**Java** is class based and object oriented **programming language**. It is a **platform independent language** i.e. the compiled code can be run on any **java** supporting **platform**. ... JVM recognizes the **platform** it is on and converts the bytecodes into native machine code. Hence **java** is called **platform independent language**.

.  


* **JDK** – **Java Development Kit** (in short JDK) is Kit which provides the environment to **develop and execute(run)** the Java program. JDK is a kit(or package) which includes two things
  + 1. Development Tools(to provide an environment to develop your java programs)
    2. JRE (to execute your java program).

**Note :**JDK is only used by Java Developers.

* **JRE** – **Java Runtime Environment** (to say JRE) is an installation package which provides environment to **only run(not develop)** the java program(or application)onto your machine. JRE is only used by them who only wants to run the Java Programs i.e. end users of your system.
* **JVM** – **Java Virtual machine**(JVM) is a very important part of both JDK and JRE because it is contained or inbuilt in both. Whatever Java program you run using JRE or JDK goes into JVM and JVM is responsible for **executing the java program line by line** hence it is also known as interpreter.

Static Keywork in Java

@SuppressWarnings("static-access")

What is the difference between instance variable and class variable?

**What is the difference between class variables and class instance variables**? The main **difference** is the behavior concerning inheritance: **class variables** are shared **between** a **class** and all its subclasses, while **class instance variables** only belong to one specific **class**.

Though both **float** and **double** datatype are used to represent **floating**-point numbers in **Java**, a **double** data type **is** more precise than **float**. A **double** variable can provide precision up to 15 to 16 decimal points as compared to **float** precision of 6 to 7 decimal digits.

No, we cannot declare a top-level **class** as **private** or protected. It **can** be either public or default (no modifier)

. A **static method** can NOT access a **Non**-**static method** or variable.

this can also be used to:

* Invoke current class constructor
* Invoke current class method
* Return the current class object
* Pass an argument in the method call
* Pass an argument in the constructor call

**Java compilers** are designed in such a way that converts source code into platform independent form i-e byte codes. These byte codes are then converted to machine code by **interpreter**. This is how **compiler and interpreter both** used in one language. ... It converts bytecode into machine language and executes line by line.

A **concrete class** can only have **concrete** methods. Even a single **abstract** method makes the **class abstract**. **Abstract class** can not be instantiated using new keyword. **Concrete class** can be instantiated using new keyword

A **concrete class** is a **class** that has an implementation for all of its methods. They cannot have any unimplemented methods. It can also extend an abstract **class** or implement an interface as long as it implements all their methods. It is a complete **class** and can be instantiated.

A **concrete class** is a class that has an implementation for all of its methods. They cannot have any unimplemented methods. It can also extend an [abstract class](https://www.geeksforgeeks.org/abstract-classes-in-java/) or implement an [interface](https://www.geeksforgeeks.org/interfaces-in-java/) as long as it implements all their methods. It is a complete class and can be instantiated.

In other words, we can say that any class which is not abstract is a concrete class.

**Necessary condition for a concrete class:** There must be an implementation for each and every method.

Interfaces in Java

* Difficulty Level : [Easy](https://www.geeksforgeeks.org/easy)
* Last Updated : 05 May, 2020

Like a class, an interface can have methods and variables, but the methods declared in an interface are by default abstract (only method signature, no body).

* Interfaces specify what a class must do and not how. It is the blueprint of the class.
* An Interface is about capabilities like a Player may be an interface and any class implementing Player must be able to (or must implement) move(). So it specifies a set of methods that the class has to implement.
* If a class implements an interface and does not provide method bodies for all functions specified in the interface, then the class must be declared abstract.
* A Java library example is, [Comparator Interface](https://www.geeksforgeeks.org/comparator-interface-java/). If a class implements this interface, then it can be used to sort a collection.

**Compile-Time Errors**: Errors that occur when you **violate the rules** of writing syntax are known as Compile-Time errors. This compiler error indicates something that must be fixed before the code can be compiled. All these errors are detected by the compiler and thus are known as compile-time errors.

**Run-Time Errors**: Errors which occur during program execution(run-time) after successful compilation are called run-time errors. One of the most common run-time error is division by zero also known as Division error. These types of error are hard to find as the compiler doesn’t point to the line at which the error occurs.

For more understanding run the example given below.

The **singleton** design **pattern** is used to restrict the instantiation of a **class** and ensures that only one instance of the **class** exists in the JVM. In other words, a **singleton class** is a **class** that can have only one object (an instance of the **class**) at a time per JVM instance.

**finalize method**

It is a **method** that the [Garbage Collector](https://www.geeksforgeeks.org/garbage-collection-java/) always calls just **before** the deletion/destroying the object which is eligible for Garbage Collection, so as to perform **clean-up activity**. Clean-up activity means closing the resources associated with that object like Database Connection, Network Connection or we can say resource de-allocation. Remember it is **not** a reserved keyword.

MicroServices

Microservices are small applications that your development teams create independently. Since they communicate via messaging if at all, they’re not dependent on the same coding language. Developers can use the programming language that they’re most familiar with. This helps them come work faster, with lower costs and fewer bugs

Since your teams are working on smaller applications and more focused problem domains, their projects tend to be more agile, too. They can iterate faster, address new features on a shorter schedule, and turn around bug fixes almost immediately. They often find more opportunities to reuse code, also.

Microservices improve your architecture’s scalability, too.

**ABSTRACTION**

In Page Object Model design pattern, we write locators (such as id, name, xpath etc.,) in a Page Class. We utilize these locators in tests but we can’t see these locators in the tests. Literally we hide the locators from the tests.

Abstraction is the methodology of hiding the implementation of internal details and showing the functionality to the users.

**INTERFACE**

Basic statement we all know in Selenium is WebDriver driver = new FirefoxDriver();

WebDriver itself is an Interface. So based on the above statement WebDriver driver = new FirefoxDriver(); we are initializing Firefox browser using Selenium WebDriver. It means we are creating a reference variable (driver) of the interface (WebDriver) and creating an Object. Here WebDriver is an Interface as mentioned earlier and FirefoxDriver is a class.

An interface in Java looks similar to a class but both the interface and class are two different concepts. An interface can have methods and variables just like the class but the methods declared in interface are by default abstract. We can achieve 100% abstraction and multiple inheritance in Java with Interface.

**INHERITANCE**

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

We extend the Base Class in other classes such as Tests and Utility Class. Extending one class into other class is known as Inheritance.

**POLYMORPHISM**

Combination of overloading and overriding is known as Polymorphism. We will see both overloading and overriding below.

Polymorphism allows us to perform a task in multiple ways.

### ****METHOD OVERLOADING****

We use implicit wait in Selenium. Implicit wait is an example of overloading. In Implicit wait we use different time stamps such as SECONDS, MINUTES, HOURS etc.,

A class having multiple methods with same name but different parameters is called Method Overloading

**METHOD OVERRIDING**

We use a method which was already implemented in another class by changing its parameters. To understand this you need to understand Overriding in Java.

Declaring a method in child class which is already present in the parent class is called Method Overriding. Examples are get and navigate methods of different drivers in Selenium .

**ENCAPSULATION**

All the classes in a framework are an example of Encapsulation. In POM classes, we declare the data members using @FindBy and initialization of data members will be done using Constructor to utilize those in methods.

Encapsulation is a mechanism of binding code and data together in a single unit.

**WEB ELEMENT:**

Web element is an interface used to identify the elements in a web page.

**WEBDRIVER:**

WebDriver is an interface used to launch different browsers such as Firefox, Chrome, Internet Explorer, Safari etc.,

**FIND BY:**

FindBy is an annotation used in Page Object Model design pattern to identify the elements.

**FIND ELEMENT:**

Find Element is a method in POM to identify the elements in a web page.

**String Methods**

1. **int length():**Returns the number of characters in the String.

"GeeksforGeeks".length();  // returns 13

1. [Char charAt(int i)](https://www.geeksforgeeks.org/java-string-charat-method-example/)**:**Returns the character at ith index.

"GeeksforGeeks".charAt(3); // returns  ‘k’

1. [String substring (int i)](https://www.geeksforgeeks.org/substring-in-java/)**:**Return the substring from the ith index character to end.

"GeeksforGeeks".substring(3); // returns “ksforGeeks”

1. [String substring (int i, int j)](https://www.geeksforgeeks.org/substring-in-java/)**:**Returns the substring from i to j-1 index.

"GeeksforGeeks".substring(2, 5); // returns “eks”

1. [String concat( String str)](https://www.geeksforgeeks.org/java-string-concat-examples/)**:**Concatenates specified string to the end of this string.
2. String s1 = ”Geeks”;
3. String s2 = ”forGeeks”;
4. String output = s1.concat(s2); // returns “GeeksforGeeks”
5. [int indexOf (String s)](https://www.geeksforgeeks.org/java-string-indexof/)**:**Returns the index within the string of the first occurrence of the specified string.
6. String s = ”Learn Share Learn”;
7. int output = s.indexOf(“Share”); // returns 6
8. [int indexOf (String s, int i)](https://www.geeksforgeeks.org/java-string-indexof/)**:**Returns the index within the string of the first occurrence of the specified string, starting at the specified index.
9. String s = ”Learn Share Learn”;
10. int output = s.indexOf("ea",3);// returns 13
11. [Int lastIndexOf( String s)](https://www.geeksforgeeks.org/java-lang-string-lastindexof-method/)**:**Returns the index within the string of the last occurrence of the specified string.
12. String s = ”Learn Share Learn”;
13. int output = s.lastIndexOf("a"); // returns 14
14. **boolean equals( Object otherObj):**Compares this string to the specified object.
15. Boolean out = “Geeks”.equals(“Geeks”); // returns true
16. Boolean out = “Geeks”.equals(“geeks”); // returns false
17. [boolean  equalsIgnoreCase (String anotherString)](https://www.geeksforgeeks.org/equalsignorecase-in-java/)**:**Compares string to another string, ignoring case considerations.
18. Boolean out= “Geeks”.equalsIgnoreCase(“Geeks”); // returns true

Boolean out = “Geeks”.equalsIgnoreCase(“geeks”); // returns true

1. [int compareTo( String anotherString)](https://www.geeksforgeeks.org/java-lang-string-compareto/)**:**Compares two string lexicographically.
2. int out = s1.compareTo(s2);  // where s1 ans s2 are
3. // strings to be compared
4. This returns difference s1-s2. If :
5. out < 0 // s1 comes before s2
6. out = 0 // s1 and s2 are equal.
7. out > 0 // s1 comes after s2.
8. **int compareToIgnoreCase( String anotherString):**Compares two string lexicographically, ignoring case considerations.
9. int out = s1.compareToIgnoreCase(s2);
10. // where s1 ans s2 are
11. // strings to be compared
12. This returns difference s1-s2. If :
13. out < 0 // s1 comes before s2
14. out = 0 // s1 and s2 are equal.
15. out > 0 // s1 comes after s2.

Note- In this case, it will not consider case of a letter (it will ignore whether it is uppercase or lowercase).

1. [String toLowerCase()](https://www.geeksforgeeks.org/java-string-tolowercase-examples/)**:**Converts all the characters in the String to lower case.
2. String word1 = “HeLLo”;
3. String word3 = word1.toLowerCase(); // returns “hello"
4. [String toUpperCase()](https://www.geeksforgeeks.org/java-touppercase-examples/)**:**Converts all the characters in the String to upper case.
5. String word1 = “HeLLo”;
6. String word2 = word1.toUpperCase(); // returns “HELLO”
7. [String trim()](https://www.geeksforgeeks.org/java-string-trim-method-example/)**:**Returns the copy of the String, by removing whitespaces at both ends. It does not affect whitespaces in the middle.
8. String word1 = “ Learn Share Learn “;
9. String word2 = word1.trim(); // returns “Learn Share Learn”
10. [String replace (char oldChar, char newChar)](https://www.geeksforgeeks.org/java-lang-string-replace-method-java/)**:**Returns new string by replacing all occurrences of oldChar with newChar.
11. String s1 = “feeksforfeeks“;

String s2 = “feeksforfeeks”.replace(‘f’ ,’g’); // returns “geeksgorgeek

The **ArrayList** uses an array for storing the data. ... So essentially searching in array again and again will have O(n^2) complexity. While **HashSet** uses hashing mechanism for storing the elements into their respective buckets. The operation of **HashSet** will be **faster** for long list of values.

**Hashing** is the **mechanism** of assigning unique code to a variable or attribute using an **algorithm** to enable easy retrieval. A true **hashing mechanism** should always return the same hashCode() when it is applied to the same object. \* The table, resized as necessary.

GIT Commands