Defining char array

char[] charArray={'a','b','c','d','e'};

char[] revCharArray = new char[len];

**Typecasting** and **Upcasting** and **Downcasting**

A process of converting one data type to another is known as **Typecasting** and **Upcasting** and **Downcasting** is the type of object typecasting. In Java, the object can also be typecasted like the datatypes. **Parent** and **Child** objects are two types of objects. So, there are two types of typecasting possible for an object, i.e., **Parent to Child** and **Child to Parent** or can say **Upcasting** and **Downcasting**.

**Finalize**

Finalize method in Java is **an Object Class method that is used to perform cleanup activity before destroying any object**. It is called by Garbage collector before destroying the object from memory. Finalize() method is called by default for every object before its deletion.

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

**Final keyword**

If we initialize a variable with the final keyword, then we cannot modify its value.

If we declare a method as final, then it cannot be overridden by any subclasses.

Final class cannot be inherited.

**final:**

final is a keyword. The variable decleared as final should be initialized only once and cannot be changed. Java classes declared as final cannot be extended. Methods declared as final cannot be overridden.

**finally:**

finally is a block. The finally block always executes when the try block exits. This ensures that the finally block is executed even if an unexpected exception occurs. But finally is useful for more than just exception handling - it allows the programmer to avoid having cleanup code accidentally bypassed by a return,

continue, or break. Putting cleanup code in a finally block is always a good practice, even when no exceptions are anticipated.

**finalize:**

finalize is a method. Before an object is garbage collected, the runtime system calls its finalize() method. You can write system resources release code in finalize() method before getting garbage

collected.

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

**Can we have a try block without a catch block in Java?**

Yes, It is possible to have a try block without a catch block by using a final block.

As we know, a final block will always execute even there is an exception occurred in a try block, except System.exit() it will execute always.

**Java Custom Exception**

**public** **class** InvalidAgeException **extends** Exception {

**public** WrongFileNameException(String errorMessage) {

**super**(errorMessage);

    }

}

Class1

Create a class by extending Exception class

Class2

Write validate condition. If condition fails invoke throw method

**throw** **new** InvalidAgeException("age is not valid to vote");

**Difference Between throw, throws and Throwable In Java**

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. Below example shows how to throw an exception using throw keyword.

class ThrowAndThrowsExample

{

void method() throws Exception

{

Exception e = new Exception();

throw e; //throwing an exception using 'throw'

}

}

**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. Below is the syntax for using throws keyword.

return\_type method\_name(parameter\_list) throws exception\_list

{

//some statements

}

**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. Click here to see the hierarchy of exception classes in java.

Below example shows how to create customized exceptions by extending java.lang.Throwable class.

class MyException extends Throwable

{

//Customized Exception class

}

class ThrowAndThrowsExample

{

void method() throws MyException

{

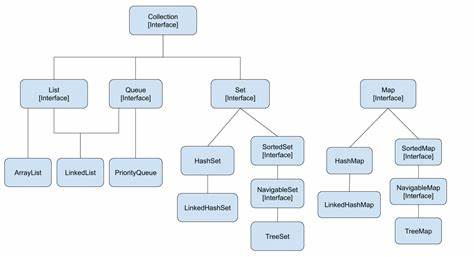
MyException e = new MyException();

throw e;

}

}

**Collections in java**



**What is Abstract Class?**

Abstract Class is a type of class in OOPs, that declare one or more abstract methods. These classes can have abstract methods as well as concrete methods. A normal class cannot have abstract methods. An abstract class is a class that contains at least one abstract method.

**Can we create abstract classes without any abstract methods?**

Yes, we can create abstract classes without any abstract methods.

# What is abstract class or abstract method?

## Difference between Abstract Class and 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. |

**Singleton Class in Java**

In object-oriented programming, a singleton class is a class that can have only one object (an instance of the class) at a time. After the first time, if we try to instantiate the Singleton class, the new variable also points to the first instance created

Purpose of Singleton Class

The primary purpose of a Singleton class is to restrict the limit of the number of object creation to only one. This often ensures that there is access control to resources, for example, socket or database connection.

**Collection in java**



**Java Collections : Interface**

Iterator interface : Iterator is an interface that iterates the elements. It is used to traverse the list and modify the elements. Iterator interface has three methods which are mentioned below:

public boolean hasNext() – This method returns true if the iterator has more elements.

public object next() – It returns the element and moves the cursor pointer to the next element.

public void remove() – This method removes the last elements returned by the iterator.

## **Java collections: List**

A List is an ordered Collection of elements which may contain duplicates. It is an interface that extends the Collection interface. Lists are further classified into the following:

1. ArrayList
2. LinkedList
3. Vectors

**Array list:** ArrayList is the implementation of List Interface where the elements can be dynamically added or removed from the list. Also, the size of the list is increased dynamically if the elements are added more than the initial size.

**Syntax**:

ArrayList object = new ArrayList ();  
Some of the methods in array list are listed below:

|  |  |
| --- | --- |
| **Method** | **Description** |
| boolean add(Collection c) | Appends the specified element to the end of a list. |
| void add(int index, Object element) | Inserts the specified element at the specified position. |
| void clear() | Removes all the elements from this list. |
| int lastIndexOf(Object o) | Return the index in this list of the last occurrence of the specified element, or -1 if the list does not contain this element. |
| Object clone() | Return a shallow copy of an ArrayList. |
| Object[] toArray() | Returns an array containing all the elements in the list. |
| void trimToSize() | Trims the capacity of this ArrayList instance to be the list’s current size. |

**Linked List:**

In Java, a LinkedList is a data structure that stores elements in a non-contiguous location. It is a linear data structure.

Linked List is a sequence of links which contains items. Each link contains a connection to another link.

**Enlisted below are some of the characteristics of the LinkedList class that you should remember:**

* This class is not synchronized.
* It allows duplicate values.
* Retains the insertion order.
* As elements are not required to be shifted while moving, the manipulation of elements in it is faster.
* This class can be used to implement a stack, queue, and list.

*Syntax*: Linkedlist object = new Linkedlist();

Java Linked List class uses two types of Linked list to store the elements:

* Singly Linked List
* Doubly Linked List

**Singly Linked List**: In a singly Linked list each node in this list stores the data of the node and a pointer or reference to the next node in the list.

**Some of the methods in the linked list are listed below:**

LinkedList<String> colorsList = new LinkedList();

Collections.sort(colorsList); : To sort the linked list

colorsList.add(null)d

colorsList.addAll(colorsList)

colorsList.addFirst(null);d

colorsList.addLast(null);

colorsList.clear();

colorsList.contains(colorsList)

colorsList.getFirst()

colorsList.indexOf(colorsList)

colorsList.isEmpty()

colorsList.pop()

colorsList.size()

colorsList.remove(2)

**We will discuss the main operations/methods in the following section.**

| Method | Prototype | Description |
| --- | --- | --- |
| Add | boolean add (E e) | Add a specified element to the LinkedList |
| void add (int index, E element) | Add element at the given index in LinkedList |
| AddAll | boolean addAll (Collection < ? extends E > c) | Adds the elements of given collection c at the end of the LinkedList. |
| boolean addAll (int index, Collection < ? extends E > c) | Adds the elements of given collection c at the specified index in the LinkedList |
| addFirst | void addFirst (E e) | Add the given element as the first element to the LinkedList. |
| addLast | void addLast (E e) | Append the given element at the end of the list. |
| Clear | void clear () | Deletes all the elements from the list. |
| Clone | Object clone () | Makes a shallow copy of LinkedList |
| Contains | Boolean contains (Object o) | Checks if the list contains specified elements; if yes returns true. |
| descendingIterator | Iterator < E > descendingIterator () | Returns a reverse ordered iterator for the LinkedList. |
| Element | E element () | Returns the element at the head of the list. |
| Get | E get (int index) | Gets the element at the specified index. |
| getFirst | E getFirst () | Retrieves the first element in the LinkedList. |
| getLast | E getLast () | Retrieves the last element in the LinkedList. |
| indexOf | Int indexOf (Object o) | Find the index of the first occurrence of the given elements in the list and return the index. -1 if element not found. |
| lastIndexOf | Int lastIndexOf (Object o) | Returns the position of the last occurrence of the given element in the LinkedList;-1 if given element is not present |
| listIterator | ListIterator < E > listIterator (int index) | Returns the listIterator from the specified index in the linkedlist. |
| Offer | boolean offer (E e) | Adds the given element as the last element (tail) in the LinkedList. |
| offerFirst | Boolean offerFirst (E e) | Adds the given element as the first element in the LinkedList. |
| offerLast | Boolean offerLast (E e) | Add given element e at the end of the LinkedList. |
| Peek | E peek () | Returns the head of the list without removing it. |
| peekFirst | E peekFirst () | Returns the first element in the list. returns null if the list is empty. |
| peekLast | E peekLast () | Returns the last element or null if the list is empty. It does not delete the element. |
| Poll | E poll () | Returns the head of the LinkedList and also removes it. |
| pollFirst | E pollFirst () | Returns and deletes the first element in the list; returns null if the list is empty. |
| pollLast | E pollLast () | Returns and deletes the last element in the list; returns null if the list is empty. |
| Pop | E pop () | Pops the element from the stack representation of LinkedList. |
| Push | Void push (E e) | Pushes or inserts an element into the stack representation of the LinkedList. |
| Remove | E remove () | Removes and returns the head of the LinkedList. |
| E remove (int index) | Deletes the element at the given index from the LinkedList. |
| boolean remove (Object o) | Deletes the first occurrence of the given element from the LinkedList. |
| removeFirst | E removeFirst () | Returns and deletes the first element from the list. |
| removeFirstOccurence | boolean removeFirstOccurrence (Object o) | Deletes the first occurrence of the given element from the list when the list is being traversed from head to tail. |
| removeLast | E removeLast () | Returns the last element in the LinkedList and also deletes it. |
| removeLastOccurence | boolean removeLastOccurrence (Object o) | Removes the last occurrence of the given element from the LinkedList when traversed from head to tail |
| Set | E set (int index, E element) | Sets the given element at the given index. Replaces the current element with new. |
| Size | Int size () | Returns size or number of elements in the LinkedList |
| toArray | Object[] toArray () | Converts the LinkedList to an array containing all list elements in proper sequence |
| < T > T [] toArray( T [] a) | Converts LinkedList to an array with runtime type same as argument a. |

|  |  |
| --- | --- |
| **The below program reverses the linked list using the descendingIterator () method.**   |  | | --- | | **import** java.util.\*;  **public** **class** Main{  **public** **static** **void** main(String args[]){      //create a LinkedList object      LinkedList&lt;String&gt; l\_list=**new** LinkedList&lt;String&gt;();      l\_list.add("Pune");      l\_list.add("Mumbai");      l\_list.add("Nagpur");      System.out.println("Linked List : " + l\_list);      System.out.println("Linked List in reverse order:");      //use descendingIterator method to get a reverse iterator      Iterator iter=l\_list.descendingIterator();      //traverse the list using iterator and print the elements.  **while**(iter.hasNext())      {          System.out.print(iter.next() + " ");      }   }  } | |

**When to use LinkedList or ArrayList?**

Accessing elements are faster with ArrayList, because it is index based. But accessing is difficult with LinkedList. It is slow access. This is to access any element, you need to navigate through the elements one by one. But insertion and deletion is much faster with LinkedList, because if you know the node, just change the pointers before or after nodes. Insertion and deletion is slow with ArrayList, this is because, during these operations ArrayList need to adjust the indexes according to deletion or insetion if you are performing on middle indexes. Means, an ArrayList having 10 elements, if you are inserting at index 5, then you need to shift the indexes above 5 to one more.

**Doubly Linked List**: In a doubly Linked list, it has two references, one to the next node and another to previous node. You can refer to the below image to get a better understanding of doubly linked list.

Some of the methods in the linked list are listed below:

|  |  |
| --- | --- |
| **Method** | **Description** |
| boolean add( Object o) | It is used to append the specified element to the end of the vector. |
| boolean contains(Object o) | Returns true if this list contains the specified element. |
| void add (int index, Object element) | Inserts the element at the specified element in the vector. |
| void addFirst(Object o) | It is used to insert the given element at the beginning. |
| void addLast(Object o) | It is used to append the given element to the end. |
| int size() | It is used to return the number of elements in a list |
| boolean remove(Object o) | Removes the first occurrence of the specified element from this list. |
| int indexOf(Object element) | Returns the index of the first occurrence of the specified element in this list, or -1. |
| int lastIndexOf(Object element) | Returns the index of the last occurrence of the specified element in this list, or -1. |

## 

**Vectors** : Vectors are similar to arrays, where the elements of the vector object can be accessed via an index into the vector. Vector implements a dynamic array. Also, the vector is not limited to a specific size, it can shrink or grow automatically whenever required. It is similar to ArrayList, but with two differences :

* Vector is synchronized.
* Vector contains many legacy methods that are not part of the collections framework.

**Syntax**:

Vector object = new Vector(size,increment);

Below are some of the methods of the Vector class:

|  |  |
| --- | --- |
| **Method** | **Description** |
| boolean add(Object o) | Appends the specified element to the end of the list. |
| void clear() | Removes all of the elements from this list. |
| void add(int index, Object element) | Inserts the specified element at the specified position. |
| boolean remove(Object o) | Removes the first occurrence of the specified element from this list. |
| boolean contains(Object element) | Returns true if this list contains the specified element. |
| int indexOfObject (Object element) | Returns the index of the first occurrence of the specified element in the list, or -1. |
| int size() | Returns the number of elements in this list. |
| int lastIndexOf(Object o) | Return the index of the last occurrence of the specified element in the list, or -1 if the list does not contain any element. |

**How to get current time in milli seconds?**

System.currentTimeMillis() returns the current time in milliseconds.

It is a static method, returns long type.

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

**String vs StringBuilder vs StringBuffer**

**String is immutable whereas StringBuffer and StringBuilder are mutable classes**. StringBuffer is thread-safe and synchronized whereas StringBuilder is not. That's why StringBuilder is faster than StringBuffer. String concatenation operator (+) internally uses StringBuffer or StringBuilder class.

String str= new String("abc");

char [] charArray = str.toCharArray();

then reverse;

StringBuilder sb=new StringBuilder( "aaa");

sb.reverse();

StringBuffer sbuff=new StringBuffer( "aaa");

sbuff.reverse();

String Vs StringBuffer

We know that String is immutable object. We can not change the value of a String object once it is initiated. If we try to change the value of the existing String object then it creates new object rather than changing the value of the existing object. So incase, we are going to do more modificatios on String, then use StringBuffer. StringBuffer updates the existing objects value, rather creating new object.

**Static**

In the Java programming language, the keyword static means that **the particular member belongs to a type itself, rather than to an instance of that type**. This means we'll create only one instance of that static member that is shared across all instances of the class.

There are some important points that you need to consider when you work with static methods in Java. These are -

1. The static methods of a particular class can only access the static variables and can change them.
2. A static method can only call other static methods.
3. Static methods can’t refer to non-static variables or methods.
4. Static methods can’t refer to “super” or “this” members.

S**tatic vs Non-Static**

Before moving ahead with this Static in Java tutorial to discuss static blocks and classes in Java, let’s discuss a few differences between static and non-static variables and methods in Java.

|  |  |
| --- | --- |
| **Static Variables** | **Non-Static Variables** |
| They can access them using class names | They can be accessed only using objects. |
| They can access them with static methods as well as non-static methods. | They can be accessed only using non-static methods. |
| They are allocated memory only once while loading the class. | A memory per object is allocated. |
| These variables are shared by all the objects or instances of the class. | Each object has its own copy of the non-static variables. |
| Static variables have global scope. | They have local scope. |
| **Static Methods** | **Non-Static Methodsnon-Static Methods** |
| These methods support early or compile-time binding. | They support late, run-time, or dynamic binding. |
| These methods can only access static variables of other classes as well as their own class. | They can access both static as well as non-static members. |
| You can’t override static methods. | They can be overridden. |
| Less memory consumption since they are allocated memory only once when the class is being loaded. | Memories are allocated for each object. |

**Static Blocks in Java**

Generally, static blocks in Java are used to initialize static variables. They are executed only once when the class is loaded and hence, are perfect for this job. Also, you can include more than one static block in the class. Static blocks can only access static variables. Let’s understand static blocks using the below example.

class Test{

static int i = 10;

static int j;

static{

System.out.println("Initializing the Static Variable using Static Block ...");

j = i \* 5;

}

}

class Main{

public static void main(String args[]){

System.out.println("Value of i is: " + Test.i);

System.out.println("Value of j is: " + Test.j);

}

}

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

**Try Catch Finally**

**In case, there is a return at the end of try block, will execute finally block?**

Yes, the finally block will be executed even after writing return statement

at the end of try block. It returns after executing finally block.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
**Java Iterator**

An Iterator is an object that can be used to loop through collections, like ArrayList and HashSet. It is called an "iterator" because "iterating" is the technical term for looping.

To use an Iterator, you must import it from the java.util package.

Getting an Iterator

The iterator() method can be used to get an Iterator for any collection:

|  |
| --- |
| public class Main {  public static void main(String[] args) {  // Make a collection  ArrayList<String> cars = new ArrayList<String>();  cars.add("Volvo");  cars.add("BMW");  cars.add("Ford");  cars.add("Mazda");  // Get the iterator  Iterator<String> it = cars.iterator();  // Print the first item  System.out.println(it.next());  }  } |

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

**Enumeration**

means **a list of named constant**. In Java, enumeration defines a class type. An Enumeration can have constructors, methods and instance variables. It is created using enum keyword. Each enumeration constant is public, static and final by default.

Enumerations serve the purpose of representing a group of named constants in a programming language.

**What is the difference between Enumeration and Iterator?**

The functionality of Enumeration and the Iterator are same. You can get remove()

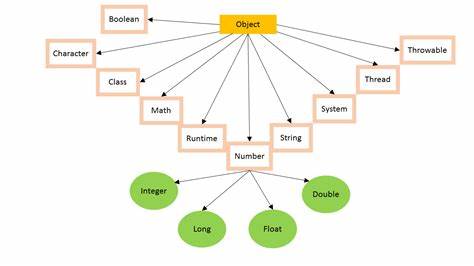
from Iterator to remove an element, while while Enumeration does not have remove()

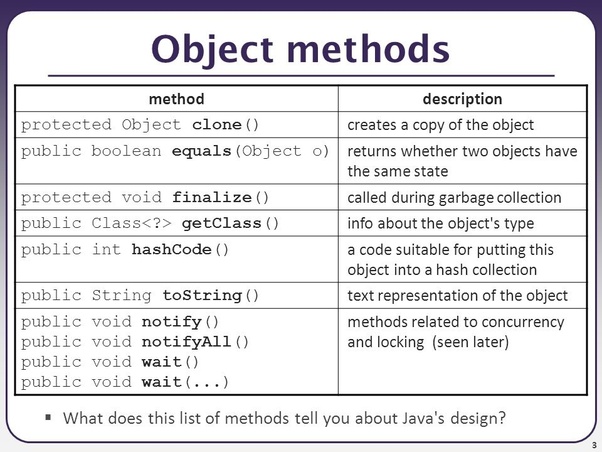
method. Using Enumeration you can only traverse and fetch the objects, where as using

Iterator we can also add and remove the objects. So Iterator can be useful if you want

to manipulate the list and Enumeration is for read-only access.

**Object class hiearchy**





**Can you convert set to list**

Using Constructor (ArrayList or LinkedList Class)

The logic is the same as above. The only difference is that we have used the constructor of the ArrayList and LinkedList class and passed set elements to the constructor.

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

List<String> lList = new LinkedList<String>(set);

Contents

[What is abstract class or abstract method? 4](#_Toc110777580)

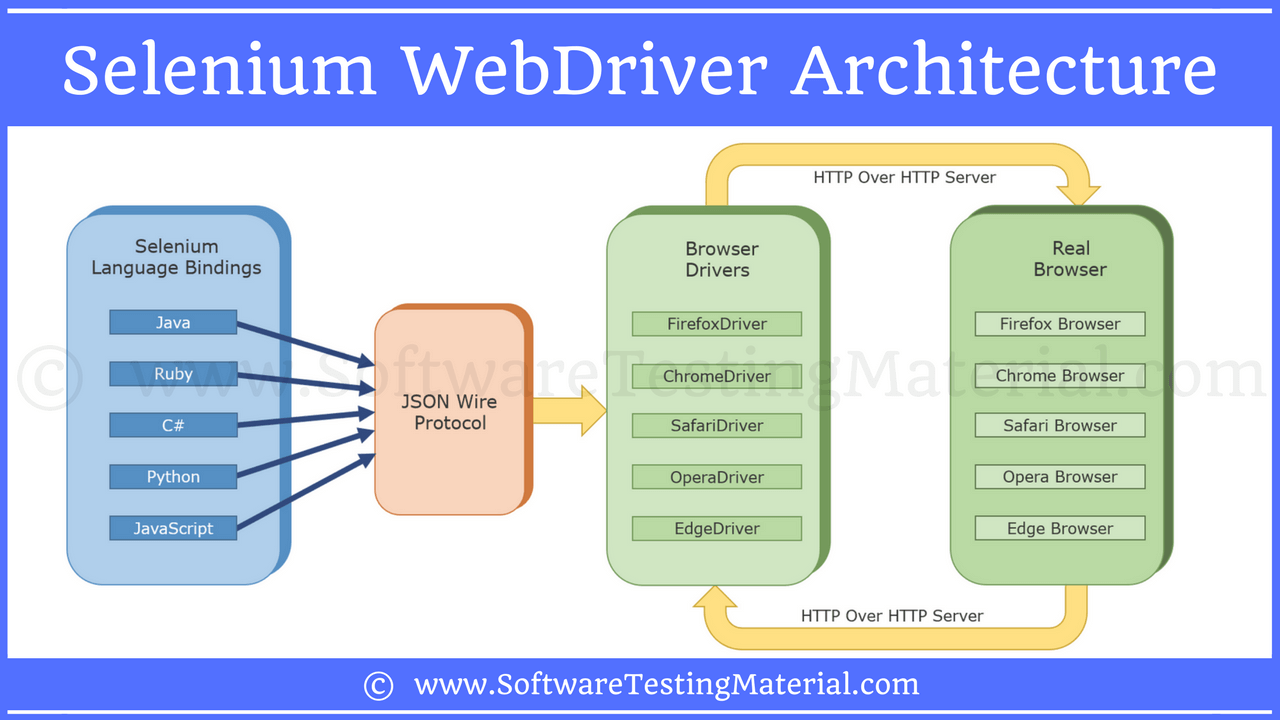
[Difference between Abstract Class and Interface 4](#_Toc110777581)

[Java collections: List 5](#_Toc110777582)

**End of Java Points**

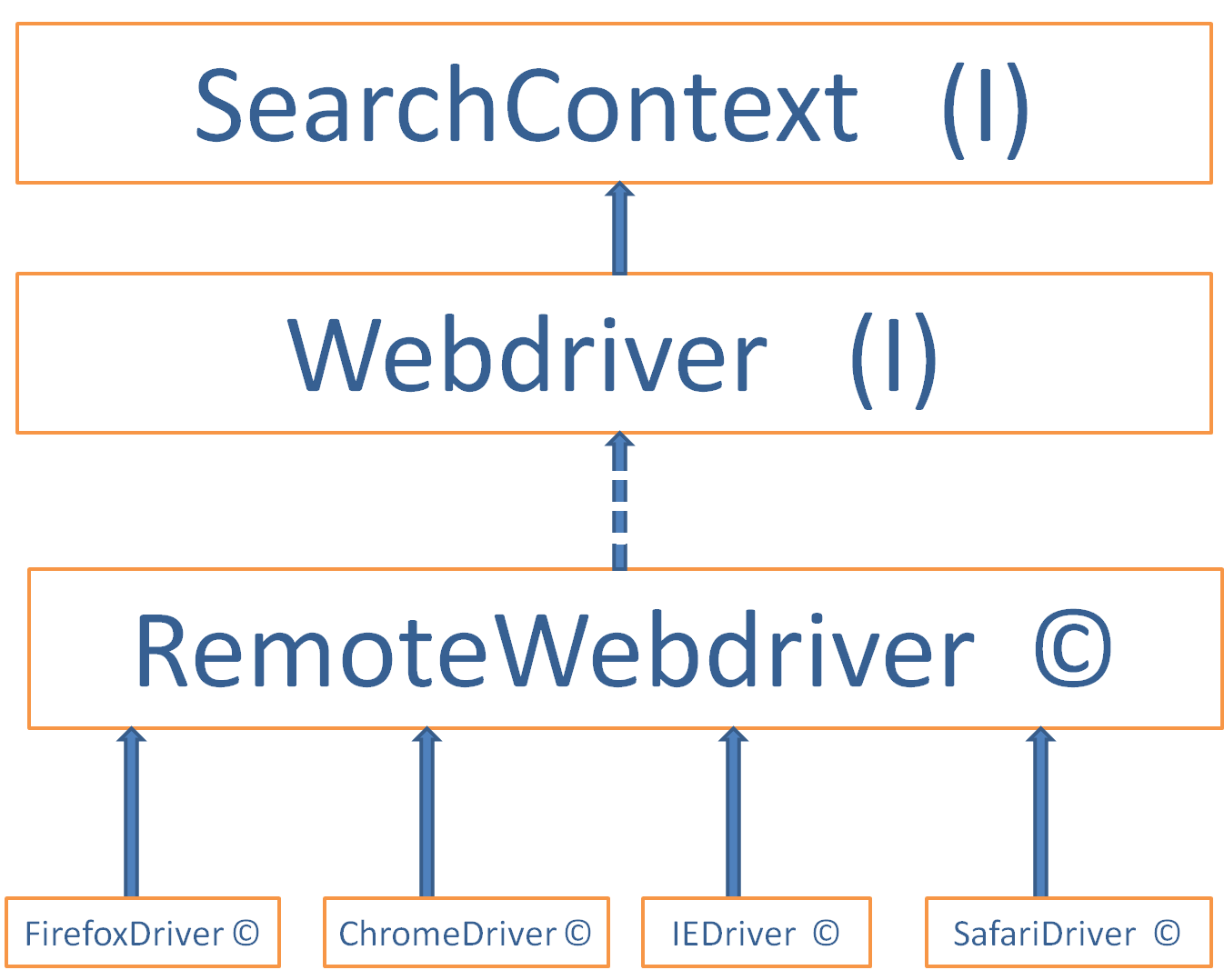
**Selenium Points**

**Selenium Architecture**

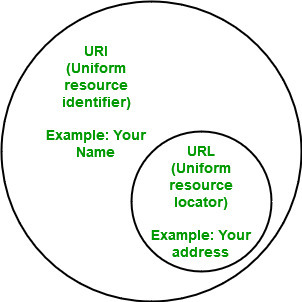


**How seleium interacts with the browser**

**Webdriver api doc**



**URL and URI**



A URL provides the location of the resource. A URI identifies the resource by name at the specified location or URL. Other URIs provide only a unique name, without a means of locating or retrieving the resource or information about it, these are Uniform Resource Names (URNs).

An example of a URI can be ISBN 0-486-35557-4.

An example of an URL is https://www.javatpoint.com.

**HTTP vs HTTPS**

HTTP requests and responses are sent across the Internet in plaintext. The problem is that anyone monitoring the connection can read these plaintexts.

HTTPS stands for Hypertext Transfer Protocol Secure (also referred to as HTTP over TLS or HTTP over SSL). HTTPS uses TLS (or SSL) to encrypt HTTP requests and responses, so instead of the plaintext, an attacker would see a series of seemingly random characters.

**POM design pattern**

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. In Page Object Model, consider each web page of an application as a class file.

In POM design pattern

Page class

Idenfifying page elements

Actions for the page elements

Test class

Create object of page class

Call action methods

Write Assertions

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

**Test Driven Development (TDD)** is software development approach in which test cases are developed to specify and validate what the code will do. In simple terms, test cases for each functionality are created and tested first and if the test fails then the new code is written in order to pass the test and making code simple and bug-free.

The simple concept of TDD is to write and correct the failed tests before writing new code (before development). This helps to avoid duplication of code as we write a small amount of code at a time in order to pass tests.

**Acceptance test–driven development** (**ATDD**) is a [development](https://en.wikipedia.org/wiki/Software_development) methodology based on communication between the business customers, the developers, and the testers.

BDD

Test scenarios in BDD are based on the concept of “given, when, then” to describe various user BDD scenarios. Test cases written so, in simple text, help even non-technical team members to understand what is going on in the software project, with or without a [test management tool](https://www.practitest.com/). A common usage of the BDD approach is with automation using Gherkin and combining it with unit testing

**Explain main 5 features of selenium**

1.enables cross browsing

2. Paralel testing

3. we can use any framework

**How do you achieve parallel testing ?**

 Below is the structure for defining this attribute in the TestNG XML:

<suite name="Parallel\_Testing" parallel="methods" thread-count="2">

The parallel attribute can be extended for multiple values, as below:

* **Methods**: Helps run methods in separate threads
* **Tests**: Help to run all methods belonging to the same tag in the same thread
* **Classes**: Helps to run all methods belonging to a class in a single thread
* **Instances**: Helps run all methods in the same instance in the same thread

**Locators**

**If you have a text box whose ID is changing dynamically, how to you find the element**

If the id is changing dynamically for a particular xpath then we can use the below xpath functions:

contains :xpath Syntax: //html tag[contains(text()/@attribute, 'Expected Value')]

Following-Siblings:

Xpath Syntax : //html tag/following-siblings::html tag

ex: //img[@src = 'image2.jpg']/following-siblings::input[@type = 'button']

Preceding-Siblings: syntax will be same as following-Siblings

The descendant function which is used to identify the grand child node by skipping all

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

**How do you handle ElementNotInteractableException  exception**

Avoiding-And-Handling: There are two ways to do this. We can either use wait for the element to get completely.

The below code waits 10 seconds for the element. If the element is visible and still exception is thrown, it is caught.

try {

WebDriverWait wait = new WebDriverWait(driver, TimeSpan.FromSeconds(10));

wait.Until(ExpectedConditions.visibilityOfElementLocated(By.id(”submit”));

try {

driver.findElement(By.id("submit")).click();

} catch (WebDriverException e) {

System.out.println(“Exceptional case”);

}

} catch (TimeOutException e)

System.out.println(“WebDriver couldn’t find this element visible”);

}

**How to avoid “StaleElementReferenceException” in Selenium?**

Page refresh.

Having a retry mechanism.

Having a try-catch block.

Waiting for some expected criteria like presenceOfElementLocated or refreshing a page on getting a stale condition for an element.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*8

**End of Selenium points**

**Start of Cucumber**

**In Cucumber**

**What is the difference between scenario and scenario outline**

The **scenario** is one of the core structures of the Gherkin language. Scenario includes all the possible circumstances of the [feature](https://www.javatpoint.com/feature-in-cucumber-testing) and test scripts for these circumstances.

**Scenario outline is exactly similar to the scenario structure, but the only difference is the provision of multiple inputs**. In order to use scenario outlines, we do not need any smart idea, we just need to copy the same steps and re-execute the code.

**Hooks in cucumber**

Cucumber supports hooks, which are blocks of code that run before or after each scenario. Cucumber hooks allows us to better manage the code workflow and helps us to reduce the code redundancy.

Types of hooks

@Before and @After tagging a method with either of these will cause the method to run before or after each scenario runs. Common functionality like starting or closing browsers are nice to place in these hooks. They reduce the number of common test steps in each scenario. Before hooks will be run before the first step of each scenario. They will run in the same order of which they are registered. After hooks will be run after the last step of each scenario, even when there are failing, undefined, pending or skipped steps.

**How to do parallel testing in cucumber**

Ex: in feature we have 8 scenarios and 2 scenario outline. How do we execute it parallel

<https://cucumber.io/docs/guides/parallel-execution/?lang=java>

Cucumber can be executed in parallel using **TestNG and Maven test execution plugins** by setting the **dataprovider parallel option to true**. In TestNG the **scenarios and rows in a scenario outline are executed in multiple threads**. One can use either Maven Surefire or Failsafe plugin for executing the runners.

Add a cucumber **runner** by **extending** the AbstractTestNGCucumberTests class and **overriding the scenarios method** in the parallel package (same name as step definition package) in src/test/java folder. Set the **parallel option value to true** for the DataProvider annotation.

package parallel;

import org.testng.annotations.DataProvider;

import io.cucumber.testng.AbstractTestNGCucumberTests;

public class RunCucumberTest extends AbstractTestNGCucumberTests{

@Override

@DataProvider(parallel = true)

public Object[][] scenarios() {

return super.scenarios();

}

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*8

**Cucumber topics**

Cucumber feature file ( scenario and scenario outlinr)

Test runner class

Step definition

Cucumber options

Data Table in Cucumber?

Data driven testing using example

Data driven testing using JSON

Data tables in cucumber

Maps in data tables

Cucumber hooks (@after and @before)

Cucumber tags

with and ----- tags= {"@FirstTest", "@SecondTest"}

with or --- tags= {"@SmokeTest, @RegressionTest"}

ignore taggs --- tags={"@SmokeTest", "~@RegressionTest"}

Tagged hooks in cucumber @Before(“@scenario”)

Execution order of hooks @Before(order=1)

Background in cucumber

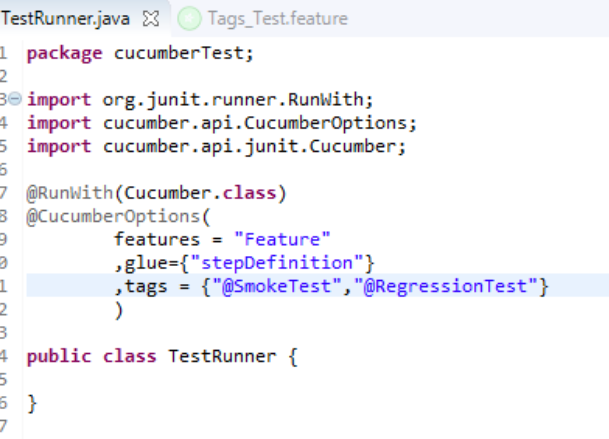
Test runner classs[@RunWith(cucumber.class @CucumberOptions(feature=”feature”, glue={“stap definition”}]

Cucumber reports

Keywords in feature files

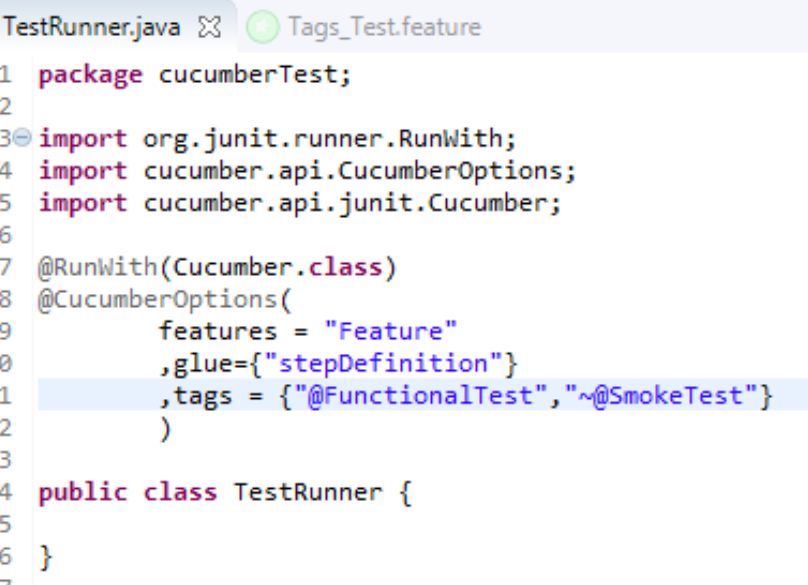
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

|  |
| --- |
| **Feature file will look like this**  Scenario: Increa@FunctionalTest  Feature: ECommerce Application  @SmokeTest @RegressionTest  Scenario: Successful Login  Given This is a blank test  @RegressionTest  Scenario: UnSuccessful Login  Given This is a blank test  @SmokeTest  Scenario: Add a product to bag  Given This is a blank test  Scenario: Add multiple product to bag  Given This is a blank test  @SmokeTest @RegressionTest  Scenario: Remove a product from bag  Given This is a blank test  @RegressionTest  Scenario: Remove all products from bag  Given This is a blank test  @SmokeTestse product quantity from bag page  Given This is a blank test  Scenario: Decrease product quantity from bag page  Given This is a blank test  @SmokeTest @End2End  Scenario: Buy a product with cash payment  Given This is a blank test  @SmokeTest @End2End  Scenario: Buy a product with CC payment  Given This is a blank test  @End2End  Scenario Outline: Payment declined  Given This is a blank test  Examples:  |PaymentMethod|  |CC Card|  |DD Card|  |Bank Transfer|  |PayPal|  |Cash| |

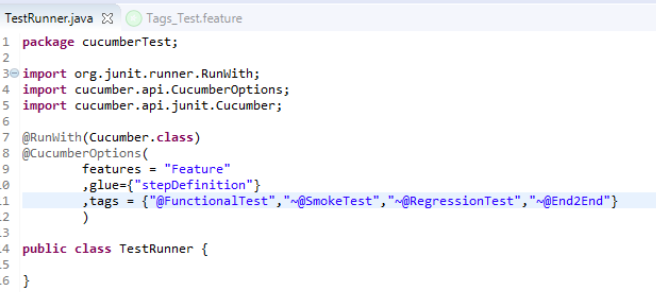


**How to ignore cucumber tests**

Special Character ~ is used to skip the tags. This also works both for Scenarios and Features. And this can also works in conjunction with AND or OR.



**Execute all tests which are not at all tagged in Single Feature**



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

**Background in Cucumber** is used to define a step or series of steps that are common to all the tests in the feature file. It allows you to add some context to the scenarios for a feature where it is defined. A Background is much like a scenario containing a number of steps. But it runs before each and every scenario were for a feature in which it is defined.

For example, to purchase a product on any E-Commerce website, you need to do the following steps:

Navigate to Login Page

Submit UserName and Password

## **Background in Cucumber**

Let's start with a simple exercise to build the understanding of Background usage in Cucumber test. If we create a feature file of the scenario we explained above, this is how it will look like:

***Feature File***

Feature: Test Background Feature

Description: The purpose of this feature is to test the Background keyword

Background: User is Logged In

Given I navigate to the login page

When I submit username and password

Then I should be logged in

Scenario: Search a product and add the first product to the User basket

Given User search for Lenovo Laptop

When Add the first laptop that appears in the search result to the basket

Then User basket should display with added item

Scenario: Navigate to a product and add the same to the User basket

Given User navigate for Lenovo Laptop

When Add the laptop to the basket

Then User basket should display with added item

***Step Definitions***

package stepDefinition;

import cucumber.api.java.en.Given;

import cucumber.api.java.en.Then;

import cucumber.api.java.en.When;

public class BackGround\_Steps {

@Given("^I navigate to the login page$")

public void i\_navigate\_to\_the\_login\_page() throws Throwable {

System.out.println("I am at the LogIn Page");

}

@When("^I submit username and password$")

public void i\_submit\_username\_and\_password() throws Throwable {

System.out.println("I Submit my Username and Password");

}

@Then("^I should be logged in$")

public void i\_should\_be\_logged\_in() throws Throwable {

System.out.println("I am logged on to the website");

}

@Given("^User search for Lenovo Laptop$")

public void user\_searched\_for\_Lenovo\_Laptop() throws Throwable {

System.out.println("User searched for Lenovo Laptop");

}

@When("^Add the first laptop that appears in the search result to the basket$")

public void add\_the\_first\_laptop\_that\_appears\_in\_the\_search\_result\_to\_the\_basket() throws Throwable {

System.out.println("First search result added to bag");

}

@Then("^User basket should display with added item$")

public void user\_basket\_should\_display\_with\_item() throws Throwable {

System.out.println("Bag is now contains the added product");

}

@Given("^User navigate for Lenovo Laptop$")

public void user\_navigate\_for\_Lenovo\_Laptop() throws Throwable {

System.out.println("User navigated for Lenovo Laptop");

}

@When("^Add the laptop to the basket$")

public void add\_the\_laptop\_to\_the\_basket() throws Throwable {

System.out.println("Laptop added to the basket");

}

}

***Hooks File***

package utilities;

import cucumber.api.java.After;

import cucumber.api.java.Before;

public class Hooks {

@Before(order=1)

public void beforeScenario(){

System.out.println("Start the browser and Clear the cookies");

}

@Before(order=0)

public void beforeScenarioStart(){

System.out.println("-----------------Start of Scenario-----------------");

}

@After(order=0)

public void afterScenarioFinish(){

System.out.println("-----------------End of Scenario-----------------");

}

@After(order=1)

public void afterScenario(){

System.out.println("Log out the user and close the browser");

}

}

***Output***

Feature: Test Background Feature

Description: The purpose of this feature is to test the Background keyword

-----------------Start of Scenario-----------------

Start the browser and Clear the cookies

I am at the LogIn Page

I Submit my Username and Password

I am logged on to the website

User searched for Lenovo Laptop

First search result added to bag

Bag is now contains the added product

Log out the user and close the browser

-----------------End of Scenario-----------------

-----------------Start of Scenario-----------------

Start the browser and Clear the cookies

I am at the LogIn Page

I Submit my Username and Password

I am logged on to the website

User navigated for Lenovo Laptop

Laptop added to the basket

Bag is now contains the added product

Log out the user and close the browser

-----------------End of Scenario-----------------

**What are Hooks in Cucumber?**

Cucumber supports hooks, which are blocks of code that run before or after each scenario. You can define them anywhere in your project or step definition layers, using the methods @Before and @After.

|  |
| --- |
| package utilities;  import cucumber.api.java.After;  import cucumber.api.java.Before;  public class Hooks {    @Before  public void beforeScenario(){  System.out.println("This will run before the Scenario");  }    @After  public void afterScenario(){  System.out.println("This will run after the Scenario");  }  } |

**End of Cucumber**

**Start of API points**

**Difference between PUT and PATCH**

PUT is a method of modifying resource where the client sends data that updates the entire resource .

If the data is present it modifies the record. If not present it creates a record

PATCH is a method of modifying resources where the client sends partial data that is to be updated without modifying the entire data.

Types of authorization;

OAuth1.0 and OAuth2.0

Bearer token

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

**HTTP defines these standard status codes that can be used to convey the results of a client’s request. The status codes are divided into five categories.**

1xx: Informational – Communicates transfer protocol-level information.

2xx: Success – Indicates that the client’s request was accepted successfully.

3xx: Redirection – Indicates that the client must take some additional action in order to complete their request.

4xx: Client Error – This category of error status codes points the finger at clients.

5xx: Server Error – The server takes responsibility for these error status codes.

**When ever a post request is made, and record is created what is the status code**

201 Created

Indicates that the request has succeeded and a new resource has been created as a result.

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

**Monolithic system and microservices arch**

* While a monolithic system acts as a single unit, in a microservice architecture, the business logic is separated into smaller self-sufficient services. Each of the services is supposed to be lightweight and have a single purpose. Also, each service ideally communicates to its database.

**What is api gateway**

An API gateway is an [API management](https://www.redhat.com/en/topics/api/what-is-api-management) tool that sits between a client and a collection of backend services.

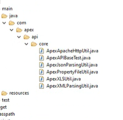
An API gateway acts as a reverse proxy to accept all [application programming interface (API)](https://www.redhat.com/en/topics/api/what-are-application-programming-interfaces) calls, aggregate the various services required to fulfill them, and return the appropriate result.

**Questions**

Difference between api and webservices

SOAP VS API

Explain the framework



**End API points**

**Different stages of jenkins**

**During the introduction of the project**

* Responsible for automation using selenium
* We are using git hub for string the project
* Maven for build testing and build management
* Jenkins for test case execution
* Jira for defect logging and tracking
* We are following agile process

Challenges faced in agile process

Requirements keep changing frequently and not much of documentation

How do you ensure you tested all the scenarios

Types of testing done by you

Functional testing

Integration testing

End to end testing

Regression testing

## 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. A Test Case contains test steps, test data, precondition, postcondition developed for specific test scenario to verify any requirement. The test case includes specific variables or conditions, using which a testing engineer can compare expected and actual results to determine whether a software product is functioning as per the requirements of the customer.

### [Test Scenario Vs Test Case](https://www.guru99.com/test-case-vs-test-scenario.html)

Test scenarios are rather vague and cover a wide range of possibilities. Testing is all about being very specific.

For a [Test Scenario](https://www.guru99.com/test-scenario.html): Check Login Functionality there many possible test cases are:

* Test Case 1: Check results on entering valid User Id & Password
* Test Case 2: Check results on entering Invalid User ID & Password
* Test Case 3: Check response when a User ID is Empty & Login Button is pressed, and many more

## The format of Standard Test Cases

Below is a format of a standard login Test cases example.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Test Case ID** | **Test Case Description** | **Test Steps** | **Test Data** | **Expected Results** | **Actual Results** | **Pass/Fail** |
| TU01 | Check Customer Login with valid Data | 1. Go to site http://demo.guru99.com 2. Enter UserId 3. Enter Password 4. Click Submit | Userid = guru99 Password = pass99 | User should Login into an application | As Expected | Pass |
| TU02 | Check Customer Login with invalid Data | 1. Go to site http://demo.guru99.com 2. Enter UserId 3. Enter Password 4. Click Submit | Userid = guru99 Password = glass99 | User should not Login into an application | As Expected | Pass |

This entire table may be created in Word, Excel or any other Test management tool. That’s all to Test Case Design

**Techniques followed while writing test cases**

Testcase design technique