Java For Selenium

Java Fundamentals:

- Java for Selenium: How much Java is required?
- Java Variables and Data types
- Operators in Java
- Control flow statements- If and switch case Statements
- Java loop Statements
- Modifiers in Java
- String Class and Built in Methods in Java
- Why String is Immutable in Java?
- Array In Java

Java OOPS Concept:

- ✓ OOP Concept - Class and Objects
- ✓ Methods in Java
- ✓ Inheritance in Java
- ✓ Polymorphism in Java
- ✓ Abstraction and Interface in Java
- ✓ Encapsulation in Java
- ✓ Constructors in Java
- ✓ static Keyword in Java
- ✓ this Keyword in Java
- ✓ super Keyword in Java

Collection Framework in Java:

- Introduction to Collection Framework in Java
- ArrayList in Java
- LinkedList in Java
- Vector and Stack Class in Java
- Queue and Set Interface in Java
- Map Interface in Java

Java Miscellaneous Topics:

- ❖ Exception Handling in Java

Java Fundamentals:

Java for Selenium: How much Java is required?

How much java is required for selenium testing? This question is always in the minds of testing professionals who would like to learn selenium, but are not so comfortable with too much of programming to learn. In this post, we will take a detailed look at the extent of Java to be learnt for Selenium scripting.

Java is one of the languages used for writing automation scripts in Selenium. Selenium supports other languages as well like python, ruby, C#, javascript etc. However, java has gained wide spread acceptance in the industry as the preferred language for selenium.

This means that it is advisable to learn java for selenium as it will also help in improving your career prospects.

For Test Automation using Selenium Core Java is sufficient, Advanced Java is not required.

Java Fundamentals and OOPS (Object Oriented Programming System) concepts are required.

We can segregate Java for selenium in to 2 categories.

A) Java Fundamentals

Data types in Java
Variables
Operators in Java
Flow control Statements- Conditional like if else statements and switch case
Loop Statements – For, while, do while and Enhanced for loop
Strings and Arrays in Java
Java methods - System and user defined
Regular Expressions

B) Java OOPS Concepts and More

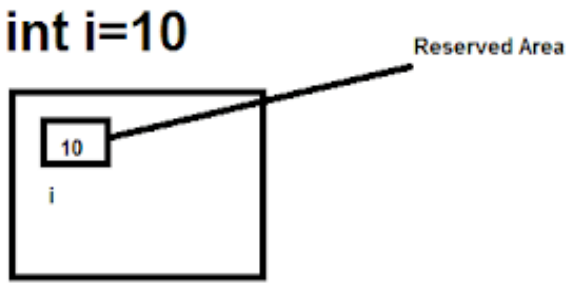
Inheritance
Polymorphism
Abstractions
Encapsulations
Abstract Class method
Interface
Interface vs Abstract Class in Java

Constructor
Packages
File Handling
Exceptions Handling
Static keyword
“this” keyword
Garbage Collection
Collection Framework
Multithreading in Java
How to use Buffered Reader in Java

Java Variables and Data types

What is a variable?
Different definitions given below:

--Variable is a name of memory location
--A memory location to store temporary data within a java program.
--A variable is a container which holds the value while the java program is executed. A variable is assigned with a datatype.
--It is a combination of "vary + able" that means its value can be changed.



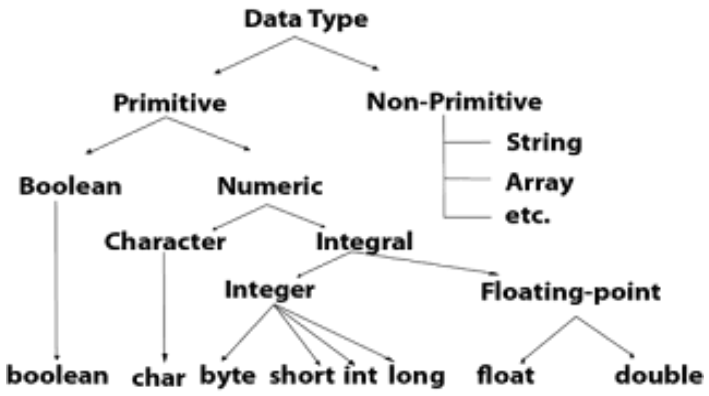
Variable Example

Types of Variables

1. Local variable -- used inside a method
Note: Local Variable contains garbage value
2. Instance/global variable -- used inside a class but outside method
Note: Global variable contain null or default value
3. Static variable -- We will discuss in separate post.

What is a Datatype?

--Data types specify the different sizes and values that can be stored in the variable.
--A data type is a classification of the type of data that a variable or object can hold.
There are two types of data types in Java:
1. Primitive data types: The primitive data types include Boolean, char, byte, short, int, long, float and double.
2. Non-primitive data types: The non-primitive data types include String, Arrays, Classes and Interfaces.



Data Type Hierarchy

Classification of Data Types

Type	Contains	Default	Size	Range
`boolean`	`true` or `false`	`false`	1 bit	NA
`char`	Unicode character	`\u0000`	16 bits	`\u0000` to `\uFFFF`
`byte`	Signed integer	0	8 bits	-128 to 127
`short`	Signed integer	0	16 bits	-32768 to 32767
`int`	Signed integer	0	32 bits	-2147483648 to 2147483647
`long`	Signed integer	0	64 bits	-9223372036854775808 to 9223372036854775807
`float`	IEEE 754 floating point	0.0	32 bits	±1.4E-45 to ±3.4028235E+38
`double`	IEEE 754 floating point	0.0	64 bits	±4.9E-324 to ±1.7976931348623157E+308

Classification of Data Types

Why Java uses Unicode system?

Unicode is a universal international standard character encoding that is capable of representing most of the world's written languages. Before Unicode, there were many language standards: ASCII (American Standard Code for Information Interchange) for the United States.

ISO 8859-1 for Western European Language.
KOI-8 for Russian.
GB18030 and BIG-5 for chinese, and so on.
So to support multinational application codes, some character was using single byte, some two. An even same code may represent a different character in one language and may represent other characters in another language.
To overcome above shortcoming, the unicode system was developed where each character is represented by 2 bytes.

Operators in Java

What is an Operator?

Operator in java is a symbol that is used to perform operations. For example: +, -, *, / etc.
Java provides a rich set of operators to manipulate variables.We will discuss about below operators in this post.

1.Arithmetic Operators
2.Relational Operators
3.Logical Operators
4.Assignment Operators
5.Misc Operators

Arithmetic Operators

Java arithmetic operators are used to perform addition, subtraction, multiplication, and division. They are used to perform basic mathematical operations.

Operator	Description	Example
+ (Addition)	Adds values on either side of the operator.	A + B will give 30
- (Subtraction)	Subtracts right-hand operand from left-hand operand.	A - B will give -10
* (Multiplication)	Multiplies values on either side of the operator.	A * B will give 200
/ (Division)	Divides left-hand operand by right-hand operand.	B / A will give 2
% (Modulus)	Divides left-hand operand by right-hand operand and returns remainder.	B % A will give 0
++ (Increment)	Increases the value of operand by 1.	B++ gives 21
-- (Decrement)	Decreases the value of operand by 1.	B-- gives 19

Arithmetic Operators

Relational Operators

Relational Operators returns Boolean/Logical result.

== (equal to)
!= (not equal to)
> (greater than)
< (less than)
>= (greater than or equal to)
<= (less than or equal to)

Logical Operators:

There are three logical operators given below:

Operator	Description	Example
&& (logical and)	Called Logical AND operator. If both the operands are non-zero, then the condition becomes true.	(A && B) is false
|| (logical or)	Called Logical OR Operator. If any of the two operands are non-zero, then the condition becomes true.	(A || B) is true
! (logical not)	Called Logical NOT Operator. Use to reverses the logical state of its operand. If a condition is true then Logical NOT operator will make false.	!(A && B) is true

Logical Operators

Logical AND		
Operand 1	Operand 2	Result
TRUE	TRUE	TRUE
TRUE	FALSE	FALSE
FALSE	TRUE	FALSE
FALSE	FALSE	FALSE
Logical OR		
Operand 1	Operand 2	Result
TRUE	TRUE	TRUE
TRUE	FALSE	TRUE
FALSE	TRUE	TRUE
FALSE	FALSE	FALSE
Logical NOT		
Operand 1	Operand 2	Result
TRUE	TRUE	FALSE
TRUE	FALSE	TRUE
FALSE	TRUE	TRUE
FALSE	FALSE	TRUE

How Logical Operator Works

Assignment Operators

Below are the Assignment operators are given with explanation:

Operator	Description	Example
=	Simple assignment operator. Assigns values from right side operands to left side operand.	C = A + B will assign value of A + B into C
+=	Add AND assignment operator. It adds right operand to the left operand and assign the result to left operand.	C += A is equivalent to C = C + A
-=	Subtract AND assignment operator. It subtracts right operand from the left operand and assign the result to left operand.	C -= A is equivalent to C = C - A
*=	Multiply AND assignment operator. It multiplies right operand with the left operand and assign the result to left operand.	C *= A is equivalent to C = C * A

Assignment Operators with Examples

Misc Operators

Conditional Operator - Conditional operator is also known as the ternary operator. This operator consists of three operands and is used to evaluate Boolean expressions. The goal of the operator is to decide, which value should be assigned to the variable. The operator is written as −

variable x = (expression) ? value if true : value if false

instanceof Operator - This operator is used only for object reference variables. The operator checks whether the object is of a particular type (class type or interface type). instanceof operator is written as −

(Object reference variable) instanceof (class/interface type)

Control flow statements- If and switch case Statements

Control flow statements

Control flow statements, would change or break the flow of execution by implementing decision making, looping, and branching your program to execute particular blocks of code based on the conditions.
Control flow statements in Java allow you to run or skip blocks of code when special conditions are met. There are 3 types of control flow statements supported by the Java programming language.

Decision-making: if-then, if-then-else, if else if, nested if else and switch
Looping: for, while, do-while
Branching: break, continue, return

In this post we will mainly discuss about **Decision-making statements**.

Java if statements:

1. If-then statement

The “if” statement in Java works exactly like in most programming languages. With the help of “if” you can choose to execute a specific block of code when a predefined condition is met.

Syntax:

```
if (expression) {
  // statements
}
```

Here expression is a boolean expression (returns either true or false).

If the expression is evaluated to true, statement(s) inside the body of if (statements inside parenthesis) are executed.

If the expression is evaluated to false, statement(s) inside the body of if are skipped from execution.

Example:

```
public class IfThenDemo {
 public static void main(String[] args) {
  int age = 2;
  System.out.println("James age is " + age + " years old");
  if (age < 3) {
   System.out.println("James is an Infant");
  }
 }
}
```

Output:

James age is 2 years old
James is an Infant

2. The if-then-else Statement:

The if-then-else statement provides a alternate path of execution when an if clause evaluates to false.

Syntax:

The syntax of if-then-else statement is:

```
if (expression) {
   // codes
}
else {
  // some other code
}
```

Example:

```
public class IfThenElseDemo {
 public static void main(String[] args) {
  int A= 4;
              int B= 7;
  if (A > B) {
   System.out.println("A is Greater than B");
  }
              else{
                    System.out.println("B is Greater Number");
                  }

 }
}
```

Output:

B is Greater Number

3. If else If Statement

In Java, it's possible to execute one block of code among many. For that, you can use if..else...if ladder.

Syntax:

```
if (expression1)
{
   // codes
}
else if(expression2)
{
   // codes
}
else if (expression3)
{
   // codes
```

```
}
.
.
else
{
   // codes
}
```

Example:

```
public class FlowControlExample {
 public static void main(String[] args) {
  int age = 17;
  System.out.println("James is " + age + " years old");
  if (age < 4) {
   System.out.println("James is a baby");
  } else if (age >= 4 && age < 13) {
   System.out.println("James is a child");
  } else if (age >= 13 && age < 20) {
   System.out.println("James is a teenager");
  } else if (age >= 20 && age < 60) {
   System.out.println("James is an adult");
  } else {
   System.out.println("Jamesr is an old men");
  }
 }
}
```

Output:
James is 17 years old
James is a teenager

4. Nested if..else Statement

It's possible to have if..else statements inside a if..else statement in Java. It's called nested if...else statement.

5. Switch Statement:

In Java, the if..else..if ladder executes a block of code among many blocks. The switch statement can a substitute for long if..else..if ladders which generally makes your code more readable.

Syntax:

```
switch (variable/expression) {
case value1:
  // statements
  break;
case value2:
  // statements
  break;
  .. .. ...
  .. .. ...
default:
  // statements
}
class SwitchDemo {
   public static void main(String[] args) {
      int age = 2;
      String yourAge;
      switch (age) {
         case 1:  System.out.println("You are one year old");
                  break;
         case 2:  System.out.println("You are two year old");
                  break;
         case 3:  System.out.println("You are three year old");
                  break;
         default: System.out.println("You are more than three year old");
                  break;
      }
   }
}
```

Output:
You are two year old

Java loop Statements

Java Loop Statements

In programming languages, loops are used to execute a set of instructions/functions repeatedly when some conditions become true.

for loop
while loop
do while loop
enhanced for loop

Loop Examples:

Fetching records (test data) from external source – like DB/ Excel file
Execute one block of code multiple times

1. for loop:

If the number of iteration is fixed, it is recommended to use for loop.

Syntax:

```
for(init;condition;incr/decr){
// code to be executed
}
```

Example: Lets print the numbers from 1 to 5.

```
for(int i=1; i<=5; i++){
   System.out.println("Printing using for loop. Count is: " + i);
 }
```

```
Printing using for loop. Count is: 1
Printing using for loop. Count is: 2
Printing using for loop. Count is: 3
Printing using for loop. Count is: 4
Printing using for loop. Count is: 5
```

2. while loop:

If the number of iteration is not fixed, it is recommended to use while loop.

Syntax:

```
while(condition){
//code to be executed
}
```

Example: Lets print the numbers from 1 to 5.

```
int i=1;
while(i<6){
  System.out.println("Printing using while loop. Count is: " + i);
}
```

```
Printing using while loop. Count is: 1
Printing using while loop. Count is: 2
Printing using while loop. Count is: 3
Printing using while loop. Count is: 4
Printing using while loop. Count is: 5
```

3. do while:

If the number of iteration is not fixed and you must have to execute the loop at least once, it is recommended to use the do-while loop.

Syntax:

```
do{
//code to be executed
}while(condition);
```

4. Enhanced/for-each loop

In Java, there is another form of for loop (in addition to standard for loop) to work with arrays and collection, the enhanced for loop.

Syntax:

```
for(Type var : arrayname){
//code to be executed
}
```

Modifiers in Java

Modifiers are key words in java that is used to restrict or add access level for classes, attributes, methods and constructors.

Access Modifiers - default, Private, Protected, Public
Non Access modifiers –
For Classes – final, abstract
For Methods or attributes – final, static, abstract, transient, synchronized, volatile

Visibility	Default	Private	Protected	Public
From the same Class	Yes	Yes	Yes	Yes
From a sub-class in the same package	Yes	No	Yes	Yes
From any class in the same package	Yes	No	Yes	Yes
From a sub-class outside package	No	No	Yes	Yes
From any class in outside package	No	No	No	Yes

Access Modifiers Visibility

1. Default:

When no access modifier is specified for a class , method or data member – It is said to be having the default access modifier by default.

The data members, class or methods which are not declared using any access modifiers i.e. having default access modifier are accessible only within the same package.

Scope – within same package

2. Private

The private access modifier is specified using the keyword private.
The methods or data members declared as private are accessible only within the class in which they are declared.
Any other class of same package will not be able to access these members.

Scope – within same class

3. Protected

The protected access modifier is specified using the keyword protected.

The methods or data members declared as protected are accessible within same package or sub classes in different package.

Scope – within same package and other package using inheritance

4. Public

The members, methods and classes that are declared public can be accessed from anywhere. This modifier doesn't put any restriction on the access.

Scope – accessible from anywhere inside or outside of the package.

String Class and Built in Methods in Java

String:

In Java, string represents sequence of char values. An array of characters works same as Java string. For example:

```
char[] h={'H','E','L','L','O'};
String s=new String(h);
```

String s="HELLO";

Java String class provides a lot of methods to perform operations on string

How to create string object?

The java.lang.String class is used to create a string object.
There are two ways to create String object:

By string literal - Java String literal is created by using double quotes. For Example:
String s1="Selenium";

By new keyword - String s=new String("Selenium");

Built in Methods:

Java has a librabry of classes and methods organised in packages
Import java.io.console
Import java.io.*

In order to use built in methods we need to import packages or classes
Java.lang package is automatically imported in every java Built in methods categories
String methods
Array methods
Number methods
Character methods

Different String methods:

compareTo - The Java String compareTo() method is used for comparing two strings lexicographically.

boolean equals() - The java string equals() method compares the two given strings based on the content of the string (case sensitive)
String concat() – concat two strings
boolean equalsIgnoreCase() - The java string equals() method compares the two given strings based on the content of the string (not casesensitive)
char charAt() – index position - The java string charAt() method returns a char value at the given index number.
boolean contains()
toUpperCase() – convert to upper case
toLowerCase() – convert to lower case
trim() – remove spaces from both sides of string

substring() -- returns part of string
boolean endsWith()
boolean startWith() – ends with specified suffix or not
int length()
replace()

Java Convert String to int:

Convert String to int using Integer.parseInt(String)

```
String str = “54321";
int num = Integer.parseInt(str);
```

Convert String to int using Integer.valueOf(String)

```
String str=“123";
int num = Integer.valueOf(str);
```

Java int to String Conversion:

Convert int to String using String.valueOf()

```
 String int ivar = 123;
String str = String.valueOf(ivar);
System.out.println("String is: "+str);
System.out.println(555+str);
```

Convert int to String using Integer.toString()

```
int ivar = 123;
String str = Integer.toString(ivar);
System.out.println("String is: "+str);
System.out.println(555+str);
```

Why String is Immutable in Java?

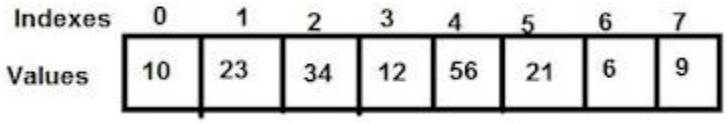
In java, string objects are immutable. Immutable simply means unmodified or unchangeable. Once string object is created its data or state can't be changed but a new string object is created.

Array In Java

What is an Array?

1. An array is a collection of similar type of elements that have a contiguous memory location.

2. The variables in the array are ordered and each have an index.



An Array Example

Array Types:

1. Single Dimensional Array
2. Multidimensional Array

Advantages -
code optimization by storing data and we can retrieve or sort the data efficiently.
We can get any data located at an index position.

Disadvantages-
Stores similar type of data
We can store only the fixed size of elements in the array. It doesn't grow its size at runtime.

1-D Array:

Declaration of an Array:
Syntax: data_type array-name[]; Example: int a[];
or
Data_type[] array-name; Example: int[] a;

Instantiation of an Array
array-var-name =new datatype[size]; Example: a= new int[5]

Declaration and Instantiation
Data_type array-name[]= new data_type[5];
int a[]=new int[5];

Array Literal
Data_type array-name[] = {<Values>}
Example: int a[] = {12,23,1,34,5,6}

2-D Array:

Declaration of an Array:
Syntax: data_type array-name[][]; Example: int a[][];
or
Data_type[][] array-name; Example: int[][] a;

Instantiation of an Array
array-var-name =new datatype[][]; Example: a= new int[2][2]

Declaration and Instantiation
Data_type array-name[][]= new data_type[2][2];
int a[][]=new int[2][2];

Array Literal
Data_type array-name[] = {<Values>}
Example: int a[][] = {{12,23},{23,34}}

Java OOPS Concept:

OOP Concept - Class and Objects

What is OOP?

Object oriented programming – As the name suggests uses objects in programming. Object oriented programming aims to implement real world entities like inheritance, hiding, polymorphism etc in programming. The main aim of OOP is to bind together the data and the functions that operates on them so that no other part of code can access this data except that function.

Oops Concept

Java is an Object-Oriented Language. As a language that has the Object-Oriented feature, Java supports the following fundamental concepts −

Polymorphism
Inheritance
Encapsulation
Abstraction
Classes
Objects
Methods

1. What is an Object:

In object oriented programming Whenever you do something you need an object.
Print, calculate, execute, process, save and to return
Object don't do things individually
Basically object has couple of information
What object knows ◊ variables -- to store values
What object does ◊ methods (execute and process)
Object are crated using class. Objects and classes are interrelated to each other.

If we consider the real-world, we can find many objects around us, cars, dogs, humans, etc. All these objects have a state and a behaviour.

If we consider a dog, then its state is - name, breed, color, and the behaviour is - barking, wagging the tail, running.

2. What is a Class?

A class is a blueprint or design from which individual objects are created.

```
public class Dog {
  String breed;
  int age;
  String color;

  void barking() {
  }

  void hungry() {
  }

  void sleeping() {
  }
}
```

How to Create an Object?

In Java, an object is created from a class.

Syntax - <ClassName> Reference = new <ClassName()>;

When an object of a class is created, the class is said to be instantiated. All the instances share the attributes and the behaviour of the class. But the values of those attributes, i.e. the state are unique for each object. A single class may have any number of instances.

Methods in Java

What is a method?

A java method is a set of statements or steps that are grouped together to perform an operation
Method are also known as functions
When we use methods? – whenever we want to perform an operation multiple times
Advantage of methods – code reusability
Types of methods - built in (predefined) and user defined

1. Built in Methods:

Java has a librabry of classes and methods organised in packages
Import java.io.Console
Import java.io.*

In order to use built in methods we need to import packages or classes

Java.lang package is automatically imported in every java Built in methods categories
String methods
Array methods
Number methods
Character methods

2. User Defined Methods:

Different Method types:
Method without returning any values
Method with returning values
Method using passing parameters

Different ways to call Methods:
Using Objects
Without Using object
Call external methods (from external class)

Inheritance in Java

What is Inheritance?

Defnition 1: Inheritance in Java is a mechanism in which one object acquires all the properties and behaviours of a parent object
Defnition 2: The process by which one class acquires the properties(data members) and functionalities(methods) of another class is called inheritance.
Defnition 3: Inheritance can be defined as the process where one class acquires the properties (methods and fields) of another.

Why we use Inheritance?

--For Code Re-usability- The biggest advantage of Inheritance is that the code that is already present in base class need not be rewritten in the child class.
--Avoid duplication in code.
--For method overriding

Different terms associated with Inheritance:

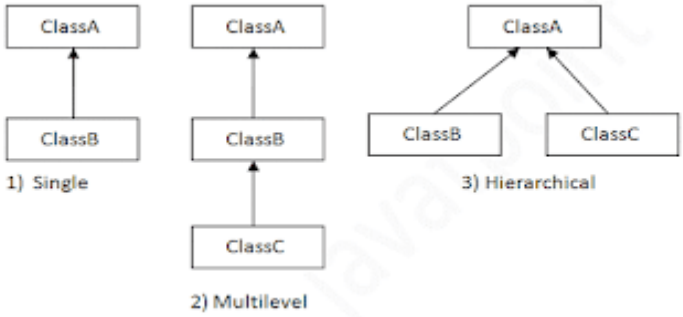
Class: A class is a group of objects which have common properties. It is a template or blueprint from which objects are created.
Sub Class/Child Class: Subclass is a class which inherits the other class. It is also called a derived class, extended class, or child class.
Super Class/Parent Class: Superclass is the class from where a subclass inherits the features. It is also called a base class or a parent class.
Reusability: As the name specifies, reusability is a mechanism which facilitates you to reuse the fields and methods of the existing class when you create a new class. You can use the same fields and methods already defined in the previous class.

There are two ways we can do code reuse either by the implementation of inheritance (IS-A relationship), or object composition (HAS-A relationship)

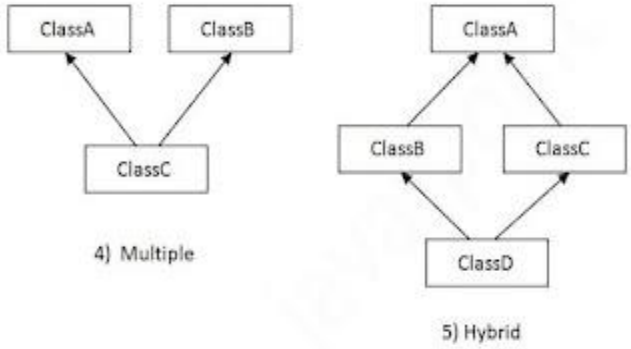
Syntax:

```
class Subclass-name extends Superclass-name
    {
        //data members and methods
     }
```

Types of Inheritance



Types of Inheritance in Java



Multiple Inheritance and Hybrid Inheritance in Java

Various Scenarios to create Parent/Child objects:

1. Sub(Child) Class reference and Sub(Child) Object- Allow you to access all the methods and data members of super class and sub class.
2. Super(Parent) Class reference and Sub(Child) Class Object- Allow you to access all the methods and data members of Super class only.
3. Super(Parent) Class reference and Super(Parent) Class Object - Allow you to access all methods and data members of super class only.
4. Sub(Child) Class reference and Super(Parent) Class object- child cannot hold parent object

Usage of inheritance in selenium:

1. We create a Base Class in the Framework to initialize WebDriver interface, WebDriver waits, Property files, Excels, etc., in the Base Class.

2. We extend the Base Class in other classes such as Tests and Utility Class.

Polymorphism in Java

What is Polymorphism?

--Polymorphism in Java is a concept by which we can perform a single action in different ways.
--Polymorphism is derived from 2 Greek words: poly and morphs. The word "poly" means many and "morphs" means forms. So polymorphism means many forms.
--There are two types of polymorphism in Java: compile-time polymorphism and runtime polymorphism.
--Polymorphism can be achieved by method overloading and method overriding

Method Overloading:

--Method Overloading is a feature that allows a class to have more than one method having the same name, if their argument lists are different.
--Signature should be different:

1. Number of parameters.
2. Data type of parameters.
3. Sequence of Data type of parameters.

Method Overriding:

--Declaring a method in sub class which is already present in parent class is known as method overriding.

--Advantage: The main advantage of method overriding is that the class can give its own specific implementation to a inherited method without even modifying the parent class code.

Types of Polymorphism:

1. Static Polymorphism (Static Binding - Early binding) is also known as compile time binding or compile time polymorphism-- Method Overloading in an example -- when type of object is determined at compiled time it is known as Static binding.
2. Dynamic Polymorphism (Dynamic Binding -late binding) is also known as runtime time binding or run time polymorphism -- example method overriding-- When type of object is determined at run time, it is known as dynamic binding.

Abstraction and Interface in Java

Abstraction:

--Abstraction is a process of hiding the implementation details and showing only functionality to the user.
--In Abstraction only the essential details are displayed to the user. The trivial or the non-essentials units are not displayed to the user. Ex: A car is viewed as a car rather than its individual.
--There are two ways to achieve abstraction in java
1. Abstract class (0 to 100%)
2. Interface (100%)

Abstract Class:

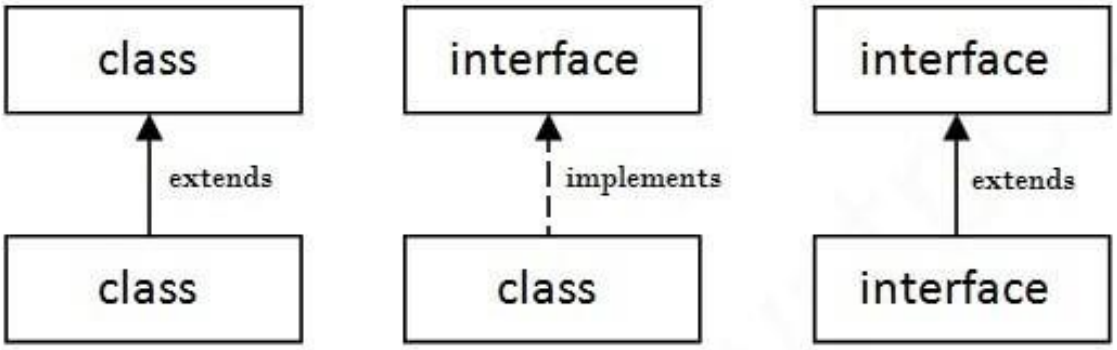
A class which is declared as abstract is known as an abstract class. It can have abstract and non-abstract methods. It needs to be extended and its method implemented.
An abstract class must be declared with an abstract keyword.
Abstract Class can have abstract and non-abstract methods.
Abstract class cannot be instantiated.
Abstract class can have constructors final and static methods also.
Abstract class can have final methods
If there is an abstract method in a class, that class must be abstract.

Interface:

--An interface in java is a blueprint of a class. It has static constants and abstract methods.

--There can be only abstract methods in the Java interface, not method body. It is used to achieve abstraction and multiple inheritance in Java.
--It cannot be instantiated just like the abstract class.
--In interface we can have default and static methods.
--In interface we can have private methods.

Class and Interface Relationships:



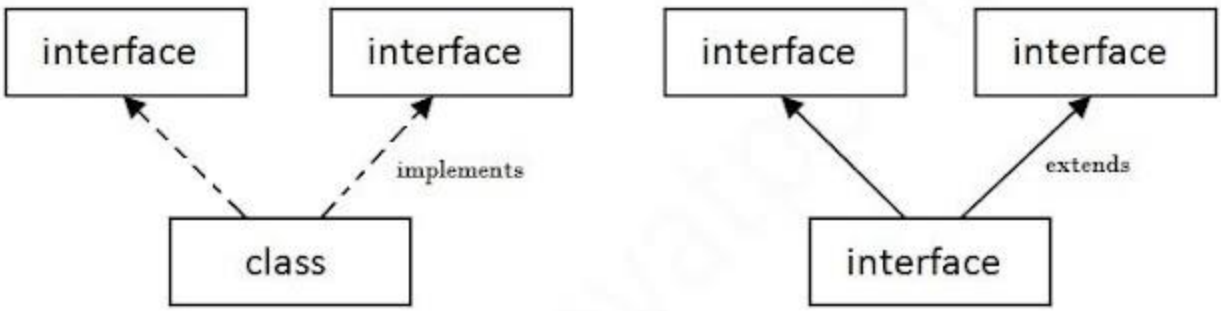
Class and Interface Relationships

Why we use Interface?

--It is used to achieve abstraction.
--By interface, we can support the functionality of multiple inheritance.

Multiple inheritance in Java by interface:

--If a class implements multiple interfaces, or an interface extends multiple interfaces, it is known as multiple inheritance.
--Multiple inheritance is not supported through class in java, but it is possible by an interface,



Multiple Inheritance in Java

Abstract Class Vs Interface

Abstract Class	Interface
An abstract class can have both abstract and non-abstract methods.	The interface can have only abstract methods.
It does not support multiple inheritances.	It supports multiple inheritances.
It can provide the implementation of the interface.	It can not provide the implementation of the abstract class.
An abstract class can have protected and abstract public methods.	An interface can have only have public abstract methods.
An abstract class can have final, static, or static final variable with any access specifier.	The interface can only have a public static final variable.

Difference between Abstract Class and Interface

Encapsulation in Java

1. Encapsulation in Java is a mechanism of wrapping the data (variables) and methods together as a single unit.
2. In encapsulation, the variables of a class will be hidden from other classes, and can be accessed only through the methods of their current class. Therefore, it is also known as data hiding.

How to Achieve?

1. Declare the variables of a class as private.
2. Provide public setter and getter methods to modify and view the variables values.

Benefits:

1. The fields of a class can be made read-only (class which has only getter method) or write-only (class which has only setter method) .
2. A class can have total control over what is stored in its fields.

Constructors in Java

Constructor Definition:

1. It is called constructor because it constructs the value at the time of object creation.
2. Block of code similar to method.
3. It is called when an object of class is created.

4. At the time of calling constructor, memory for the object is allocated in the memory.
5. It is used to initialize the object.
6. Java compiler created the default constructor if your class doesn't have any constructor.

Rules to Create Constructor:

1. Constructor name must be the same as its class name
2. Must have no return type
3. A Java constructor cannot be abstract, static, final, and synchronized

Types of Constructor:

1. Default or no arguments – Provides default values
2. Parameterized constructor

Constrtuctore Overloading:

having more than one constructor with different parameter lists.

Constructor Vs Method:

Java Constructor	Java Method
It is used to initialize the state of an object.	It is used to expose the behavio of an object.
A constructor must not have a return type.	A method must have a return type.
Invoked implicitly.	Invoked explicitly.
The Java compiler provides a default constructor if you don't have any constructor in a class.	The method is not provided by the compiler in any case.
The constructor name must be same as its class name.	Not same as its Class name

Java Constructor Vs Java method

static Keyword in Java

The static keyword in Java is used for memory management
The static keyword belongs to the class than an instance of the class.
Static can be : Variable, Method, Block , Nested class

static Variable:
1. Static variable is used to fulfil the common requirement. For Example company name of employees, college name of students etc. Name of the college is common for all students
2. static variable gets memory only once in the class area at the time of class loading.

static Method:
1. A static method belongs to the class rather than the object of a class.

2. A static method can be invoked without the need for creating an instance of a class.
3. A static method can access static data member and can change the value of it.
Restrictions -
--The static method can not use non static data member or call non-static method directly.
--this and super cannot be used in static context.

Why main method is static?

It is because the object is not required to call a static method. If it were a non-static method, JVM creates an object first then call main() method that will lead the problem of extra memory allocation.

Difference between static and final keyword:

static keyword always fixed the memory that means that will be located only once in the program where as final keyword always fixed the value that means it makes variable values constant

Java static Block:
Is used to initialize the static data member.
It is executed before the main method at the time of classloading.

this Keyword in Java

what is this keyword in java?

this is a reference variable that refers to the current object.

this Keyword usage:
1. To refer current class instance variable
2. To invoke current class method
3. To invoke current class constructor
4. this can be passed as an argument in the method call
5. this can be used to return the current class instance from the method
6. this can be passed as an argument in the constructor call

super Keyword in Java

Three Important Usage of super keyword:

Use of super with variables: We can use super keyword to access the data member or field of parent class. It is used if parent class and child class have same fields.

Use of super with methods: It should be used if subclass contains the same method as parent class.

Use of super with constructors: super is used to invoke parent class constructor.

Collection Framework in Java

Introduction to Collection Framework in Java

1. A Collection is a group of individual objects represented as a single unit
2. Java Collections can achieve all the operations that you perform on a data such as searching, sorting, insertion, manipulation, and deletion.
3. The Java Collections Framework is a collection of interfaces and classes which helps in storing and processing the data efficiently.

Interfaces:
List
Set
Map
Queue
Deque
SortedSet

Classes:
ArrayList, LinkedList, Vector, Stack, PriorityQueue, HashSet, LinkedHashSet, TreeSet, HashMap, LinkedHashMap, TreeMap and HashTable.

Collection Interface:
1. Root interface with basic methods like add(), remove(), contains(), isEmpty(), addAll(), clear().. etc.

2. All other collection interfaces and classes (except Map) either extend or implement this interface. For example, List (indexed, ordered) and Set (sorted) interfaces implement this collection.

ArrayList in Java

List Interface:

1. Can contain duplicates and elements are ordered.
2. List interface is implemented by the classes ArrayList, LinkedList, Vector, and Stack.
3. Lists represents an ordered collection of elements. Using lists, we can access elements by their integer index (position in the list), and search for elements in the list. index start with 0, just like an array.

ArrayList Class in Java:

Properties:

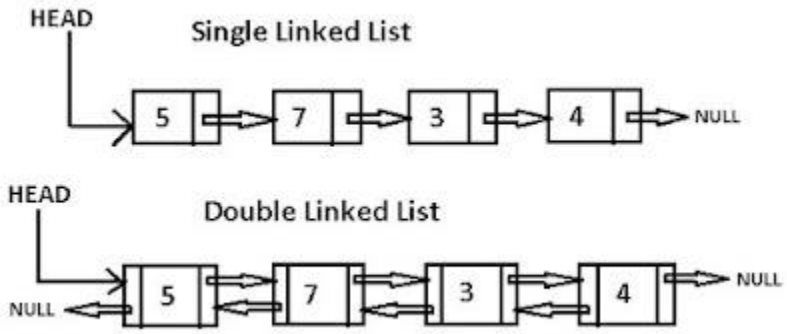
Ordered – Elements in arraylist preserve their ordering which is by default the order in which they were added to the list.

Index based – Elements can be randomly accessed using index positions. Index start with '0'.
Dynamic resizing:
Non synchronized:
Duplicates allowed – We can add duplicate elements in arraylist. It is not possible in sets.
How to create:
ArrayList<String> alist=new ArrayList<String>();

Methods: add(), set(int index, Object o), remove(index or value), get(int index), indexOf(Object o), int size(), boolean contains(Object o), clear() etc.

LinkedList in Java

Java LinkedList class is doubly-linked list implementation of the List and Deque interfaces.



Single and Doubly Linked List

Properties:

--Permits all elements including duplicates and NULL.
--LinkedList maintains the insertion order of the elements.
--It is not synchronized.
--the manipulation is fast because no shifting is required.

How to create:

LinkedList<String> linkedList = new LinkedList<>();

Methods: boolean add(Object item), void add(int index, Object item), boolean addAll(Collection c), void addFirst(Object item), void addLast(Object item), void clear(), Object clone(), Object getFirst(), Object getLast(), Object poll() etc.

Vector and Stack Class in Java

Properties:
Vector uses a dynamic array to store the data elements. It is similar to ArrayList. However, It is synchronized. it is synchronized and due to which it gives poor performance in searching, adding, delete and update of its elements.

How to Create?
Vector<String> v=new Vector<String>();

Methods: addElement(Object element), int capacity(), int size(), firstElement(), lastElement(), get(int index) etc.

Queue and Set Interface in Java

Queue Interface:

Java Queue interface orders the element in FIFO(First In First Out) manner. In FIFO, first element is removed first and last element is removed at last.

How to create the objects?

Queue<String> q1 = new PriorityQueue();
Queue<String> q2 = new ArrayDeque();

PriorityQueue Class

Properties: It holds the elements or objects which are to be processed by their priorities. PriorityQueue doesn't allow null values to be stored in the queue.

Methods: boolean add(object), boolean offer(object), boolean remove(object), Object poll(), Object element() , Object peek(), void clear(), int size()

Deque Interface:

Deque interface extends the Queue interface. In Deque, we can remove and add the elements from both the side. Deque stands for a double-ended queue which enables us to perform the operations at both the

Deque d = new ArrayDeque();

ArrayDeque Class

ArrayDeque class implements the Deque interface. It facilitates us to use the Deque. Unlike queue, we can add or delete the elements from both the ends.

ArrayDeque is faster than ArrayList and Stack and has no capacity restrictions.

Set Interface

Properties: 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.

Set<data-type> s1 = new HashSet<data-type>();
Set<data-type> s2 = new LinkedHashSet<data-type>();
Set<data-type> s3 = new TreeSet<data-type>();

HashSet Class

Properties: This class implements the Set interface.
--HashSet doesn’t maintain any order,
--HashSet doesn’t allow duplicates
--HashSet allows null values however if you insert more than one nulls it would still return only one null value.
--HashSet is non-synchronized.
--Hashing is used to store the elements in the HashSet

HashSet<String> hset = new HashSet<String>();

Methods: boolean add(E e) , void clear(), Object clone(), boolean contains(Object o), boolean isEmpty(), int size().

LinkedHashSet Class

Properties: This class implements the Set interface.
--LinkedHashSet maintains insertion order,
--LinkedHashSet doesn’t allow duplicates

--LinkedHashSet allows null values however if you insert more than one nulls it would still return only one null value.
--LinkedHashSet is non-synchronized.
LinkedHashSet<String> set=new LinkedHashSet<String>();

Methods: boolean add(E e) , void clear(), Object clone(), boolean contains(Object o), boolean isEmpty(), int size(), boolean remove(Object o), removeAll().

TreeSet Class

Properties: This class implements the SortedSet interface.
--The access and retrieval time of TreeSet is quite fast. The elements in TreeSet stored in ascending order.
--TreeSet doesn't allow duplicates
--TreeSet doesn't allows null values
--TreeSet is non-synchronized.

TreeSet<String> tset = new TreeSet<String>();

Methods: boolean add(E e) , void clear(), Object clone(), boolean contains(Object o), boolean isEmpty(), int size(), boolean remove(Object o), Object first(), Object last().

Map Interface in Java

Properties: A map contains values on the basis of key, i.e. key and value pair. Each key and value pair is known as an entry. A Map contains unique keys.

A Map doesn't allow duplicate keys, but you can have duplicate values. HashMap and LinkedHashMap allow null keys and values, but TreeMap doesn't allow any null key or value.

A Map can't be traversed, so you need to convert it into Set using keySet() or entrySet() method.

HashMap Class

Properties:
Java HashMap class contains values based on the key.
Java HashMap class contains only unique keys.
Java HashMap class may have one null key and multiple null values.
Java HashMap class is non synchronized.

Java HashMap class maintains no order.

Methods: void clear(), Object clone(), boolean containsKey(Object key), boolean containsValue(Object Value) , Object get(Object key), boolean isEmpty(), Set keySet(), Object put(Key k, Value v), int size(), Collection values(), Value remove(Object key)

LinkedHashMap Class

Properties:
Java LinkedHashMap contains values based on the key.
Java LinkedHashMap contains unique elements.
Java LinkedHashMap may have one null key and multiple null values.
Java LinkedHashMap is non synchronized.
Java LinkedHashMap maintains insertion order.

Methods: void clear(), void size(), void isEmpty(), boolean containsKey(Object key), boolean containsValue(Object key), Object get(Object key)

TreeMap Class

Properties:
Java TreeMap contains values based on the key. It implements the NavigableMap interface and extends AbstractMap class.
Java TreeMap contains only unique elements.
Java TreeMap cannot have a null key but can have multiple null values.
Java TreeMap is non synchronized.
Java TreeMap maintains ascending order.

Methods: void clear(), void size(), void isEmpty(), boolean containsKey(Object key), boolean containsValue(Object key), Object get(Object key), Object firstKey(), Object lastKey()

HashTable Class:

Properties: Hashtable internally contains buckets in which it stores the key/value pairs. The Hashtable uses the key's hashcode to determine to which bucket the key/value pair should map.

A Hashtable is an array of a list. Each list is known as a bucket. The position of the bucket is

identified by calling the hashcode() method. A Hashtable contains values based on the key.
Java Hashtable class contains unique elements.
Java Hashtable class doesn't allow null key or value.
Java Hashtable class is synchronized.

Methods: void clear(), void size(), void isEmpty(), boolean containsKey(Object key), boolean containsValue(Object key), Object get(Object key)

Java Miscellaneous Topics:

Exception Handling in Java

What is an Exception?

--An exception is a problem that arises during the execution of a program
--Its an event that disrupts the normal flow of the program

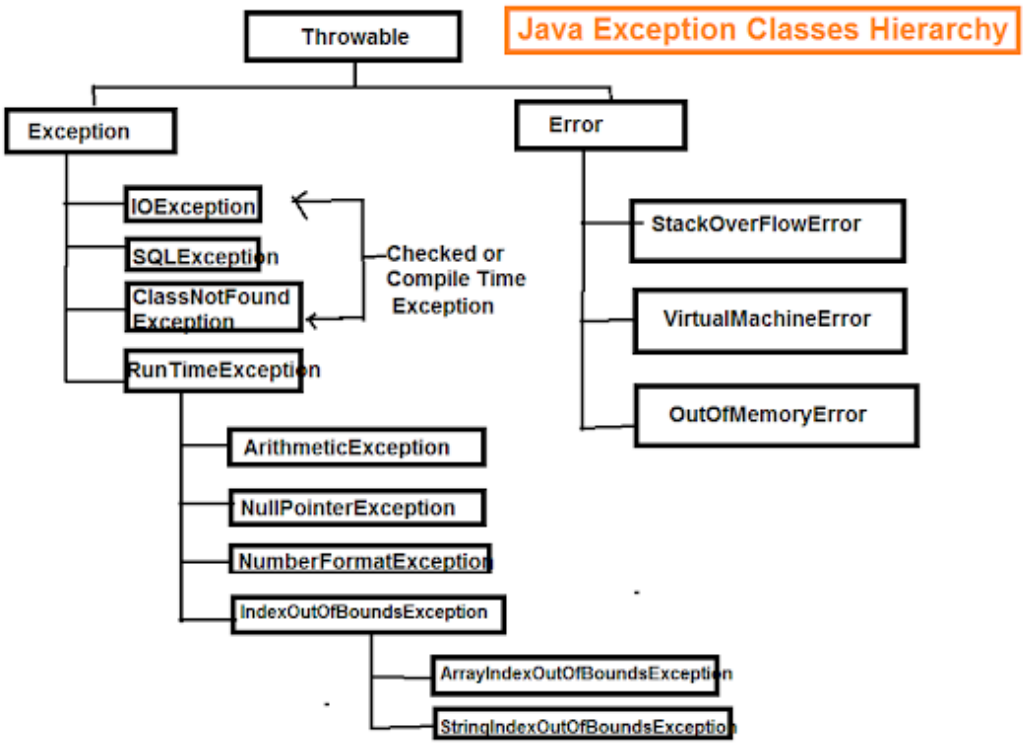
How exception occurs?

Following are some scenarios where an exception occurs.

1. A user has entered an invalid data.

2. A file that needs to be opened cannot be found.

3. A network connection has been lost in the middle of communications or the JVM has run out of memory.



Exception Class Hierarchy

Types of java exceptions:

1. **Checked Exception** - Checked exceptions are checked at compile-time so this is also called compile time exceptions. Example of checked exceptions are : ClassNotFoundException, IOException, SQLException and so on.They occur usually interacting with outside resources/network resources e.g. database problems, network connection errors, missing files etc.

2. **UnChecked Exception** - Unchecked exceptions are not checked at compile-time, but they are checked at runtime. Also Called Run time Exceptions. Example of unchecked exceptions are : ArithmeticException, ArrayStoreException, ClassCastException and so on.

3. **Error** - Error is irrecoverable e.g. OutOfMemoryError, VirtualMachineError, AssertionError etc.

How to handle exception?

Java exception handling is managed via five keywords:

try: this keyword is used to specify a block where we should place exception code. The try block must be followed by either catch or finally. It means, we can't use try block alone.

catch: this block is used to handle the exception. It must be preceded by try block which means we can't use catch block alone. It can be followed by finally block later.

finally: this block is used to execute the important code of the program. It is executed whether an exception is handled or not.

throw: this keyword is used to throw an exception.

throws: this keyword is used to declare exceptions. It doesn't throw an exception. It specifies that there may occur an exception in the method. It is always used with method signature.

try catch syntax:

```
try{
//code that may throw an exception
}catch(ExceptionClassName ref){
}
```

try finally block

```
try{
//code that may throw an exception
}finally{
}
```