get default value and they must be explicitly initialized at the time of object creation. Uninitialized final field is called **Blank Final Field**.

class A

{

    int i;   //Non-final global variable, no need to initialize them

    final int j;         //Blank Final Field

    A()

    {

        j=20;

        //final global variable must get a value at the time of object creation.

    }

}

public class UseOfFinalKeyword

{

    public static void main(String[] args)

    {

        A a = new A();

    }

}

**8)** final non-static global variable must be initialized at the time of declaration or in all constructors or in any one of IIBs – Instance Initialization Blocks.

class A

{

    final int i;  //Final non-static global variable may be initialized here  OR

  //may be initialized in any one of IIB's,

    // because while object creation, all IIBs are called.  OR

    {

        i = 30;

    }

    {

        //i = 40;

    }

  //must be initialized in all constructors.

    //because while object creation, only one constructor is called

    A()

    {

        //i=20;

    }

    A(int j)

    {

       // i=j;

    }

    A(int j, int k)

    {

       // i = 50;

    }

}

**9)** final static global variable must be initialized at the time of declaration or in any one of SIBs – Static Initialization Blocks. (final static global variable can’t be initialized in constructors)

    class A

{

    static final int i;   //final static global variable may be initialized here OR

    //may be initialized in any one of SIBs.

    static

    {

        i = 30;

    }

    static

    {

        //i = 40;

    }

    //final static global variable can not be initialized in constructors

    A()

    {

        //i=20;

    }

    A(int j)

    {

        //i=j;

    }

    A(int j, int k)

    {

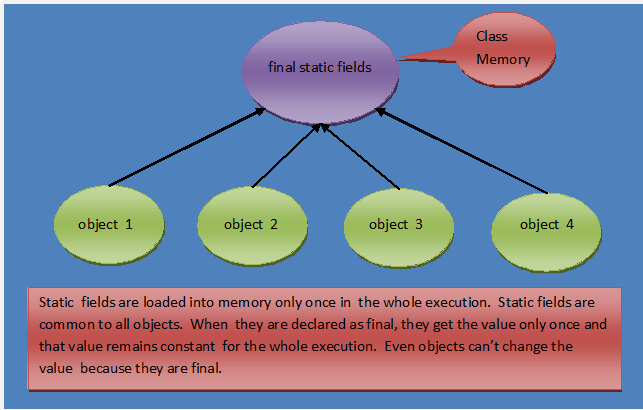
        //i = 50;

    }

}

**10)** The global variable which is declared as final and static remains unchanged for the whole execution.

Because, Static members are stored in the class memory and they are loaded only once in the whole execution. They are common to all objects of the class. If you declare static variables as final, any of the objects can’t change their value as it is final. Therefore, variables declared as final and static are sometimes referred to as **Constants**. All fields of interfaces are referred as constants, because they are final and static by default.



**Static keyword**

[**https://www.softwaretestinghelp.com/java/static-in-java/**](https://www.softwaretestinghelp.com/java/static-in-java/)

* The **static keyword** in [Java](https://www.javatpoint.com/java-tutorial) is used for memory management mainly. We can apply static keyword with [variables](https://www.javatpoint.com/java-variables), methods, blocks and [nested classes](https://www.javatpoint.com/java-inner-class). The static keyword belongs to the class than an instance of the class.
* **The static can be:**

1. Variable (also known as a class variable)
2. Method (also known as a class method)
3. Block
4. Nested class

* When a member is declared static, then it can be accessed without using an object. This means that before a class is instantiated, the static member is active and accessible. Unlike other non-static class members that cease to exist when the object of the class goes out of scope, the static member is still obviously active.

## Java static variable

🡺 If you declare any variable as static, it is known as a static variable. It is also called as the “Class variable”. Once the variable is declared as static, memory is allocated only once and not every time when a class is instantiated. Hence you can access the static variable without a reference to an object. The static variable can be used to refer to the common property of all objects (which is not unique for each object), for example, the company name of employees, college name of students, etc.

## Java static method

If you apply static keyword with any method, it is known as static method.

* A static method belongs to the class as against other non-static methods that are invoked using the instance of a class.
* To invoke a static method, you don’t need a class object.
* The static data members of the class are accessible to the static method. The static method can even change the values of the static data member.
* A static method cannot have a reference to ‘this’ or ‘super’ members. Even if a static method tries to refer them, it will be a compiler error.
* Just like static data, the static method can also call other static methods.
* A static method cannot refer to non-static data members or variables and cannot call non-static methods too.
* Overloading And Overriding Of Static Method
  1. **Overloading**

You can overload a static method in Java with different parameter lists but with the same name.

One point to note is that you cannot overload the method merely depending on the ‘static’ keyword. **For. Example,**if you have an instance method ‘sum’ and if you define another method “sum” and declare it as static, then it is not going to work. This attempt to overload based on a “static” keyword is going to result in a compilation failure

1. **Overriding**
   * 1. As static methods are invoked without any object of the class, even if you have a static method with the same signature in the derived class, it will not be overriding. This is because there is no run-time polymorphism without an instance.
     2. Hence you cannot override a static method. But if at all there is a static method with the same signature in the derived class, then the method to call doesn’t depend on the objects at run time but it depends on the compiler.
     3. You have to note that though static methods cannot be overridden, the Java language does not give any compiler errors when you have a method in the derived class with the same signature as a base class method.

## 3) Java static block

* **Is used to initialize the static data member.**
* **It is executed before the main method at the time of class loading.**

Just as you have function blocks in programming languages like C++, C#, etc. in Java also, there is a special block called “static” block that usually includes a block of code related to static data.

This static block is executed at the moment when the first object of the class is created (precisely at the time of classloading) or when the static member inside the block is used.

### Q) Can we execute a program without main() method?

**🡺 No, one of the ways was the static block, but it was possible till JDK 1.6. Since JDK 1.7, it is not possible to execute a Java class without the**[**main method**](https://www.javatpoint.com/java-main-method)**.**

## 4) Java static Class

1. In Java, you have static blocks, static methods, and even static variables. Hence it’s obvious that you can also have static classes. In Java, it is possible to have a class inside another class and this is called a Nested class. The class that encloses the nested class is called the Outer class.
2. In Java, although you can declare a nested class as Static it is not possible to have the outer class as Static.

**Static Nested Class In Java**

1. As already mentioned, you can have a nested class in Java declared as static. The static nested class differs from the non-static nested class(inner class) in certain aspects as listed below.
2. Unlike the non-static nested class, the nested static class doesn’t need an outer class reference.
3. A static nested class can access only static members of the outer class as against the non-static classes that can access static as well as non-static members of the outer class.

### 5) Static Import In Java

As you know, we usually include various packages and predefined functionality in the Java program by using the “import” directive. Using the word static with the import directive allows you to use the class functionality without using the class name.

**Example:**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| import static java.lang.System.\*;  class Main {       public static void main(String[] args)      {          //here we import System class using static, hence we can directly use functionality         out.println("demonstrating static import");        }  }  **Enlisted below are the differences between** **Static and Non-Static variables**.   | **Static Variables** | **Non-static Variables** | | --- | --- | | It can be accessed using class name only. | Requires objects of a class to access. | | Are accessible to both static as well as non-static methods. | Are accessible to non-static methods only. | | A memory for static variable is allocated only once per class. | A memory for non-static variables is allocated per object. | | Shared by all the objects of the class. | A copy of variable per object is made. | | Has global scope and is available to all the methods and blocks. | Has local scope and is visible to objects of the class. |   **Given below is the difference between Static and Non-Static methods**.   | **Static Methods** | **Non-static Methods** | | --- | --- | | A method that is preceded by a static keyword and is available at the class level. | A method not preceded by static keyword and available for each of the instances of the class. | | Supports compile-time or early binding. | Supports run-time or dynamic binding. | | Can access only static data members of its class and any other class. | Can access static as well as non-static members of the class and other classes. | | Static methods cannot be overridden. | Can be overridden. | | Memory is allocated only once. Hence memory used is less. | Memory consumption is more since memory is allocated every time the method is invoked. | |

| **Static** | **Final** |
| --- | --- |
| A static data member (nested class, variable or method) is a data member preceded by static keyword and can be accessed without an object. | The final keyword can be applied to a variable, method, class,etc. and imposes restrictions on the entities. |
| Not mandatory to initialize the static variable with value during declaration. | It is required that the final variable be initialized to a value at the time of declaration |
| You can reinitialize the static variables. | Not possible to reinitialize final variables. |
| Static methods are those that can only access static members. | Final methods are the methods that cannot be inherited/overridden. |
| Static classes are classes whose objects cannot be created. | Final classes are classes that cannot be inherited. |

-----------------------------------------------------------------------------------------------------------------------------

**Q #3) Can a Static Class have a Constructor?**

**🡺** Yes, a static class can have a constructor and its purpose is solely to initialize static data members. It will be invoked only for the first time when the data members are accessed. It will not be invoked for subsequent access.

**Q #4) What is the use of Static Constructor?**

**🡺** In general, the constructor is used to initialize static data members. It is also used to perform operations/actions that need to be carried out only once.

**Q #5) Are static methods inherited in Java?**

**🡺** Yes, static methods in Java are inherited but are not overridden.

OOPS

OOps, concepts in java is to improve code readability and reusability by defining a Java program efficiently. The main principles of object-oriented programming are **abstraction, encapsulation, inheritance, and polymorphism**. These concepts aim to implement real-world entities in programs.

Abstraction

* [Abstraction in Java](https://www.scientecheasy.com/2020/05/java-abstraction.html/) is a technique by which we can hide the data that is not required to users. It hides all unwanted data so that users can work only with the required data.
* Another way, it shows only essential things to the user and hides the internal details, for example, sending SMS where you type the text and send the message. You don't know the internal processing about the message delivery.
* Abstraction lets you focus on what the [object](https://www.javatpoint.com/object-and-class-in-java) does instead of how it does it.

### Ways to achieve Abstraction

There are two ways to achieve abstraction in java:

1. Abstract class (0 to 100%)
2. Interface (100%)

**======================================================================================**

### Abstract class in Java

* + 1. A class which is declared as abstract is known as an **abstract class**. It can have abstract and non-abstract methods. It needs to be extended and its method implemented. It cannot be instantiated.
    2. To access the abstract class, it must be inherited from another class.
    3. An abstract class can have a data member, abstract method, method body (non-abstract method), constructor, and even main() method.
    4. The abstract class can also be used to provide some implementation of the [interface](https://www.javatpoint.com/interface-in-java). In such case, the end user may not be forced to override all the methods of the interface.

The main advantages of using abstract class are as follows:

* Abstract class makes programming better and more flexible by giving the scope of implementing abstract methods.
* Programmer can implement abstract method to perform different tasks depending on the need.
* We can easily manage code.

<https://www.javatpoint.com/abstract-class-in-java>

#### Points to Remember

* An abstract class must be declared with an abstract keyword.
* It can have abstract and non-abstract methods.
* It cannot be instantiated.
* It can have [constructors](https://www.javatpoint.com/java-constructor) and static methods also.
* It can have final methods which will force the subclass not to change the body of the method.

=====================================================================

### Abstract Method in Java

* + - 1. A method which is declared with abstract modifier and has no implementation (means no body) is called abstract method in java.
      2. It does not contain any body. It has simply a signature declaration followed by a semicolon. It has the following general form as given below.

**Questions**

* + - * 1. **What is the difference between abstract class and concrete class?**

🡺There are mainly two differences between an abstract class and concrete class. They are:

a) We cannot create an object of abstract class. Only objects of its non-abstract (or concrete) sub classes can be created.

b) It can have zero or more abstract methods that are not allowed in a non-abstract class (concrete class).

* + - * 1. **What is Abstract in Java?**

🡺Abstract is a non-access modifier in java that is applicable for classes, interfaces, methods, and inner classes.

* + - * 1. **Can abstract modifier applicable for variables?**

🡺No.

* + - * 1. **Can an abstract method be declared as static?**

🡺 No.

* + - * 1. **Can an abstract method be declared with private modifier?**

🡺 No, it cannot be private because the abstract method must be implemented in the child class. If we declare it as private, we cannot implement it from outside the class.

* + - * 1. **What is Concrete method in Java?**

🡺 A concrete method in Java is a method which has always the body. It is also called a complete method in java.

* + - * 1. **When to use Abstract class in Java?**

🡺 An abstract class can be used when we need to share the same method to all non-abstract sub classes with their own specific implementations.

* + - * 1. **When to use Abstract method in Java?**

🡺 An abstract method can be used

a) When the same method has to perform different tasks depending on the object calling it.  
b) When you need to be overridden in its non-abstract subclasses.

* + - * 1. **Is abstract class a pure abstraction in Java?**

🡺 No, It provides 0 to 100% abstraction.

* + - * 1. **Is it possible that an abstract class can have without any abstract method?**

🡺 Yes.

* + - * 1. **Can an abstract class have constructor?**

🡺 Yes.

* + - * 1. **Is abstract class allow to define private, final, static, and concrete methods?**

🡺Yes.

* + - * 1. **Is it possible to achieve multiple inheritance through abstract class?**

🡺 No.

* + - * 1. **Can we define an abstract method inside non-abstract class (concrete class)?**

🡺 No, we cannot define an abstract method in non-abstract class.

* + - * 1. **What will happen if we do not override all abstract methods in subclass?**  
           Or, what will happen if we do not provide implementation for all abstract methods in subclass?

🡺 Java compiler will generate compile time error. We will have to override all abstract methods in subclass.

* + - * 1. **What is the difference between Abstraction and Encapsulation?**

🡺 Abstraction hides the implementation details from users whereas, [encapsulation](https://www.scientecheasy.com/2020/07/encapsulation-in-java.html/) wraps (binds) data and code into a single unit.

* + - * 1. **Why abstract class has constructor even though you cannot create object?**

🡺We cannot create an object of abstract class but we can create an object of subclass of abstract class. When we create an object of subclass of an abstract class, it calls the constructor of subclass.

This subclass constructor has a super keyword in the first line that calls constructor of an abstract class. Thus, the constructors of an abstract class are used from constructor of its subclass.

If the abstract class doesn’t have constructor, a class that extends that abstract class will not get compiled.

* + - * 1. **Why final and abstract can not be used at a time?**

🡺Because, final and abstract are totally opposite in nature. A final class or method can not be modified further where as abstract class or method must be modified further. “final” keyword is used to denote that a class or method does not need further improvements. “abstract” keyword is used to denote that a class or method needs further improvements.

* + - * 1. **Can we instantiate a class which does not have even a single abstract methods but declared as abstract?**

🡺No, We can’t instantiate a class once it is declared as abstract even though it does not have abstract methods.

* + - * 1. **Can we declare abstract methods as private? Justify your answer?**

🡺No. Abstract methods can not be private. If abstract methods are allowed to be private, then they will not be inherited to sub class and will not get enhanced.

* + - * 1. **Can we declare abstract methods as static?**

🡺No, abstract methods can not be static.

* + - * 1. **Can a class contain an abstract class as a member?**

🡺Yes, a class can have abstract class as it’s member.

* + - * 1. **Abstract classes can be nested. True or false?**

🡺True. Abstract classes can be nested i.e an abstract class can have another abstract class as it’s member.

Encapsulation

* 1. The meaning of **Encapsulation**, is to make sure that "sensitive" data is hidden from users. To achieve this, you must:
* declare class variables/attributes as private
* provide public **get** and **set** methods to access and update the value of a private variable
  1. **Encapsulation in Java** is a process of wrapping code and data together into a single unit,
  2. private variables can only be accessed within the same class (an outside class has no access to it). However, it is possible to access them if we provide public **get** and **set** methods.

### Advantage of Encapsulation in Java

1. It provides you the **control over the data**.
2. It is a way to achieve **data hiding** in Java because other class will not be able to access the data through the private data members.
3. The encapsulate class is **easy to test**. So, it is better for unit testing.
4. The standard IDE's are providing the facility to generate the getters and setters. So, it is **easy and fast to create an encapsulated class** in Java.
5. Flexible: the programmer can change one part of the code without affecting other parts
6. Increased security of data

**======================================================================================Questions**

1. **What is data hiding in Java?**

🡺 An outside person cannot access our internal data directly or our internal data should not go out directly. This oops feature is called data hiding n Java. After validation or authentication, the outside person can access our internal data.

1. **How to achieve Data hiding programmatically?**

🡺 By declaring data members (variables) as private, we can achieve or implement data hiding. If the variables are declared as private in the class, nobody can access them from outside the class.

The biggest advantage of data hiding is we can achieve security.

1. **What is a Tightly encapsulated class in Java?**

🡺 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.

1. **What is the difference between Abstraction and Encapsulation?**

🡺 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.

# Inheritance

* + 1. The technique of creating a new class by using existing class functionality is called inheritance in Java. In other words, inheritance is a process where a child class acquires all the properties and behaviors of the parent class.
* **subclass** (child) - the class that inherits from another class
* **superclass** (parent) - the class being inherited from
  + 1. To inherit from a class, use the extends keyword.
    2. The idea behind inheritance in Java is that you can create new [classes](https://www.javatpoint.com/object-and-class-in-java) that are built upon existing classes. When you inherit from an existing class, you can reuse methods and fields of the parent class

### Why use inheritance in java

* For [Method Overriding](https://www.javatpoint.com/method-overriding-in-java) (so [runtime polymorphism](https://www.javatpoint.com/runtime-polymorphism-in-java) can be achieved).
* For Code Reusability.

**===================================================================================**

## Types of inheritance in java

On the basis of class, there can be three types of inheritance in java: single, multilevel and hierarchical.

In java programming, multiple and hybrid inheritance is supported through interface only.





## Q) Why multiple inheritances is not supported in java?

🡺

To reduce the complexity and simplify the language, multiple inheritance is not supported in java.

Consider a scenario where A, B, and C are three classes. The C class inherits A and B classes. If A and B classes have the same method and you call it from child class object, there will be ambiguity to call the method of A or B class.

Since compile-time errors are better than runtime errors, Java renders compile-time error if you inherit 2 classes. So whether you have same method or different, there will be compile time error.

## The final Keyword

If you don't want other classes to inherit from a class, use the final keyword:

1. **Why do we need to use inheritance?**

🡺 Inheritance is one of the main pillars of OOPs concept. Some objects share certain properties and behaviors. By using inheritance, a child class acquires all properties and behaviors of parent class.

There are the following reasons to use inheritance in java.

* We can reuse the code from the base class.
* Using inheritance, we can increase features of class or method by overriding.
* Inheritance is used to use the existing features of class.
* It is used to achieve runtime polymorphism i.e method overriding.

1. **What is Is-A relationship in Java?**

🡺 Is-A relationship represents Inheritance. It is implemented using the “extends” keyword. It is used for code reusability.

1. **Which class in Java is superclass of every other class?**

🡺 In Java, Object class is the superclass of every other class.

1. **Can a class extend itself?**

🡺 No, a class cannot extend itself.

1. **Can we assign superclass to subclass?**

🡺 No.

1. **Can a class extend more than one class?**

🡺 No, one class can extend only a single class.

1. **Are constructor and instance initialization block inherited to subclass?**

🡺 No, constructor and instance initialization block of the superclass cannot be inherited to its subclass but they are executed while creating an object of the subclass.

1. **Are static members inherited to subclass in Java?**

🡺 Static block cannot be inherited to its subclass.

A static method of superclass is inherited to the subclass as a static member and non-static method is inherited as a non-static member only.

1. **Can we extend (inherit) final class?**

🡺 No, a class declared with final keyword cannot be inherited.

1. **Can a final method be overridden?**

🡺 No, a final method cannot be overridden.

1. **Can we inherit private members of base class to its subclass?**

🡺 No.

1. **What is order of calling constructors in case of inheritance?**

**🡺 In case of inheritance, constructors are called from the top to down hierarchy.**

1. **What are the advantages of inheritance in Java?**

🡺 The advantages of inheritance in java are as follows:

* We can minimize the length of duplicate code in an application by putting the common code in the superclass and sharing it amongst several subclasses.
* Due to reducing the length of code, the redundancy of the application is also reduced.
* Inheritance can also make application code more flexible to change.

1. **How does Multiple inheritance implement in Java?**

🡺 Multiple inheritance can be implemented in Java by using interfaces. A class cannot extend more than one class but a class can implement more than one interface.

1. **Can we access subclass members if we create an object of superclass?**

🡺 No, we can access only superclass members but not the subclass members.

1. **Can we access both superclass and subclass members if we create an object of subclass?**

🡺 Yes, we can access both superclass and subclass members.

1. **What happens if both superclass and subclass have a field with the same name?**

🡺Only subclass members are accessible if an object of subclass is instantiated.

1. **Is interface inherited from the Object class?**

🡺 No.

**===================================================================================**

**Diamond Problem**

Java does not allow is multiple inheritance where one class can inherit properties from more than one class. It is known as the **diamond problem**

## The Solution of Diamond Problem

The solution to the diamond problem is **default methods** and **interfaces**. We can achieve multiple inheritance by using these two things.

# Polymorphism

=======================================================================

* 1. 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.
  2. 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.
  3. In other words, if a single object shows multiple forms or multiple behaviors, it is called polymorphism.

There are two types of polymorphism in java. They are:

* Static polymorphism (Compile time Polymorphism)
* Dynamic polymorphism (Runtime Polymorphism)

**===================================================================================**

**Questions**

1. **What are different ways to achieve or implement polymorphism in Java?**

🡺 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

1. **What are the advantages of Polymorphism?**

🡺 There are the following advantages of polymorphism in java:

a. Using polymorphism, we can achieve flexibility in our code because we can perform various operations by using methods with the same names according to requirements.

b. The main benefit of using polymorphism is when we can provide implementation to an abstract base class or an interface.

1. **What are the differences between Polymorphism and Inheritance in Java?**

🡺 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.

b. 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. How Java compiler differentiate between methods in Compile time Polymorphism?**

🡺 During compilation, Java compiler differentiates multiple methods having the same name by their signatures.

**5. Is it possible to implement runtime polymorphism by data members in Java?**

🡺 No, we cannot implement runtime polymorphism by data members in java.

**6. What are the differences between compile-time polymorphism and runtime polymorphism in java?**

🡺 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.

**7. What is Binding in Java?**

🡺 The connecting (linking) between a method call and method definition is called [binding in java](https://www.scientecheasy.com/2020/02/static-and-dynamic-binding-in-java.html/).

**8. What are the types of binding in Java?**

🡺 There are two types of binding in java. They are as follows:

a. Static Binding (also known as Early Binding). Compile time  
b. Dynamic Binding (also known as Late Binding).

**9. Why binding of private, static, and final methods are always static binding in Java?**

🡺 Static binding is better performance-wise because java compiler knows that all such methods cannot be overridden and will always be accessed by object reference variable.

Hence, the compiler doesn’t have any difficulty in binding between a method call and method definition. That’s why binding for such methods is always static.

# Exception Handling

* 1. **Exception in Java** is an event that interrupts the execution of program instructions and disturbs the normal flow of program execution.
  2. There are two types of errors:

1. Compile time errors
2. Runtime errors
   1. Exceptions in Java are handled using try, catch and finally blocks.
3. **try block :** The code or set of statements which are to be monitored for exception are kept in this block.
4. **catch block :** This block catches the exceptions occurred in the try block.
5. **finally block :** This block is always executed whether exception is occurred in the try block or not and occurred exception is caught in the catch block or not.
   1. **Exception Types** :
      1. **Checked exceptions** are the exceptions which are known to compiler. These exceptions are checked at compile time only. Hence the name checked exceptions. These exceptions are also called compile time exceptions. Because, these exceptions will be known during compile time.

(ClassNotFoundException, SQLException, IOException)

* 1. **Unchecked exceptions** are those exceptions which are not at all known to compiler. These exceptions occur only at run time. These exceptions are also called as run time exceptions. All sub classes of java.lang.RunTimeException and java.lang.Error are unchecked exceptions.

NullPointerException, ArrayIndexOutOfBoundsException, NumberFormatException

## throw In Java :

**throw** is a keyword in java which is used to throw an exception manually. Using throw keyword, you can throw an exception from any method or block. But, that exception must be of type **java.lang.Throwable** class

or it’s sub classes.

## throws In Java :

**throws** is also a keyword in java which is used in the method signature to indicate that this method may throw

mentioned exceptions. The caller to such methods must handle the mentioned exceptions either using try-catch blocks or using throws keyword.

## Throwable In Java :

**Throwable** is a super class for all types of errors and exceptions in java. This class is a member of **java.lang** package. Only instances of this class or it’s sub classes are thrown by the java virtual machine or by the throw statement. The only argument of catch block must be of this type or it’s sub classes. If you want to create your own customized exceptions, then your class must extend this class.

**Q) Can we keep the statements after finally block If the control is returning from the finally block itself?**

🡺No, it gives unreachable code error. Because, control is returning from the finally block itself. Compiler will not see the statements after it. That’s why it shows unreachable code error.

**Q) What is Re-throwing an exception in Java?**

🡺Exceptions raised in the try block are handled in the catch block. If it is unable to handle that exception, it can re-throw that exception using throw keyword. It is called re-throwing an exception.

## finally Vs finalize() :

one similarity between **finally block** and **finalize() method**. Both are used to close the resources used by the

program. finally block is used to close the resources soon after their use. finalize() method is used to close the resources before an object is removed from the memory. That means if you use finalize() method to close the resources, they will remain open until an object,  which is using them, is garbage collected.

But, using finalize() method to close the resources is less recommended as it is not guaranteed that garbage collector will always call finalize() method on an object before it is removed from the memory. If it is not called, the resources will remain open. Therefore, it is always good to close the resources soon after their use using finally block.

**24) Can we override a super class method which is throwing an unchecked exception with checked exception in the sub class?**

🡺No. If a super class method is throwing an unchecked exception, then it can be overridden in the sub class with same exception or any other unchecked exceptions but can not be overridden with checked exceptions.

**28) What is the use of printStackTrace() method?**

🡺printStackTrace() method is used to print the detailed information about the exception occurred.

**29) Give some examples to checked exceptions?**

🡺ClassNotFoundException, SQLException, IOException

**30) Give some examples to unchecked exceptions?**

🡺NullPointerException, ArrayIndexOutOfBoundsException, NumberFormatException

# Access Modifiers

* 1. There are two types of modifiers in Java: **access modifiers** and **non-access modifiers**.
  2. The access modifiers in Java specifies the accessibility or scope of a field, method, constructor, or class. We can change the access level of fields, constructors, methods, and class by applying the access modifier on it.

**=================================================================================**

There are four types of Java access modifiers:

**1. Private**: The access level of a private modifier is only within the class. It cannot be accessed from outside the class. If you make any class constructor private, you cannot create the instance of that class from outside the class.

a. Private variables and methods can be accessed only in the class they are declared.

b. Private variables and methods from SuperClass are NOT available in SubClass.

**------------------------------------------------------------------------------------------------------------------------------------------------**

**2. Default**: The access level of a default modifier is only within the package. It cannot be accessed from outside the package. If you do not specify any access level, it will be the default. It provides more accessibility than private. But, it is more restrictive than protected, and public.

a. Default variables and methods can be accessed in the same package Classes.

b. Default variables and methods from SuperClass are available only to SubClasses in same package.

**------------------------------------------------------------------------------------------------------------------------------------------**

**3. Protected**: The access level of a protected modifier is within the package and outside the package through child class. If you do not make the child class, it cannot be accessed from outside the package. The protected access modifier can be applied on the data member, method and constructor. It can't be applied on the class. a. Protected variables and methods can be accessed in the same package Classes.

b. Protected variables and methods from SuperClass are available to SubClass in any package

**--------------------------------------------------------------------------------------------------------------------------------------------**

**4. Public**: The access level of a public modifier is everywhere. It can be accessed from within the class, outside the class, within the package and outside the package. It has the widest scope among all other modifiers.

a. Public variables and methods can be accessed from every other Java classes.

b. Public variables and methods from SuperClass are all available directly in the SubClass

**------------------------------------------------------------------------------------------------------------------------------------------**

There are many non-access modifiers, such as static, abstract, synchronized, native, volatile, transient, etc. Here, we are going to learn the access modifiers only.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Access Modifier** | **within class** | **within package** | **outside package by subclass only** | **outside package** |
| **Private** | Y | N | N | N |
| **Default** | Y | Y | N | N |
| **Protected** | Y | Y | Y | N |
| **Public** | Y | Y | Y | Y |

### What happens when a variable is marked as volatile?

* Volatile can only be applied to instance variables.
* A volatile variable is one whose value is always written to and read from "main memory". Each thread has its own cache in Java. The volatile variable will not be stored on a Thread cache.

**CONSTRUCTOR In Modifiers**

* + - 1. Private: Yes, we can have a private constructor in Java.
* The private constructor is used when we do not want to create the object of that class.
* We cannot create a subclass of that class.
* It is also used in Singleton design and Factory method design patterns.
  + - 1. Default

the default constructor has the default access implied by no access modifier

* + - 1. Protected

A protected constructor means that only derived members can construct instances of the class (and derived instances) using that constructor. This sounds a bit chicken-and-egg, but is sometimes useful when implementing class factories. Technically, this applies **only if ALL constructors are protected**

* + - 1. Public

You make a constructor public if **you want the class to be instantiated from any where**. You make a constructor protected if you want the class to be inherited and its inherited classes be instantiated.

**=======================================================================**

* 1. **Which access modifiers can be used with a class?**

🡺 Public and Default access modifiers can be used with a class.

* 1. **Can we instantiate the object of derived class if the parent constructor is protected?**

🡺 No, we cannot instantiate the object of derived class if the parent constructor is protected.

* 1. **What are non-access modifiers in Java?**

🡺 There are four [non-access modifiers in Java](https://www.scientecheasy.com/2020/06/non-access-modifiers-in-java.html/). They are as follows:

* Static
* Final
* Abstract
* Synchronized

a) **Static:** This modifier is used to check that a member is a class member or instance member. If you declare a class as static, this class will be executed first.

b) **Final:** Final is a keyword that is used to restrict the users. In other words, it is used to restrict further modification of a class, field, or method. If a class is declared as ‘final’, the class cannot be subclassed.

c) **Abstract:** Abstract is a keyword that is used with a class or a method. An abstract class or abstract method is used for further modification. If a class is declared as ‘abstract’, the class cannot be instantiated.

d) **Synchronized:** It is used to achieve thread safeness. Only one thread can enter in a synchronized method or block at a given time.

**4. Can we declare a top-level class as private?**

🡺 No, we cannot declare a top-level or outer class as private. It can have either “public” or no modifier.

If you declare a top-level class as a private, the compiler will complain that the “modifier private is not allowed here” but an inner class can be private.

Inner class means class as a member of another class. The same is the case with protected.

**5. Can we declare an abstract method as private?**

🡺 No, an abstract method cannot be private. They must be declared as public, protected, or default so that they can be further modified.

**6. Can we declare a top-level class as protected?**

🡺 No, we cannot declare a class as protected. An inner class can be protected but not an outer class.

**7. Can a method or a class be final and abstract at the same time?**

🡺 No, it is not possible. A class or a method cannot be final or abstract at the same time because the final method or final class cannot be further modified whereas an abstract class or an abstract method must be modified further.

**8. What is the default access specifier for a class, an interface, and struct declared directly with a namespace?**

🡺 Internal

**9. What is access modifier for enumeration?**

🡺 Enumeration members are always public. No other access modifiers are allowed.

**10. Which access modifier is also known as Universal access modifier?**

🡺 Public

# This keyword

* + 1. There can be a lot of usage of **Java this keyword**. In Java, this is a **reference variable** that refers to the current object.
    2. The most common use of the this keyword is to eliminate the confusion between class attributes and parameters with the same name
    3. this can also be used to:
* [this can be used to refer current class instance variable.](https://www.javatpoint.com/this1)
* [this can be used to invoke current class method (implicitly)](https://www.javatpoint.com/this2)
* [this() can be used to invoke current class constructor.](https://www.javatpoint.com/this3)
* [this can be passed as an argument in the method call.](https://www.javatpoint.com/this4)
* [this can be passed as argument in the constructor call.](https://www.javatpoint.com/this5)
* [this can be used to return the current class instance from the method.](https://www.javatpoint.com/this6)

**Q.Can we use this in static methods?**  
  
🡺No we can not use this in static methods. if we try to use compile time error will come:Cannot use this in a static context

**Q.What are all the differences between this and super keyword?**

* This refers to current class object where as super refers to super class object
* Using this we can access all non static methods and variables. Using super we can access super class variable and methods from sub class.
* Using this(); call we can call other constructor in same class. Using super we can call super class constructor from sub class constructor.

# Super Keyword

* 1. The super keyword refers to superclass (parent) objects.
  2. It is used to call superclass methods, and to access the superclass constructor.
  3. The most common use of the super keyword is to eliminate the confusion between superclasses and subclasses that have methods with the same name.
  4. To understand the super keyword, you should have a basic understanding of [Inheritance](https://www.w3schools.com/java/java_inheritance.asp) and [Polymorphism](https://www.w3schools.com/java/java_polymorphism.asp).

## The use of super keyword

1) To access the data members of parent class when both parent and child class have member with same name  
2) To explicitly call the no-arg and parameterized constructor of parent class  
3) To access the method of parent class when child class has overridden that method.

### Can we use super keyword in static method of a sub class for calling parent class method?

🡺No, We cannot use super keyword in static methods because it belongs to the immediate parent object and static belongs to the class level.

# Collections

The **Collection in Java** is a framework that provides an architecture to store and manipulate the group of objects.

Java Collections can achieve all the operations that you perform on a data such as searching, sorting, insertion, manipulation, and deletion.

Java Collection means a single unit of objects. Java Collection framework provides many interfaces (Set, List, Queue, Deque) and classes ([ArrayList](https://www.javatpoint.com/java-arraylist), Vector, [LinkedList](https://www.javatpoint.com/java-linkedlist), [PriorityQueue](https://www.javatpoint.com/java-priorityqueue), HashSet, LinkedHashSet, TreeSet).

**===================================================================================**

What is a framework in Java

* It provides readymade architecture.
* It represents a set of classes and interfaces.
* It is optional.

**===================================================================================**

### Methods of Collection interface

There are many methods declared in the Collection interface. They are as follows:

|  |  |  |
| --- | --- | --- |
| **No.** | **Method** | **Description** |
| 1 | public boolean add(E e) | It is used to insert an element in this collection. |
| 2 | public boolean addAll(Collection<? extends E> c) | It is used to insert the specified collection elements in the invoking collection. |
| 3 | public boolean remove(Object element) | It is used to delete an element from the collection. |
| 4 | public boolean removeAll(Collection<?> c) | It is used to delete all the elements of the specified collection from the invoking collection. |
| 5 | default boolean removeIf(Predicate<? super E> filter) | It is used to delete all the elements of the collection that satisfy the specified predicate. |
| 6 | public boolean retainAll(Collection<?> c) | It is used to delete all the elements of invoking collection except the specified collection. |
| 7 | public int size() | It returns the total number of elements in the collection. |
| 8 | public void clear() | It removes the total number of elements from the collection. |
| 9 | public boolean contains(Object element) | It is used to search an element. |
| 10 | public boolean containsAll(Collection<?> c) | It is used to search the specified collection in the collection. |
| 11 | public Iterator iterator() | It returns an iterator. |
| 12 | public Object[] toArray() | It converts collection into array. |
| 13 | public <T> T[] toArray(T[] a) | It converts collection into array. Here, the runtime type of the returned array is that of the specified array. |
| 14 | public boolean isEmpty() | It checks if collection is empty. |
| 15 | default Stream<E> parallelStream() | It returns a possibly parallel Stream with the collection as its source. |
| 16 | default Stream<E> stream() | It returns a sequential Stream with the collection as its source. |
| 17 | default Spliterator<E> spliterator() | It generates a Spliterator over the specified elements in the collection. |
| 18 | public boolean equals(Object element) | It matches two collections. |
| 19 | public int hashCode() | It returns the hash code number of the collection. |



|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Iterator interface  |  | | --- | | Iterator interface provides the facility of iterating the elements in a forward direction only. |  Methods of Iterator interface There are only three methods in the Iterator interface. They are:   |  |  |  | | --- | --- | --- | | **No.** | **Method** | **Description** | | 1 | public boolean hasNext() | It returns true if the iterator has more elements otherwise it returns false. | | 2 | public Object next() | It returns the element and moves the cursor pointer to the next element. | | 3 | public void remove() | It removes the last elements returned by the iterator. It is less used. |  ---------------------------------------------------------------------------------------Iterable Interface The Iterable interface is the root interface for all the collection classes. The Collection interface extends the Iterable interface and therefore all the subclasses of Collection interface also implement the Iterable interface.  It contains only one abstract method. i.e.,   1. Iterator<T> iterator()   It returns the iterator over the elements of type T. Collection Interface The Collection interface is the interface which is implemented by all the classes in the collection framework. It declares the methods that every collection will have. In other words, we can say that the Collection interface builds the foundation on which the collection framework depends.  Some of the methods of Collection interface are Boolean add ( Object obj), Boolean addAll ( Collection c), void clear(), etc. which are implemented by all the subclasses of Collection interface.  **=======================================================================** List Interface  1. List interface is the child interface of Collection interface. It inhibits a list type data structure in which we can store the ordered collection of objects. It can have duplicate values. 2. List interface is implemented by the classes ArrayList, LinkedList, Vector, and Stack. 3. To instantiate the List interface, we must use : 4. List <data-type> list1= **new** ArrayList(); 5. List <data-type> list2 = **new** LinkedList(); 6. List <data-type> list3 = **new** Vector(); 7. List <data-type> list4 = **new** Stack(); 8. There are various methods in List interface that can be used to insert, delete, and access the elements from the list. 9. The classes that implement the List interface are given below.   **=====================================================================** ArrayList The ArrayList class implements the List interface. It uses a dynamic array to store the duplicate element of different data types. The ArrayList class maintains the insertion order and is non-synchronized. The elements stored in the ArrayList class can be randomly accessed.  **------------------------------------------------------------------------------------------------------------------------------------------** LinkedList LinkedList implements the Collection interface. It uses a doubly linked list internally to store the elements. It can store the duplicate elements. It maintains the insertion order and is not synchronized. In LinkedList, the manipulation is fast because no shifting is required.  **-------------------------------------------------------------------------------------------------------------------------------------------** Vector Vector uses a dynamic array to store the data elements. It is similar to ArrayList. However, It is synchronized and contains many methods that are not the part of Collection framework. **===================================================================================** Set Interface  1. Set Interface in Java is present in java.util package. It extends the Collection interface. It represents the unordered set of elements which doesn't allow us to store the duplicate items. We can store at most one null value in Set. Set is implemented by HashSet, LinkedHashSet, and TreeSet. 2. Set can be instantiated as: 3. Set<data-type> s1 = **new** HashSet<data-type>(); 4. Set<data-type> s2 = **new** LinkedHashSet<data-type>(); 5. Set<data-type> s3 = **new** TreeSet<data-type>(); 6. **===================================================================================**  HashSet HashSet class implements Set Interface. It represents the collection that uses a hash table for storage. Hashing is used to store the elements in the HashSet. It contains unique items.  **------------------------------------------------------------------------------------------------------------------------------------------** LinkedHashSet LinkedHashSet class represents the LinkedList implementation of Set Interface. It extends the HashSet class and implements Set interface. Like HashSet, It also contains unique elements. It maintains the insertion order and permits null elements.  **------------------------------------------------------------------------------------------------------------------------------------------** SortedSet Interface SortedSet is the alternate of Set interface that provides a total ordering on its elements. The elements of the SortedSet are arranged in the increasing (ascending) order. The SortedSet provides the additional methods that inhibit the natural ordering of the elements.  The SortedSet can be instantiated as:   1. SortedSet<data-type> set = **new** TreeSet(); 2. **-------------------------------------------------------------------------------------------------------------------------------------------**  TreeSet Java TreeSet class implements the Set interface that uses a tree for storage. Like HashSet, TreeSet also contains unique elements. However, the access and retrieval time of TreeSet is quite fast. The elements in TreeSet stored in ascending order. |

**-------------------------------------------------------------------------------------------------------------------------------------------------**

**Test Process**

Test process is the steps which we go through while testing our software.

1. **Test planning and control:** Here we define the objective or the goal of our test process
2. **Test Analysis and design:** Here we create all test conditions and test cases
3. **Test Implementation and execution:** it is a fundamental test process where actual work is done; here we execute test cases with the test data.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**There are 3 ways you can do testing.**

1. **Manual testing**: - This is the type of testing where we execute all the test cases manually without using any automation tool.

[**https://www.javatpoint.com/manual-testing**](https://www.javatpoint.com/manual-testing)

1. **Automated testing: -**Here we use test Scripts and testing tools to automate software testing, it executes the test scripts without any human interference

With the help of an [automation testing tool](https://www.javatpoint.com/automation-testing-tool), we can easily approach the test data, handle the test implementation, and compares the actual output against the expected outcome.

<https://www.javatpoint.com/automation-testing>

1. **Continuous testing** goes even further, applying the principles of automated testing in a scaled, continuous manner to achieve the most reliable test coverage for an enterprise. **here** The code is continuously developed, delivered, tested and deployed.

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**Different types of testing**

**-----------------------------------------------------------------------------------**

* [**Functional testing**](https://www.perfecto.io/functional-testing-web-mobile-apps)**:** - here we check the functionality of the software, we check that the function is going to work properly or not according to given requirement.

We test the main function of the system

<https://www.javatpoint.com/functional-testing>

--------------------------------------------------------------------------------------------------------

* [**Interactive testing**](https://www.perfecto.io/interactive-testing)**:** - Interactive testing, also known as manual testing, allows test managers to set up and distribute manual Test Cases for testers.

<https://www.google.com/search?q=%E2%80%A2+Interactive+testing&oq=%E2%80%A2%09Interactive+testing&aqs=chrome..69i57.303j0j4&sourceid=chrome&ie=UTF-8>

--------------------------------------------------------------------------------------------------------

* **Integration testing: -**In this testing, units or individual components of the software are tested in a group. The focus of the integration testing level is to expose defects at the time of interaction between integrated components or units.

<https://www.javatpoint.com/integration-testing>

--------------------------------------------------------------------------------------------------------

* [**Regression testing**](https://www.perfecto.io/regression-testing)**: -** Regression testing is a black box testing techniques. It is used to authenticate a code ***change in the software does not impact the existing functionality of the product***. Regression testing is making sure that the product works fine with new functionality, [bug](https://www.javatpoint.com/bug-in-software-testing) fixes, or any change in the existing feature. <https://www.javatpoint.com/regression-testing>

--------------------------------------------------------------------------------------------------------

* **Smoke testing: -** In the smoke testing, we only focus on the positive flow of the application and enter only valid data, not the invalid data. In smoke testing, we verify every build is testable or not.

**Also called as acceptance testing**

**Perform on initial builds**

<https://www.javatpoint.com/smoke-testing>

--------------------------------------------------------------------------------------------------------

* **Sanity testing: -** sanity testing is performed to make sure that all the defects have been solved also ensures that the modification in the code or functions does not affect the associated modules.

<https://www.javatpoint.com/sanity-testing>

--------------------------------------------------------------------------------------------------------

* **Security testing: -** Security testing is an integral part of software testing, which is used to discover the weaknesses, risks, or threats in the software application and also help us to stop the nasty attack from the outsiders and make sure the security of our software applications

<https://www.javatpoint.com/security-testing>

--------------------------------------------------------------------------------------------------------

* [**Non functional testing**](https://www.perfecto.io/blog/what-is-non-functional-testing)**: -**Non-functional testing is a type of software testing to test non-functional parameters such as reliability, **load test, performance** and accountability of the software

<https://www.javatpoint.com/non-functional-testing>

**============================================================**

* 1. **Performance testing:-** Due to multiple users accessing the software concurrently there May be some performance issue

To avoid these performance issues, the tester performs one round of performance testing.

<https://www.javatpoint.com/performance-testing>

**=======================================================**

There are different types of performance testing:

**=======================================================**

1. **Load Testing: -**  The load testing is the most important essential part of performance testing which is used to check the performance of an application by ***applying some load like less than or equal to the desired load*** is known as load testing. And the load is a quantity, which means it only focuses on the numbers of users.

<https://www.javatpoint.com/load-testing>

**=======================================================**

1. **Stress testing: -** The stress testing is testing, which checks the behavior of an application by ***applying load greater*** ***than the desired load.***

Stress testing tests suddenly increased traffic of the application.

<https://www.javatpoint.com/stress-testing>

**=======================================================**

1. **Scalability Testing: -** It is used to check an application's performance by ***increasing*** ***or decreasing the load in particular scales*** known as **scalability testing**. It is executed at a **hardware, software, or database level.**

**E.g.** a **web page** scalability testing depends on the number of users, CPU usage, and network usage. In contrast, scalability testing of a **web server depends on the number of requests processed.**

<https://www.javatpoint.com/scalability-testing>

**=======================================================**

**There are two types of Scalability Testing: -**

* **Upward scalability testing: -** The upward scalability testing is used to expand the number of users on a specific scale until we got a crash point. It is mainly used to **identify the maximum capacity of an application***.*
* **Downward scalability testing: -** When the load testing is not passed, we will use the downward scalability testing and then start **decreasing the number of users in a particular interval** until the goal is achieved.

**=======================================================**

1. **Stability Test: -** Checking the performance of an application by **applying the load for a particular duration of time** is known as **Stability Testing**.

<https://www.javatpoint.com/stability-testing>

**---------------------------------------------------------------------------------------------------------------**

* **Sanity testing: -** sanity testing is performed to make sure that all the defects have been solved also ensures that the modification in the code or functions does not affect the associated modules.

<https://www.javatpoint.com/sanity-testing>

**---------------------------------------------------------------------------------------------------------------**

* **Security testing: -** Security testing is an integral part of software testing, which is used to discover the weaknesses, risks, or threats in the software application and also help us to stop the nasty attack from the outsiders and make sure the security of our software applications

<https://www.javatpoint.com/security-testing>

* **Unit testing: -**Unit testing involves the testing of each unit or an individual component of the software application.

<https://www.javatpoint.com/unit-testing>

**---------------------------------------------------------------------------------------------------------------**

* **White box testing: -White Box Testing** is software testing technique in which internal structure, design and coding of software are tested

<https://www.guru99.com/white-box-testing.html>

**---------------------------------------------------------------------------------------------------------------**

* **Acce**[**ssibility testing**](https://www.perfecto.io/accessibility-testing)**:**  **accessibility testing** is widely used to check the application for **disabled persons** and make sure the developer will create the application which can be accessible by all types of users, like a regular user and physically challenged (color blindness, learning disabilities, and so on).

<https://www.javatpoint.com/accessibility-testing>

**---------------------------------------------------------------------------------------------------------------**

* **Acceptance testing: -** A testing technique performed to determine whether the software system has met the requirement specifications. The main purpose of this test is to evaluate the system's compliance with the business requirements and verify if it is has met the required criteria for delivery to end users.

<https://www.tutorialspoint.com/software_testing_dictionary/acceptance_testing.htm#:~:text=Acceptance%20testing%2C%20a%20testing%20technique,for%20delivery%20to%20end%20users>.

**---------------------------------------------------------------------------------------------------------------**

* **Black box testing**

we test the system without having the internal knowledge about the system.

<https://www.javatpoint.com/black-box-testing>

**---------------------------------------------------------------------------------------------------------------**

* **End-to-end testing**

End-to-end testing is a technique that tests the entire software product from beginning to end to ensure the application flow behaves as expected

**---------------------------------------------------------------------------------------------------------------**

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**SELENIUM**

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

* **Selenium is a web based automation tool, not used for desktop application and mobile app.**
* **It works on multiple OS as well as multiple browsers.**

**------------------------------------------------------------------------------------------**

**“Selenium can be used to automate functional tests and can be integrated with automation test tools such as Maven**, **Jenkins, to achieve continuous testing. It can also be integrated with tools such as TestNG**, & **JUnit for managing test cases and generating reports”**

**------------------------------------------------------------------------------------------**

* **Components of selenium are as follows: -**

**------------------------------------------------------------------------------------------**

* 1. **Selenium IDE** - it is a record and play based tool it is a browser plug-in, which only compatible with Mozilla Firefox.
  2. **Selenium RC** -   user is allowed to create their own test cases in user defined languages.
  3. **Selenium grid** - Selenium grid is the tool which is used for execution of script, it enables a simultaneous running of test in multiple Browsers and environment  it is used for parallel execution
  4. **Selenium WebDriver** - WebDriver is the most widely used automation tool for web applications and supports all major browsers, including Chrome, Firefox, Internet Explorer and edge

**------------------------------------------------------------------------------------------**

**------------------------------------------------------------------------------------------Locators ------------------------------------------------------------------------------------------**

**Selenium uses locators to find and match the elements of the web page ,the types of locator are as follows: -**

xpath:

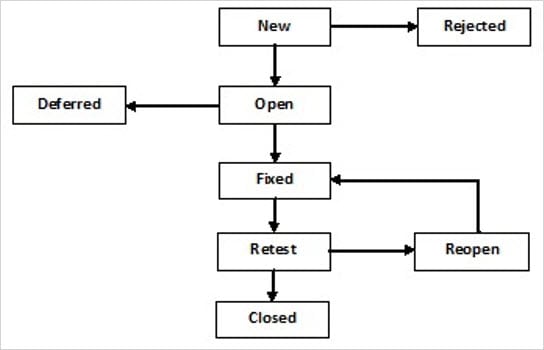
Sometimes, we may not identify the element using the locators such as id, class, name, etc. ... At times, XPath may change dynamically and we need to handle the elements while writing scripts. Standard way of writing xpath may not work and we need to write dynamic XPath in selenium scripts.

1. ID
2. Name
3. className
4. Tag Name
5. Link text
6. partial link text
7. CSS selector
8. Xpath

**------------------------------------------------------------------------------------------**

**Bug Life cycle**

**------------------------------------------------------------------------------------------**



Defect States

**------------------------------------------------------------------------------------------ 1) New**: This is the first state of a defect in the Defect Life Cycle. When any new defect is found, it falls in a ‘New’ state, and validations and testing are performed on this defect in the later stages of the Defect Life Cycle.

------------------------------------------------------------------------------------------------------------------------

**2) Assigned:** In this stage, a newly created defect is assigned to the development team for working on the defect. This is assigned by the project lead or the manager of the testing team to a developer.

------------------------------------------------------------------------------------------------------------------------

**3) Open:**Here, the developer starts the process of analyzing the defect and works on fixing it, if required. If the developer feels that the defect is not appropriate then it may get transferred to any of the below four states namely **Duplicate, Deferred, Rejected, or Not a Bug**-based upon the specific reason.

------------------------------------------------------------------------------------------------------------------------

**4) Fixed:**When the developer finishes the task of fixing a defect by making the required changes then he can mark the status of the defect as ‘Fixed’.

------------------------------------------------------------------------------------------------------------------------

**5) Pending Retest:**After fixing the defect, the developer assigns the defect to the tester for retesting the defect at their end, and till the tester works on retesting the defect, the state of the defect remains in ‘Pending Retest’.

------------------------------------------------------------------------------------------------------------------------

**6) Retest:**At this point, the tester starts the task of working on the retesting of the defect to verify if the defect is fixed accurately by the developer as per the requirements or not.

------------------------------------------------------------------------------------------------------------------------

**7) Reopen:**If any issue persists in the defect then it will be assigned to the developer again for testing and the status of the defect gets changed to ‘Reopen’.

------------------------------------------------------------------------------------------------------------------------

**8) Verified:**If the tester does not find any issue in the defect after being assigned to the developer for retesting and he feels that if the defect has been fixed accurately then the status of the defect gets assigned to ‘Verified’.

------------------------------------------------------------------------------------------------------------------------

**9) Closed:**When the defect does not exist any longer then the tester changes the status of the defect to ‘Closed’.

------------------------------------------------------------------------------------------------------------------------

**What is a Test Case?**

🡺A **TEST CASE** is a set of actions executed to verify a particular feature or functionality of your software application

------------------------------------------------------------------------------------------------------------------------

**What is a Test Scenario?**

🡺A Test Scenario is defined as any functionality that can be tested. It is a collective set of test cases which helps the testing team to determine the positive and negative characteristics of the project.

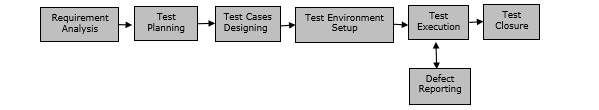
------------------------------------------------------------------------------------------------------------------------

**Difference between test cases and test scenarios**

**🡺**[**https://www.guru99.com/test-case-vs-test-scenario.html#:~:text=KEY%20DIFFERENCE-,Test%20Case%20is%20a%20set%20of%20actions%20executed%20to%20verify,functionality%20that%20can%20be%20tested.&text=Test%20Case%20includes%20test%20steps,end%20functionality%20to%20be%20tested**](https://www.guru99.com/test-case-vs-test-scenario.html#:~:text=KEY%20DIFFERENCE-,Test%20Case%20is%20a%20set%20of%20actions%20executed%20to%20verify,functionality%20that%20can%20be%20tested.&text=Test%20Case%20includes%20test%20steps,end%20functionality%20to%20be%20tested)

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**STLC**



* **Requirement Analysis** − When the SRD is ready and shared with the stakeholders, the testing team starts high level analysis concerning the AUT (Application under Test).
* **Test Planning** − Test Team plans the strategy and approach.
* **Test Case Designing** − Develop the test cases based on scope and criteria’s.
* **Test Environment Setup** − When integrated environment is ready to validate the product.
* **Test Execution** − Real-time validation of product and finding bugs.
* **Test Closure** − Once testing is completed, matrix, reports, results are documented.

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**We learned following things:**

1. **JSON**

JSON stands for JavaScript Object Notation. JSON is a lightweight format for storing and transporting data. JSON is often used when data is sent from a server to a web page. JSON is "self-describing" and easy to understand.

1. **Apache POI**

Apache POI is your Java Excel solution (for Excel 97-2008). We have a complete API for porting other OOXML and OLE2 formats and welcome others to participate. OLE2 files include most Microsoft Office files such as XLS, DOC, and PPT as well as MFC serialization API based file

1. **TestNG**

TestNG is a testing framework that is capable of making Selenium tests easier to understand and of generating reports that are easy to understand. The main advantages of TestNG over JUnit are the following. Annotations are easier to use and understand. Test cases can be grouped more easily

1. **Maven**

Maven" is really just a core framework for a collection of Maven Plugins. In other words, plugins are where much of the real action is performed, plugins are used to: create jar files, create war files, compile code, unit test code, create project documentation, and on and on.

1. **BDD cucumber**

Behavior Driven Development (BDD) framework is a software development process that is an offshoot of Test Driven Development (TDD) framework. BDD is an agile testing methodology. It is the process of development, based on test-driven development and domain-driven, object-oriented analysis

1. **SoapUI**

SoapUI is the world's leading Functional Testing tool for SOAP and REST testing. With its easy-to-use graphical interface, and enterprise-class features, SoapUI allows you to easily and rapidly create and execute automated functional, regression, and load tests

1. **Jenkins**

Jenkins is an open-source automation tool written in Java with plugins built for Continuous Integration purposes. Jenkins is used to build and test your software projects continuously making it easier for developers to integrate changes to the project, and making it easier for users to obtain a fresh build.

1. **GIT**

Git is the most commonly used version control system. Git tracks the changes you make to files, so you have a record of what has been done, and you can revert to specific versions should you ever need to. Git also makes collaboration easier, allowing changes by multiple people to all be merged into one source.

**9.POM**

Page Object Model, also known as POM, is a design pattern in Selenium that creates an object repository for storing all web elements. It is useful in reducing code duplication and improves test case maintenance. ... Using these elements, testers can perform operations on the website under test.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**AGILE**

Agile principles

1. Customer satisfies by early and continuous delivery of useful software

 2. Welcome changing requirements

 3. Working software is delivered frequently

 4. Daily Corporation between business people and developers

 5. Sustainable development

 6. Simplicity

Agile enables organizations to master continuous change. Agile methodology is a type of project management process, mainly used for software development, where demands and solutions evolve through the collaborative effort of self-organizing and cross-functional teams and their customers.

The Product Owner – Often an executive or key stakeholder, the Product Owner has a vision for the end product and a sense of how it will fit into the company’s long-term goals. This person will need to direct communication efforts, alerting the team to major developments and stepping in to course-correct and implement high-level changes as necessary.

The Scrum Master – The Scrum Master is most akin to a project manager. They are guardians of process, givers of feedback, and mentors to junior team members. They oversee day-to-day functions, maintain the Scrum board, check in with team members, and make sure tasks are being completed on target.

The Team Member – Team members are the makers: front- and back-end engineers, copywriters, designers, videographers, you name it. Team members have varied roles and skills but all are responsible for getting stuff done on time and in excellent quality.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**WATERFALL MODEL**

Waterfall Model followed in the sequential order and so we move to next step of development or testing if the previous step completed successfully. Waterfall Model is very successful approach for the small projects and if the requirements are very clear. In Waterfall Model, testing starts at the end when development work is completed

steps involved in waterfall model:

Requirements

Analysis

Design

Implementation

Testing

Deployment

Maintanence

**STLC -LIFE CYCLE?**

STLC stands for Software Testing Life Cycle. STLC is a sequence of different activities performed by the testing team to ensure the quality of the software or the product.

STLC is an integral part of Software Development Life Cycle (SDLC). But, STLC deals only with the testing phases.

There are 6 major phases of STLC −

Requirement Analysis − When the SRD is ready and shared with the stakeholders, the testing team starts high level analysis concerning the AUT (Application under Test).

Test Planning − Test Team plans the strategy and approach.

Test Case Designing − Develop the test cases based on scope and criteria’s.

Test Environment Setup − When integrated environment is ready to validate the product.

Test Execution − Real-time validation of product and finding bugs.

Test Closure − Once testing is completed, matrix, reports, results are documented.

**ATLC?**

Test Automation is applied with the structural sequence of testing process, that can have all the stages of testing life cycle with additional phases for including the test preparation and automation implementation processes. This entire process is known as Automation Testing Life Cycle. The various phases of ATLC are

‘define the scope of the test automation’

‘automation testing tool selection process depending on the type of functionality’,

‘designing test plan’

‘defining test strategy’,

‘setting up the test environments’,

‘designing and scripting the test cases,

‘execution of the test scripts’,

‘validating the test results’,

and finally ‘generating test reports’.

**What are TestNG Annotations?**

TestNG Annotations are used to control the next method to be executed in the test script. TestNG annotations are defined before every method in the test code. In case any method is not prefixed with annotations, it will be ignored and not be executed as part of the test code. To define them, methods need to be simply annotated with ‘@Test‘.

Types of TestNG Annotations

Below is the list of annotations that TestNG support in Selenium

BeforeSuite

BeforeTest

BeforeClass

BeforeMethod

Test Case

AfterMethod

AfterClass

AfterTest

AfterSuite

Let’s explore how these methods work.

@BeforeMethod: This will be executed before every @test annotated method.

@AfterMethod: This will be executed after every @test annotated method.

@BeforeClass: This will be executed before first @Test method execution. It will be executed one only time throughout the test case.

@AfterClass: This will be executed after all test methods in the current class have been run

@BeforeTest: This will be executed before the first @Test annotated method. It can be executed multiple times before the test case.

@AfterTest: A method with this annotation will be executed when all @Test annotated methods complete the execution of those classes inside the <test> tag in the TestNG.xml file.

@BeforeSuite: It will run only once, before all tests in the suite are executed.

@AfterSuite: A method with this annotation will run once after the execution of all tests in the suite is complete.

@BeforeGroups: This method will run before the first test run of that specific group.

@AfterGroups: This method will run after all test methods of that group complete their execution.

**Selenium Frameworks**

### **Linear Scripting Framework:**

This framework is based on the concept of record and playback mode that is always achieved in a linear manner. It is more commonly named as record and playback model.

Typically, in this scripting driven framework, the creation and execution of test scripts is done individually and this framework is an [effective way](https://www.testingxperts.com/blog/6-Most-Effective-Ways-to-Build-Business-Agility) to get started for enterprises.

The automation scripting is done in an incremental manner where every new interaction will be added to the automation tests.

### **Modular Testing Framework:**

Abstraction is the concept on which this framework is built. Based on the modules, independent [test scripts are developed to test the software](https://www.testingxperts.com/blog/automated-software-testing). Specifically, an abstraction layer is built for the components to be hidden from the application under test.

This sort of abstraction concept ensures that changes made to the other part of the application does not affect the underlying components.

### **Data Driven Testing Framework:**

In this testing framework, a separate file in a tabular format is used to store both the input and the expected output results. In this framework, a single driver script can execute all the test cases with multiple sets of data.

This driver script contains navigation that spreads through the program which covers both reading of data files and logging of test status information.

### **Keyword Driven Testing Framework:**

[Keyword Driven Testing framework](https://www.testingxperts.com/blog/keyword-driven-testing-why-you-should-use-it) is an application independent framework and uses data tables and keywords to explain the actions to be performed on the application under test. This is more so called as keyword driven test automation framework for web based applications and can be stated as an extension of data driven testing framework.

### **Hybrid Testing Framework:**

This form of hybrid testing framework is the combination of modular, data-driven and keyword test automation frameworks. As this is a hybrid framework, it has been based on the combination of many [types of end-to-end testing](https://www.testingxperts.com/blog/types-of-software-testing) approaches.

### **Test Driven Development framework (TDD):**

Test driven development is a technique of using automated unit tests to drive the design of software and separates it from any dependencies. Earlier, with traditional testing a successful test could find one or more defects, but by using TDD, it increases the speed of tests and improves the confidence that system meets the requirements and is working properly when compared to traditional testing.

### **Behavior Driven Development Framework (BDD):**

This has been derived from the TDD approach and in this method tests are more focussed and are based on the system behavior. In this approach, the testers can create test cases in simple English language. This simple English language helps even the non-technical people to easily analyse and understand the tests.

## Test Plan

<https://www.guru99.com/what-everybody-ought-to-know-about-test-planing.html#:~:text=A%20Test%20Plan%20is%20a,of%20the%20application%20under%20test>.

A **Test Plan** is a detailed document that describes the test strategy, objectives, schedule, estimation, deliverables, and resources required to perform testing for a software product. Test Plan helps us determine the effort needed to validate the quality of the application under test. The test plan serves as a blueprint to conduct software testing activities as a defined process, which is minutely monitored and controlled by the test manager.

## How to write a Test Plan

You already know that making a **Test Plan** is the most important task of Test Management Process. Follow the seven steps below to create a test plan as per IEEE 829

1. Analyze the product
2. Design the Test Strategy
3. Define the Test Objectives
4. Define Test Criteria
5. Resource Planning
6. Plan Test Environment
7. Schedule & Estimation
8. Determine Test Deliverables

1. What is **Element Not Visible Exception**?

🡺When webdriver is able to locate web **element** on current webpage but it **is not visible** on screen.

2. What is **Element is not** clickable **exception?**

**🡺** When webdriver is able to locate web **element** and also **visible** in screen but overlapping by another web **element**.

3. What is **NO Such Element Found Exception?**

**selenium**. NoSuchElementException occurs when **WebDriver** is unable to find and locate elements. Usually, this happens when tester writes incorrect **element** bin the findElement(By, by) method. This **exception** is thrown even if the **element** is **not** loaded.

**Exception Handling**

1.How do you handle an element not visible exception?

🡺Solutions for **Element Not Visible Exception** in Selenium Webdriver. First Solution: Try to write unique XPATH that matches with a single **element** only. Second Solution: Use Explicit wait feature of Selenium and wait till the **element** is **not visible**.

2. How do you fix no such element exception in selenium?

🡺

i.e. We have to use try .. catch blocks to handle the **exception** and also '**NoSuchElementException**' **WebDriver Exception** Class needs to be used in the catch block as shown in the below code: 2. Hover the mouse over the '**NoSuchElementException**' error in the above image and select 'import **NoSuchElementException** org.

## Implicit Wait in Selenium

The **Implicit Wait in Selenium** is used to tell the web driver to wait for a certain amount of time before it throws a "No Such Element Exception". The default setting is 0. Once we set the time, the web driver will wait for the element for that time before throwing an exception.

## Explicit Wait in Selenium

The **Explicit Wait in Selenium** is used to tell the Web Driver to wait for certain conditions (Expected Conditions) or maximum time exceeded before throwing "Element Not Visible Exception" exception. It is an intelligent kind of wait, but it can be applied only for specified elements.

## Difference between Implicit Wait Vs Explicit Wait

|  |  |
| --- | --- |
| **Implicit Wait** | **Explicit Wait** |
| * Implicit Wait time is applied to all the elements in the script | * Explicit Wait time is applied only to those elements which are intended by us |
| * In Implicit Wait, we need **not** specify "Expected Conditions" on the element to be located | * In Explicit Wait, we need to specify "ExpectedConditions" on the element to be located |
| * It is recommended to use when the elements are located with the time frame specified in Selenium implicit wait | * It is recommended to use when the elements are taking long time to load and also for verifying the property of the element like(visibilityOfElementLocated, elementToBeClickable,elementToBeSelected) |

Drop Down Handling

Drop down can be handled in selenium by 3 Ways

* 1. Sendkeys : Fetch the dropdown element and Simply use the sendkeys to send the input
  2. Select Class :

Select obj = new Select(Dropdown webelement);

* + 1. selMonth.selectByIndex(11);

* + 1. selMonth.selectByValue("9");

* + 1. selMonth.selectByVisibleText("May");
    2. List<WebElement> allOps = selMonth.getOptions();

for (WebElement ele : allOps) {

System.out.println(ele.getText());

}

* + 1. System.out.println(selMonth.isMultiple());