Important Java Topics for Selenium

**1)Constructor in Java**

In Java, constructor is a block of codes similar to method. It is called when an instance of object is created and memory is allocated for the object.

It is a special type of method which is used to initialize the object.

**Note:** It is called constructor because it constructs the values at the time of object creation. It is not necessary to write a constructor for a class. It is because java compiler creates a default constructor if your class doesn't have any.

Rules for creating java constructor

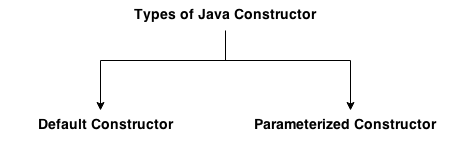
There are basically two rules defined for the constructor.

1. Constructor name must be same as its class name
2. Constructor must have no explicit return type

Types of java constructors

There are two types of constructors in java:

1. Default constructor (no-arg constructor)
2. Parameterized constructor



## Default Constructor

A constructor is called "Default Constructor" when it doesn't have any parameter.

### Syntax of default constructor:

<class\_name>(){}

**Ex for default constructor**

**class** Bike1{

Bike1(){

System.out.println("Bike is created");

}

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

Bike1 b=**new** Bike1();

}  }

#### Rule: If there is no constructor in a class, compiler automatically creates a default constructor.

### What is the purpose of default constructor?

Default constructor is used to provide the default values to the object like 0, null etc. depending on the type.

### Example of default constructor that displays the default values

**class** Student3{

**int** id;

String name;

**void** display(){System.out.println(id+" "+name);

}

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

Student3 s1=**new** Student3();

Student3 s2=**new** Student3();

s1.display();

s2.display();

}

}

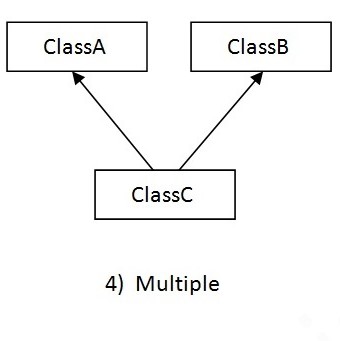
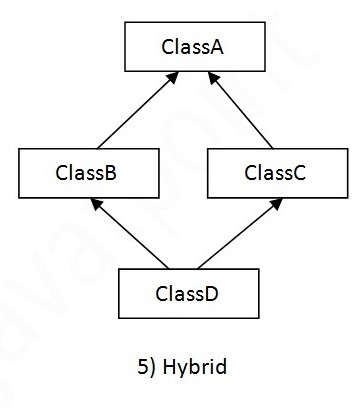
Parameterized constructor

A constructor which has a specific number of parameters is called parameterized constructor.

Why use parameterized constructor?

|  |
| --- |
| Parameterized constructor is used to provide different values to the distinct objects. Example of parameterized constructor  1. **class** Student4{ 2. **int** id; 3. String name; 5. Student4(**int** i,String n){ 6. id = i; 7. name = n; 8. } 9. **void** display(){System.out.println(id+" "+name);} 11. **public** **static** **void** main(String args[]){ 12. Student4 s1 = **new** Student4(111,"Karan"); 13. Student4 s2 = **new** Student4(222,"Aryan"); 14. } 15. s1.display(); 16. s2.display(); 17. }  2)Constructor Overloading in Java In Java, a constructor is just like a method but without return type. It can also be overloaded like Java methods.  Constructor overloading in Java is a technique of having more than one constructor with different parameter lists. They are arranged in a way that each constructor performs a different task. They are differentiated by the compiler by the number of parameters in the list and their types. Example of Constructor Overloading  1. **class** Student5{ 2. **int** id; 3. String name; 4. **int** age; 5. Student5(**int** i,String n){ 6. id = i; 7. name = n; 8. } 9. Student5(**int** i,String n,**int** a){ 10. id = i; 11. name = n; 12. age=a; 13. } 14. **void** display(){ 15. System.out.println(id+" "+name+" "+age); 16. } 17. **public** **static** **void** main(String args[]){ 18. Student5 s1 = **new** Student5(111,"Karan"); 19. Student5 s2 = **new** Student5(222,"Aryan",25); 20. s1.display(); 21. s2.display(); 22. } 23. }  Java Copy Constructor There is no copy constructor in java. But, we can copy the values of one object to another like copy constructor in C++.  There are many ways to copy the values of one object into another in java. They are:   * By constructor * By assigning the values of one object into another * By clone() method of Object class  Copying values without constructor We can copy the values of one object into another by assigning the objects values to another object. In this case, there is no need to create the constructor.  Does constructor return any value?  **Ans:** Yes, that is current class instance (You cannot use return type yet it returns a value).  Can constructor perform other tasks instead of initialization?  Yes, like object creation, starting a thread, calling method etc. You can perform any operation in the constructor as you perform in the method. 3)Method Overloading in Java If a class has multiple methods having same name but different in parameters, it is known as **Method Overloading**.  If we have to perform only one operation, having same name of the methods increases the readability of the program.  Suppose you have to perform addition of the given numbers but there can be any number of arguments, if you write the method such as a(int,int) for two parameters, and b(int,int,int) for three parameters then it may be difficult for you as well as other programmers to understand the behavior of the method because its name differs.  So, we perform method overloading to figure out the program quickly. Advantage of method overloading Method overloading increases the readability of the program. Different ways to overload the method There are two ways to overload the method in java   1. By changing number of arguments 2. By changing the data type  In java, Method Overloading is not possible by changing the return type of the method only 1) Method Overloading: changing no. of arguments  In this example, we have created two methods, first add() method performs addition of two numbers and second add method performs addition of three numbers.  In this example, we are creating static methods so that we don't need to create instance for calling methods.   1. **class** Adder{ 2. **static** **int** add(**int** a,**int** b) 3. { 4. **return** a+b; 5. } 6. **static** **int** add(**int** a,**int** b,**int** c) 7. { 8. **return** a+b+c; }  } 9. **class** TestOverloading1{ 10. **public** **static** **void** main(String[] args){ 11. System.out.println(Adder.add(11,11)); 12. System.out.println(Adder.add(11,11,11)); }}  2) Method Overloading: changing data type of arguments In this example, we have created two methods that differs in data type. The first add method receives two integer arguments and second add method receives two double arguments.   1. **class** Adder{ 2. **static** **int** add(**int** a, **int** b){**return** a+b;} 3. **static** **double** add(**double** a, **double** b){**return** a+b;} 4. } 5. **class** TestOverloading2{ 6. **public** **static** **void** main(String[] args){ 7. System.out.println(Adder.add(11,11)); 8. System.out.println(Adder.add(12.3,12.6)); 9. }}  Q) Why Method Overloading is not possible by changing the return type of method only? In java, method overloading is not possible by changing the return type of the method only because of ambiguity. Note: Compile Time Error is better than Run Time Error. So, java compiler renders compiler time error if you declare the same method having same parameters.Can we overload java main() method? Yes, by method overloading. You can have any number of main methods in a class by method overloading. But JVM calls main() method which receives string array as arguments only.  **4)Method Overriding in Java**  If subclass (child class) has the same method as declared in the parent class, it is known as **method overriding in java**.  In other words, If subclass provides the specific implementation of the method that has been provided by one of its parent class, it is known as method overriding.  Usage of Java Method Overriding   * Method overriding is used to provide specific implementation of a method that is already provided by its super class. * Method overriding is used for runtime polymorphism  Rules for Java Method Overriding  1. method must have same name as in the parent class 2. method must have same parameter as in the parent class. 3. must be IS-A relationship (inheritance).   Can we override static method?  No, static method cannot be overridden. It can be proved by runtime polymorphism, so we will learn it later.  Why we cannot override static method?  because static method is bound with class whereas instance method is bound with object. Static belongs to class area and instance belongs to heap area.  Can we override java main method?  No, because main is a static method.  5)Inheritance in Java **Inheritance in java** is a mechanism in which one object acquires all the properties and behaviors of parent object.  The idea behind inheritance in java is that you can create new classes that are built upon existing classes. When you inherit from an existing class, you can reuse methods and fields of parent class, and you can add new methods and fields also.  Inheritance represents the **IS-A relationship**, also known as *parent-child* relationship.  Why use inheritance in java   * For Method Overriding (so runtime polymorphism can be achieved). * For Code Reusability.  Syntax of Java Inheritance  1. **class** Subclass-name **extends** Superclass-name 2. { 3. //methods and fields 4. }   The **extends keyword** indicates that you are making a new class that derives from an existing class. The meaning of "extends" is to increase the functionality.  In the terminology of Java, a class which is inherited is called parent or super class and the new class is called child or subclass 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. We will learn about interfaces later.  C:\Users\Admin\Desktop\edit1.jpgC:\Users\Admin\Desktop\edit3.jpgC:\Users\Admin\Desktop\edit2.jpg |

When a class extends multiple classes i.e. known as multiple inheritance.

   
1)**Single inheritance** enables a derived class to inherit properties and behavior from a single parent class.

2) **Multiple Inheritance** is a feature of object oriented concept, where a class can inherit properties of more than one parent class.

3) When a class extends a class, which extends anther class then this is called **multilevel inheritance**.

4) When more than one classes inherit a same class then this is called **hierarchical** **inheritance.**

5) **Hybrid inheritance**is a combination of both **single inheritance** and **multiple inheritance.**

## Why multiple inheritance 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 same method and you call it from child class object, there will be ambiguity to call 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 now.

# 6)Polymorphism in Java

**Polymorphism in java** is a concept by which we can perform a single action by different ways. There are two types of polymorphism in java: compile time polymorphism and runtime polymorphism. We can perform polymorphism in java by method overloading and method overriding.

## Compile time Polymorphism (or Static polymorphism)

Polymorphism that is resolved during compiler time is known as static polymorphism. Method overloading is an example of compile time polymorphism.  
**Method Overloading**: This allows us to have more than one method having the same name, if the parameters of methods are different in number, sequence and data types of parameters.

## Runtime Polymorphism (or Dynamic polymorphism)

It is also known as Dynamic Method Dispatch. Dynamic polymorphism is a process in which a call to an overridden method is resolved at runtime, thats why it is called runtime polymorphism.

### Upcasting

When reference variable of Parent class refers to the object of Child class, it is known as upcasting

**7)Exception Handling in Java**

The **exception handling in java** is one of the powerful *mechanism to handle the runtime errors* so that normal flow of the application can be maintained. Exception Handling is a mechanism to handle runtime errors such as ClassNotFound, IO, SQL, Remote etc.

### Advantage of Exception Handling

The core advantage of exception handling is **to maintain the normal flow of the application**. Exception normally disrupts the normal flow of the application that is why we use exception handling.

Types of Exception

There are mainly two types of exceptions: checked and unchecked where error is considered as unchecked exception. The sun microsystem says there are three types of exceptions:

1. Checked Exception
2. Unchecked Exception
3. Error

### 1) Checked Exception

The classes that extend Throwable class except RuntimeException and Error are known as checked exceptions e.g.IOException, SQLException etc. Checked exceptions are checked at compile-time.

### 2) Unchecked Exception

The classes that extend RuntimeException are known as unchecked exceptions e.g. ArithmeticException, NullPointerException, ArrayIndexOutOfBoundsException etc. Unchecked exceptions are not checked at compile-time rather they are checked at runtime.

### 3) Error

Error is irrecoverable e.g. OutOfMemoryError, VirtualMachineError, AssertionError etc.

### Common scenarios where exceptions may occur

### 1) Scenario where ArithmeticException occurs

If we divide any number by zero, there occurs an ArithmeticException

**Ex: int** a=50/0;//ArithmeticException

### Scenario where NullPointerException occurs

If we have null value in any variable, performing any operation by the variable occurs an NullPointerException.

Ex: String s=**null**;

System.out.println(s.length());//NullPointerException

### Scenario where NumberFormatException occurs

The wrong formatting of any value, may occur NumberFormatException. Suppose I have a string variable that have characters, converting this variable into digit will occur NumberFormatException.

Ex: String s="abc";

**int** i=Integer.parseInt(s);//NumberFormatException

### Scenario where ArrayIndexOutOfBoundsException occurs

If you are inserting any value in the wrong index, it would result ArrayIndexOutOfBoundsException

**Ex: int** a[]=**new** **int**[5];

a[10]=50; //ArrayIndexOutOfBoundsException

## Java Exception Handling Keywords

There are 5 keywords used in java exception handling.

1. try
2. catch
3. finally
4. throw
5. throws

## Java try block

Java try block is used to enclose the code that might throw an exception. It must be used within the method.

Java try block must be followed by either catch or finally block.

#### Syntax

**try**{

//code that may throw exception

}**catch**(Exception\_class\_Name ref){}

#### Syntax of try-finally block

**try**{

//code that may throw exception

}**finally**{}

## Java catch block

Java catch block is used to handle the Exception. It must be used after the try block only.

You can use multiple catch block with a single try.

**Internal working of java try-catch block**

The JVM firstly checks whether the exception is handled or not. If exception is not handled, JVM provides a default exception handler that performs the following tasks:

* Prints out exception description.
* Prints the stack trace (Hierarchy of methods where the exception occurred).
* Causes the program to terminate.

But if exception is handled by the application programmer, normal flow of the application is maintained i.e. rest of the code is executed.

## Java Multi catch block

If you have to perform different tasks at the occurrence of different Exceptions, use java multi catch block.

# Java Nested try block

The try block within a try block is known as nested try block in java.

### Why use nested try block

Sometimes a situation may arise where a part of a block may cause one error and the entire block itself may cause another error. In such cases, exception handlers have to be nested.

### Syntax:

**try**

{

  statement 1;

   statement 2;

**try**

    {

        statement 1;

        statement 2;

    }

**catch**(Exception e)

    {

    }

}

**catch**(Exception e)

{

}

# Java finally block

**Java finally block** is a block that is used to execute important code such as closing connection, stream etc.

Java finally block is always executed whether exception is handled or not.

Java finally block follows try or catch block.

## Why use java finally

* Finally block in java can be used to put "cleanup" code such as closing a file, closing connection etc.

# Java throw exception

The Java throw keyword is used to explicitly throw an exception.

We can throw either checked or uncheked exception in java by throw keyword. The throw keyword is mainly used to throw custom exception. We will see custom exceptions later.

The syntax of java throw keyword is given below.

**throw** exception;

# Java throws keyword

The **Java throws keyword** is used to declare an exception. It gives an information to the programmer that there may occur an exception so it is better for the programmer to provide the exception handling code so that normal flow can be maintained.

Exception Handling is mainly used to handle the checked exceptions. If there occurs any unchecked exception such as NullPointerException, it is programmers fault that he is not performing check up before the code being used

### Syntax

return\_type method\_name() **throws** exception\_class\_name{

//method code

}

Which exception should be declared

**Ans)** checked exception only, because:

* **unchecked Exception:** under your control so correct your code.
* **error:** beyond your control e.g. you are unable to do anything if there occurs VirtualMachineError or StackOverflowError.

### Advantage of Java throws keyword

Now Checked Exception can be propagated (forwarded in call stack).

It provides information to the caller of the method about the exception.

**8)Multithreading in Java**

**Multithreading in java** is a process of executing multiple threads simultaneously.

Thread is basically a lightweight sub-process, a smallest unit of processing. Multiprocessing and multithreading, both are used to achieve multitasking.

But we use multithreading than multiprocessing because threads share a common memory area. They don't allocate separate memory area so saves memory, and context-switching between the threads takes less time than process.

Java Multithreading is mostly used in games, animation etc.

### Advantages of Java Multithreading

1) It **doesn't block the user** because threads are independent and you can perform multiple operations at same time.

2) You **can perform many operations together so it saves time**.

3) Threads are **independent** so it doesn't affect other threads if exception occur in a single thread.

## Multitasking

Multitasking is a process of executing multiple tasks simultaneously. We use multitasking to utilize the CPU. Multitasking can be achieved by two ways:

* Process-based Multitasking(Multiprocessing)
* Thread-based Multitasking(Multithreading)

### 1) Process-based Multitasking (Multiprocessing)

* Each process have its own address in memory i.e. each process allocates separate memory area.
* Process is heavyweight.
* Cost of communication between the process is high.
* Switching from one process to another require some time for saving and loading registers, memory maps, updating lists etc.

### 2) Thread-based Multitasking (Multithreading)

* Threads share the same address space.
* Thread is lightweight.
* Cost of communication between the thread is low.

## What is Thread in java

A thread is a lightweight sub process, a smallest unit of processing. It is a separate path of execution.

Threads are independent, if there occurs exception in one thread, it doesn't affect other threads. It shares a common memory area.

Life cycle of a Thread (Thread States)

A thread can be in one of the five states. According to sun, there is only 4 states in **thread life cycle in java** new, runnable, non-runnable and terminated. There is no running state.

But for better understanding the threads, we are explaining it in the 5 states.

The life cycle of the thread in java is controlled by JVM. The java thread states are as follows:

1. New
2. Runnable
3. Running
4. Non-Runnable (Blocked)
5. Terminate

|  |
| --- |
| 1) New  The thread is in new state if you create an instance of Thread class but before the invocation of start() method. |

2) Runnable

The thread is in runnable state after invocation of start() method, but the thread scheduler has not selected it to be the running thread.

3) Running

The thread is in running state if the thread scheduler has selected it.

4) Non-Runnable (Blocked)

This is the state when the thread is still alive, but is currently not eligible to run.

5) Terminated