# Abstract Class, Interfaces

**Abstraction** is a process of hiding the implementation details and showing only functionality to the user.

Another way, it shows only essential things to the user and hides the internal details, for example, sending SMS where you type the text and send the message. You don't know the internal processing about the message delivery.

Abstraction lets you focus on what the object does instead of how it does it.

## Ways to achieve Abstraction

There are two ways to achieve abstraction in java

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

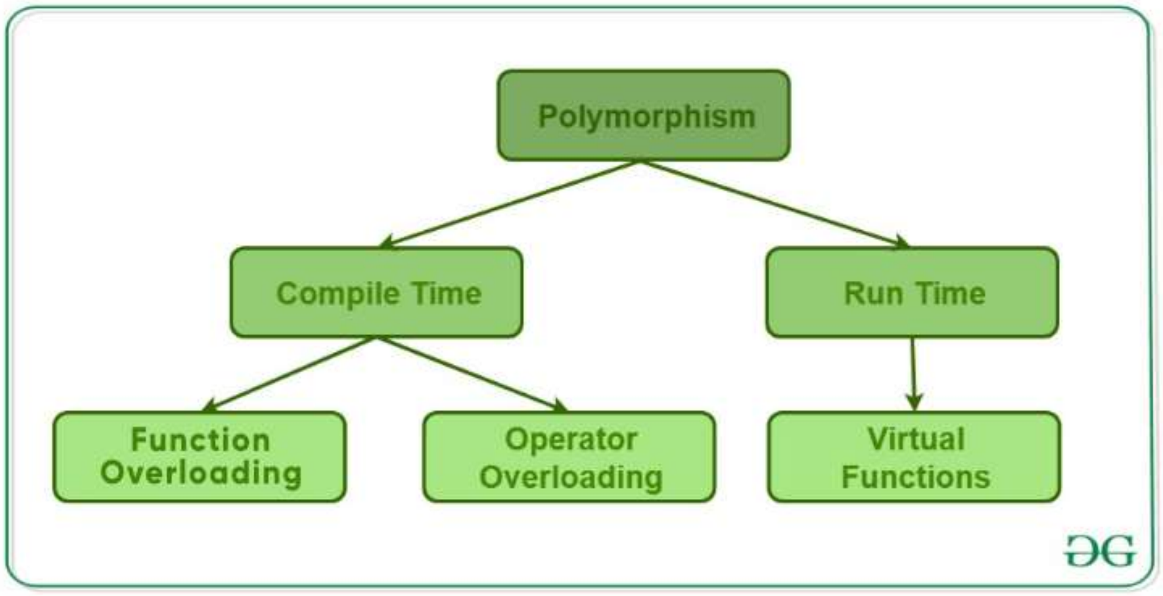
| Abstract class | Interface |
|---|---|
| 1) Abstract class can **have abstract and non-abstract** methods. | Interface can have **only abstract** methods. Since Java 8, it can have **default and static methods** also. |
| 2) Abstract class **doesn't support multiple inheritance**. | Interface **supports multiple inheritance**. |
| 3) Abstract class **can have final, non-final, static and non-static variables**. | Interface has **only static and final variables**. |
| 4) Abstract class **can provide the implementation of interface**. | Interface **can't provide the implementation of abstract class**. |
| 5) The **abstract keyword** is used to declare abstract class. | The **interface keyword** is used to declare interface. |

| | |
|---|---|
| 6) An **abstract class** can extend another Java class and implement multiple Java interfaces. | An **interface** can extend another Java interface only. |
| 7) An **abstract class** can be extended using keyword "extends". | An **interface** can be implemented using keyword "implements". |
| 8) A Java **abstract class** can have class members like private, protected, etc. | Members of a Java interface are public by default. |
| 9)**Example:**<br>public abstract class Shape{<br>public abstract void draw();<br>} | **Example:**<br>public interface Drawable{<br>void draw();<br>} |

https://www.javatpoint.com/difference-between-abstract-class-and-interface

Polymorphism and its types

Polymorphism is defined as the process of using a function or an operator for more than one purpose. In other words, we can also say that an operator or a function can serve us in different ways.



## Compile-time polymorphism

It is defined as the polymorphism in which the function is called at the compile time. This type of process is also called as *early or static binding*.

### Examples of compile-time polymorphism

**1)**Function overloading is defined as using one function for different purposes. Here, one function performs many tasks by changing the function signature(number of arguments and types of arguments). It is an example of compile-time polymorphism because what function is to be called is decided at the time of compilation.

**2)**Operator overloading is defined as using an operator for an addition operation besides the original one.

The concept of operator overloading is used as it provides special meanings to the user-defined data types. The benefit of operator overloading is we can use the same operand to serve two different operations. The basic operator overloading example is the '+' operator as it is used to add numbers and strings.

The operators **, :: ?: sizeof** cant be overloaded.

## Run-time polymorphism

The run-time polymorphism is defined as the process in which the function is called at the time of program execution.

An example of runtime polymorphism is *function overriding*.

### Function overriding

When we say a function is overridden it means that a new definition of that function is given in the derived class. Thus, in function overriding, we have two definitions of one in the base class and the other in the derived class. The decision of choosing a function is decided at the run time.

#### Virtual Function

The behavior of a virtual function can be overridden with the inheriting class function with the same name. It is basically defined in the base class and overridden in the inherited class.

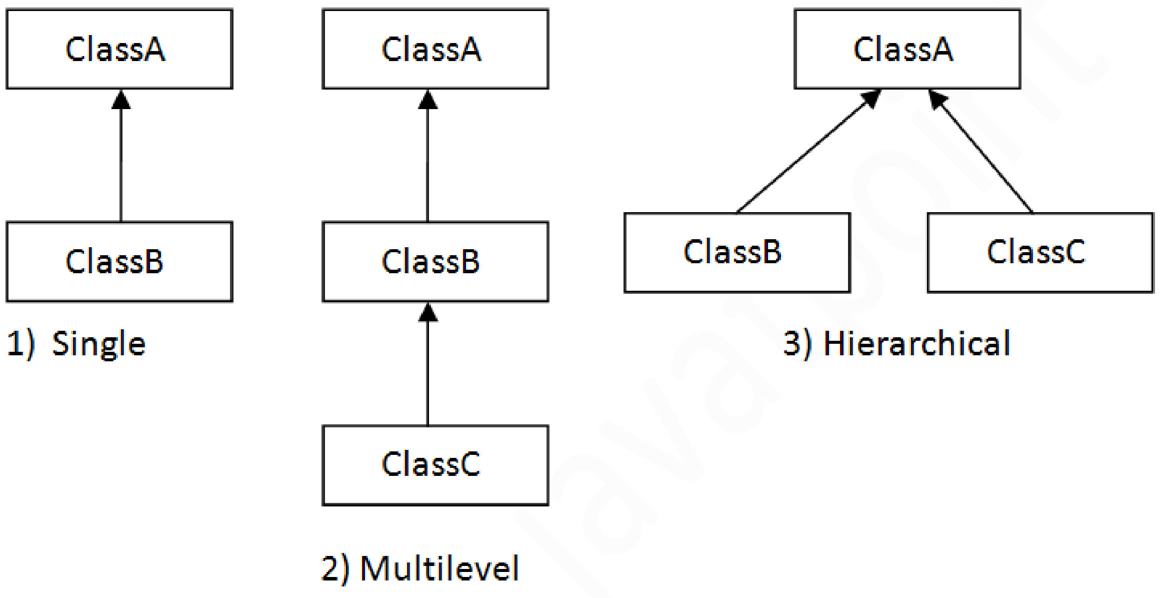
# Inheritance and its types

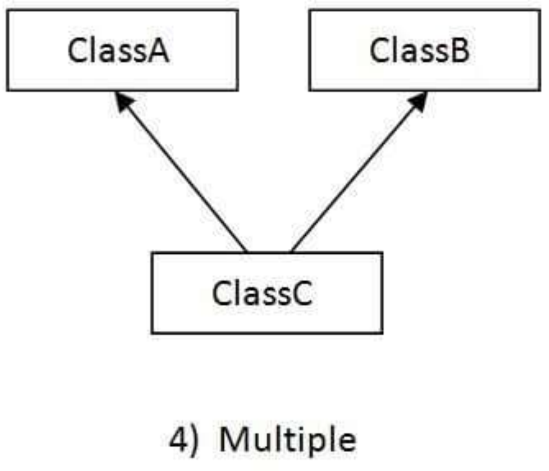
**Inheritance in Java** is a mechanism in which one object acquires all the properties and behaviors of a parent object. It is an important part of OOPs (Object Oriented programming system).

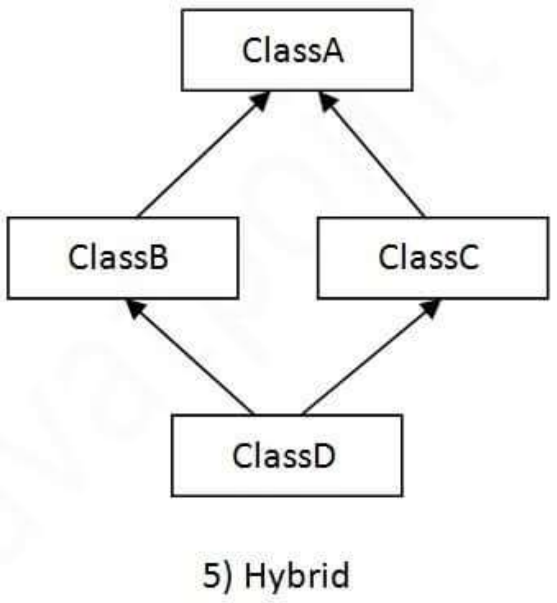
The idea behind inheritance in Java is that you can create new classes that are built upon existing classes. When you inherit from an existing class, you can reuse methods and fields of the parent class. Moreover, you can add new methods and fields in your current class also.

## Why use inheritance in java

- For Method Overriding (so runtime polymorphism can be achieved).
- For Code Reusability.

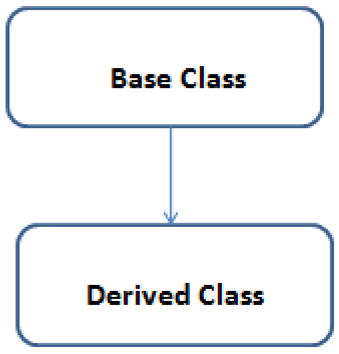






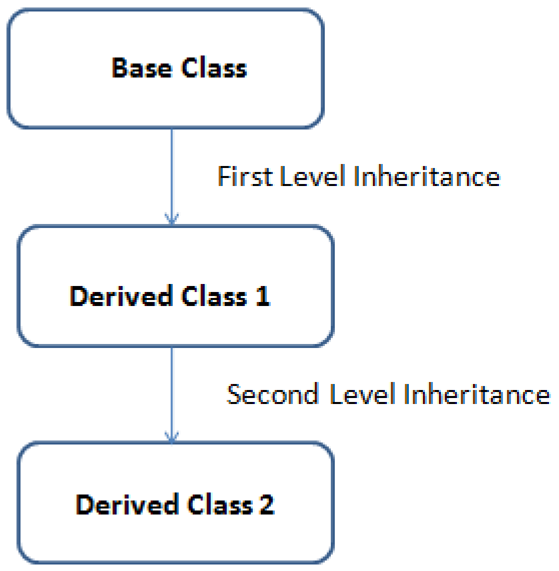
.

## Single Inheritance



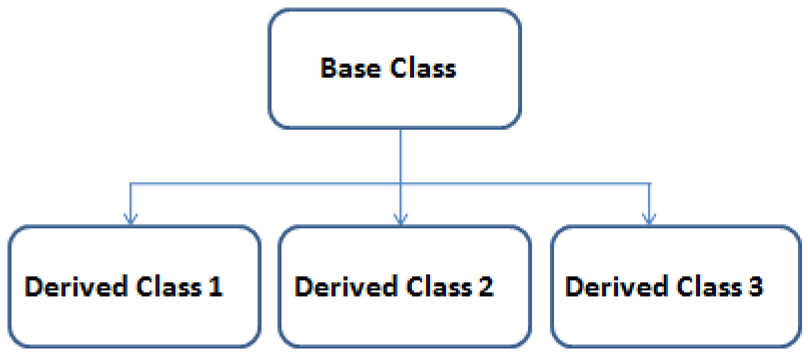
*When a **Derived Class** to inherit properties and behavior from a single **Base Class** , it is called as single inheritance.*

## Multi Level Inheritance



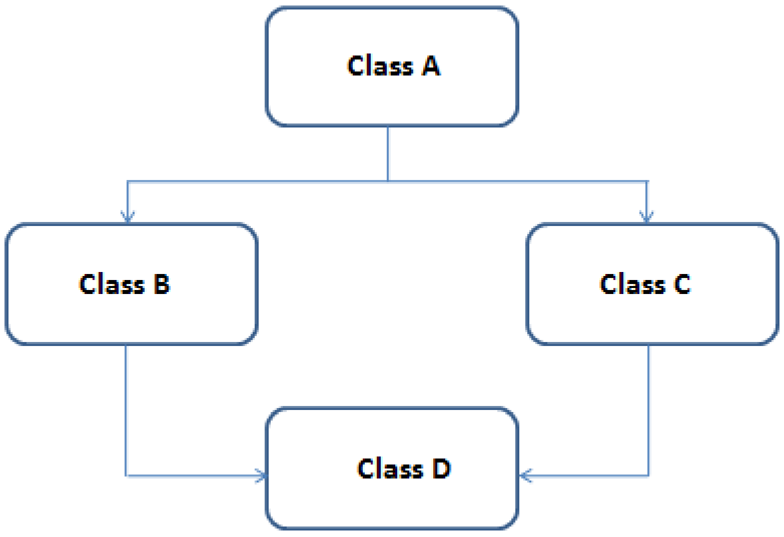
*A **derived class** is created from another derived class is called **Multi Level Inheritance** .*

## Hierarchical Inheritance



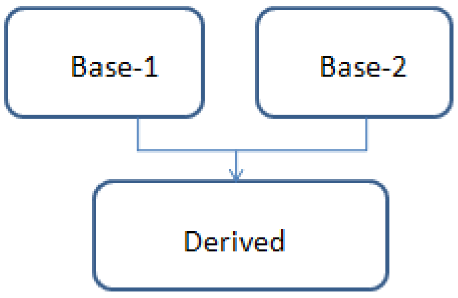
*More than one **derived classes** are created from a single base class, is called **Hierarchical Inheritance** .*

## Hybrid Inheritance(through interface not class)



*Any combination of above **three inheritance** (single, hierarchical and multi level) is called as **hybrid inheritance** .*

## Multiple Inheritance(through interface not class)



*Multiple inheritances allows programmers to create classes that combine aspects of multiple classes and their corresponding hierarchies. In .Net Framework, the classes are only allowed to inherit from a single parent class, which is called single inheritance.*

*https://www.geeksforgeeks.org/interfaces-and-inheritance-in-java/*

# Access Modifiers

The access modifiers in Java specifies the accessibility or scope of a field, method, constructor, or class. We can change the access level of fields, constructors, methods, and class by applying the access modifier on it.

There are four types of Java access modifiers:

1. **Private**: The access level of a private modifier is only within the class. It cannot be accessed from outside the class.
2. **Default**: The access level of a default modifier is only within the package. It cannot be accessed from outside the package. If you do not specify any access level, it will be the default.
3. **Protected**: The access level of a protected modifier is within the package and outside the package through child class. If you do not make the child class, it cannot be accessed from outside the package.
4. **Public**: The access level of a public modifier is everywhere. It can be accessed from within the class, outside the class, within the package and outside the package.
5. Let's understand the access modifiers in Java by a simple table.

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

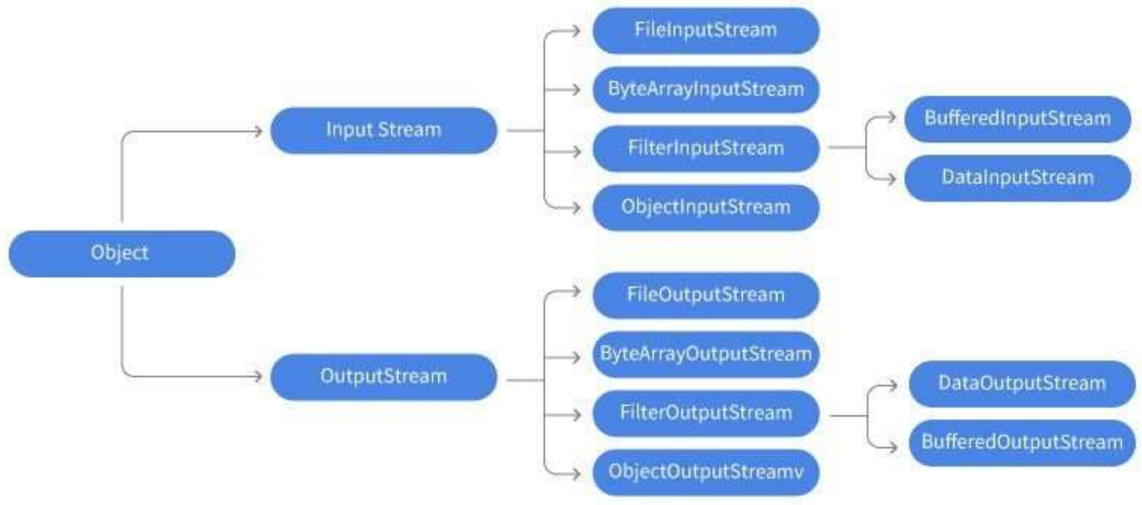
# File I/o hierarchy

To process the input and generate an output, Java I/O (Input and Output) is used.

Java utilizes the stream idea to speed up the I/O operation. The java.io package includes all the classes which are necessary for input and output operations.

We can use Java I/O API to handle files.

The java.io package contains almost every class in Java that you may ever need to perform input and output (I / O). All these streams are a source of input and a destination of output. The stream in the java.io package supports a lot of information like primitives, object, localized characters, etc.



## FileInputStream FileOutputStream , BufferedReader BufferedWriter , ByteArrayInput stream ByteArrayOutput stream, File Reader, File Writer, CharacterArray Reader, Character Array Writer

## FileInputStream

This class reads the data from a specific file (byte by byte). It is usually used to read the contents of a file with raw bytes, such as images.

To read the contents of a file using this class –

- First of all, you need to instantiate this class by passing a String variable or a File object, representing the path of the file to be read.

```
FileInputStream inputStream = new FileInputStream("file_path");
or,
File file = new File("file_path");
FileInputStream inputStream = new FileInputStream(file);
```

- Then read the contents of the specified file using either of the variants of read() method –
  - **int read()** – This simply reads data from the current InputStream and returns the read data byte by byte (in integer format).
    This method returns -1 if the end of the file is reached.
  - **int read(byte[] b)** – This method accepts a byte array as parameter and reads the contents of the current InputStream, to the given array
    This method returns an integer representing the total number of bytes or, -1 if the end of the file is reached.
  - **int read(byte[] b, int off, int len)** – This method accepts a byte array, its offset (int) and, its length (int) as parameters and reads the contents of the current InputStream, to the given array.
  - This method returns an integer representing the total number of bytes or, -1 if the end of the file is reached.

## FileOutputStream

This writes data into a specific file or, file descriptor (byte by byte). It is usually used to write the contents of a file with raw bytes, such as images.

To write the contents of a file using this class –

- First of all, you need to instantiate this class by passing a String variable or a **File** object, representing the path of the file to be read.

```
FileOutputStream outputStream = new FileOutputStream("file_path");
or,
File file = new File("file_path");
FileOutputStream outputStream = new FileOutputStream (file);
```

You can also instantiate a FileOutputStream class by passing a FileDescriptor object.

```
FileDescriptor descriptor = new FileDescriptor();
FileOutputStream outputStream = new FileOutputStream(descriptor);
```

- Then write the data to a specified file using either of the variants of write() method –
    - **int write(int b)** – This method accepts a single byte and writes it to the current OutputStream.
    - **int write(byte[] b)** – This method accepts a byte array as parameter and writes data from it to the current OutputStream.
    - **int write(byte[] b, int off, int len)** – This method accepts a byte array, its offset (int) and, its length (int) as parameters and writes its contents to the current OutputStream.

--→A buffer is a collective memory. Reader and Writer classes in java supports ***"Text Streaming"***. The ***"BufferedWriter"*** class of java supports writing a chain of characters output stream (Text based) in an efficient way. The Chain-Of-Characters can be Arrays, Strings, etc. The ***"BufferedReader"*** class is used to read a stream of text from a character-based input stream.

--→The ByteArrayInputStream is composed of two words: ByteArray and InputStream. As the name suggests, it can be used to read byte array as input stream.

Java ByteArrayInputStream class contains an internal buffer which is used to **read byte array** as stream. In this stream, the data is read from a byte array.

The buffer of ByteArrayInputStream automatically grows according to data.

Let's see the declaration for Java.io.ByteArrayInputStream class:

1. **public class** ByteArrayInputStream **extends** InputStream

--→Java ByteArrayOutputStream class is used to **write common data** into multiple files. In this stream, the data is written into a byte array which can be written to multiple streams later.

The ByteArrayOutputStream holds a copy of data and forwards it to multiple streams.

The buffer of ByteArrayOutputStream automatically grows according to data.

---

Let's see the declaration for Java.io.ByteArrayOutputStream class:

1. **public class** ByteArrayOutputStream **extends** OutputStream

Java FileWriter and FileReader classes are used to write and read data from text files (they are Character Stream classes). It is recommended **not** to use the FileInputStream and FileOutputStream classes if you have to read and write any textual information as these are Byte stream classes.

**FileWriter**
FileWriter is useful to create a file writing characters into it.

- This class inherits from the OutputStream class.
- The constructors of this class assume that the default character encoding and the default byte-buffer size are acceptable. To specify these values yourself, construct an OutputStreamWriter on a FileOutputStream.
- FileWriter is meant for writing streams of characters. For writing streams of raw bytes, consider using a FileOutputStream.
- FileWriter creates the output file if it is not present already.

**FileReader**
FileReader is useful to read data in the form of characters from a 'text' file.

- This class inherited from the InputStreamReader Class.
- The constructors of this class assume that the default character encoding and the default byte-buffer size are appropriate. To specify these values yourself, construct an InputStreamReader on a FileInputStream.
- FileReader is meant for reading streams of characters. For reading streams of raw bytes, consider using a FileInputStream.

--→The CharArrayReader is composed of two words: CharArray and Reader. The CharArrayReader class is used to read character array as a reader (stream). It inherits Reader class.

## Java CharArrayReader class declaration

Let's see the declaration for Java.io.CharArrayReader class:

```
1. public class CharArrayReader extends Reader
```

--→The CharArrayWriter class can be used to write common data to multiple files. This class inherits Writer class. Its buffer automatically grows when data is written in this stream. Calling the close() method on this object has no effect.

## Java CharArrayWriter class declaration

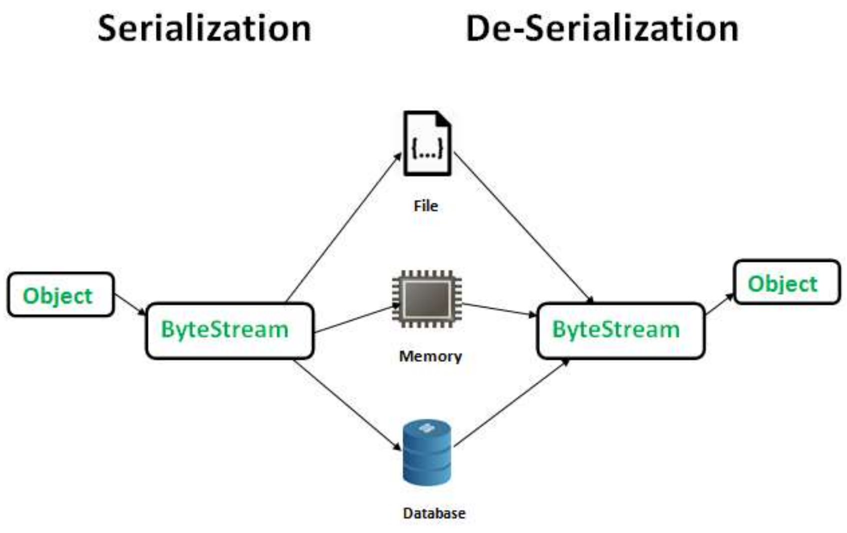
Let's see the declaration for Java.io.CharArrayWriter class:

1. **public class** CharArrayWriter **extends** Writer

# Serialization deserialization

Serialization is a mechanism of converting the state of an object into a byte stream. Deserialization is the reverse process where the byte stream is used to recreate the actual Java object in memory. This mechanism is used to persist the object.

**Advantages of Serialization**
1. To save/persist state of an object.
2. To travel an object across a network.



## Wrapper classes, Boxing, Unboxing

The **wrapper class in Java** provides the mechanism *to convert primitive into object and object into primitive*.

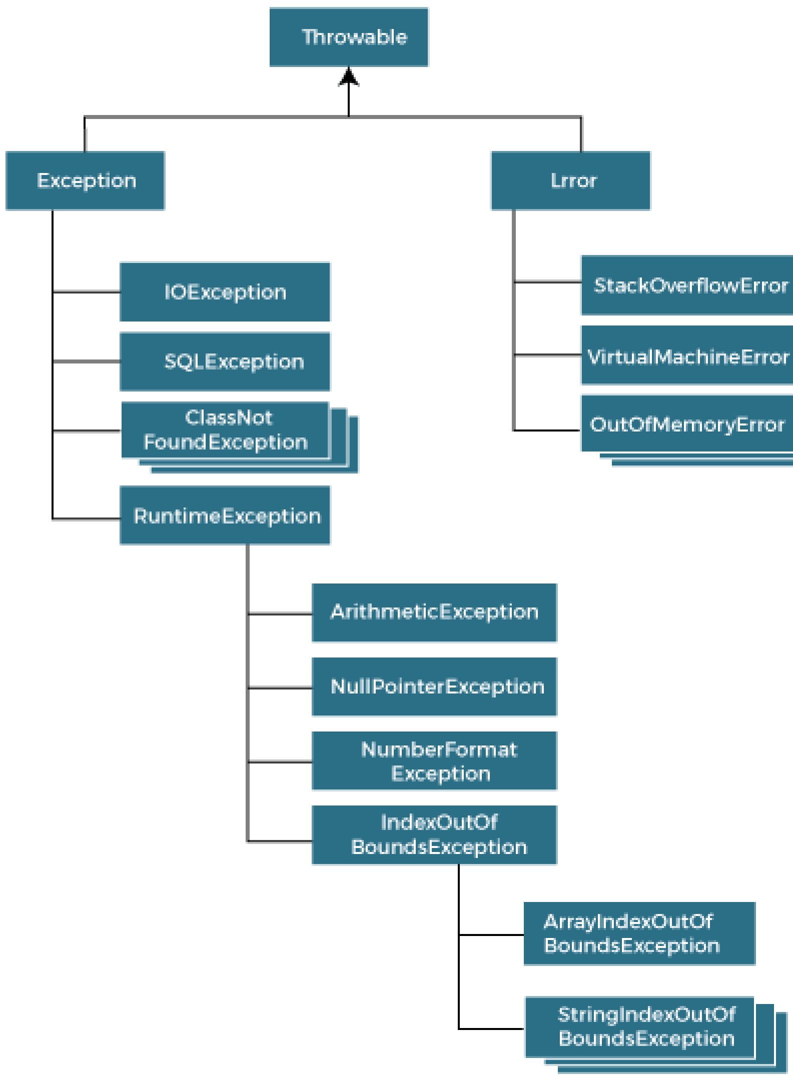
Since J2SE 5.0, **autoboxing** and **unboxing** feature convert primitives into objects and objects into primitives automatically. The automatic conversion of primitive into an object is known as autoboxing and vice-versa unboxing.

Java is an object-oriented programming language, so we need to deal with objects many times like in Collections, Serialization, Synchronization, etc. Let us see the different scenarios, where we need to use the wrapper classes.

- **Change the value in Method:** Java supports only call by value. So, if we pass a primitive value, it will not change the original value. But, if we convert the primitive value in an object, it will change the original value.
- **Serialization:** We need to convert the objects into streams to perform the serialization. If we have a primitive value, we can convert it in objects through the wrapper classes.
- **Synchronization:** Java synchronization works with objects in Multithreading.
- **java.util package:** The java.util package provides the utility classes to deal with objects.
- **Collection Framework:** Java collection framework works with objects only. All classes of the collection framework (ArrayList, LinkedList, Vector, HashSet, LinkedHashSet, TreeSet, PriorityQueue, ArrayDeque, etc.) deal with objects only.

## Exception hierarchy , All keywords used in exception handling. Unchecked Exceptions Checked Exceptions

The hierarchy of Exceptions in the Java programming language begins with the Throwable class – which comes from the Object class and is its direct subclasswhileThe Exception class presents all This Throwable class further branches into two subclasses – Error and Exception.



Exception Handling is a mechanism to handle runtime errors such as ClassNotFoundException, IOException, SQLException, RemoteException, etc. The core advantage of exception handling is **to maintain the normal flow of the application**. An exception normally disrupts the normal flow of the application; that is why we need to handle exceptions.

| Keyword | Description |
|---|---|
| try | The "try" keyword is used to specify a block where we should place an exception code. It means we can't use try block alone. The try block must be followed by either catch or finally. |
| catch | The "catch" 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 | The "finally" block is used to execute the necessary code of the program. It is executed whether an exception is handled or not. |
| throw | The "throw" keyword is used to throw an exception. |
| throws | The "throws" keyword is used to declare exceptions. It specifies that there may occur an exception in the method. It doesn't throw an exception. It is always used with method signature. |

### 1) Checked Exception

The classes that directly inherit the Throwable class except RuntimeException and Error are known as checked exceptions. For example, IOException, SQLException, etc. Checked exceptions are checked at compile-time.

### 2) Unchecked Exception

The classes that inherit the RuntimeException are known as unchecked exceptions. For example, ArithmeticException, NullPointerException, ArrayIndexOutOfBoundsException, etc. Unchecked exceptions are not checked at compile-time, but they are checked at runtime.

### 3) Error

Error is irrecoverable. Some example of errors are OutOfMemoryError, VirtualMachineError, AssertionError etc.

# Thread , thread synchronization

Threads allows a program to operate more efficiently by doing multiple things at the same time.

Threads can be used to perform complicated tasks in the background without interrupting the main program.

There are two ways to create a thread.

It can be created by extending the `Thread` class and overriding its `run()` method:

## Extend Syntax

```
public class Main extends Thread {
  public void run() {
    System.out.println("This code is running in a thread");
  }
}
```

Another way to create a thread is to implement the `Runnable` interface:

## Implement Syntax

```
public class Main implements Runnable {
  public void run() {
    System.out.println("This code is running in a thread");
  }
}
```

Synchronization in Java is the capability to control the access of multiple threads to any shared resource.

Java Synchronization is better option where we want to allow only one thread to access the shared resource.

## Why use Synchronization?

The synchronization is mainly used to

1. To prevent thread interference.
2. To prevent consistency problem.

## Types of Synchronization

There are two types of synchronization

1. Process Synchronization
2. Thread Synchronization

Here, we will discuss only thread synchronization.

## Thread Synchronization

There are two types of thread synchronization mutual exclusive and inter-thread communication.

1. Mutual Exclusive
    1. Synchronized method.
    2. Synchronized block.
    3. Static synchronization.
2. Cooperation (Inter-thread communication in java)

NOTE IMPORTANT:-

**Sleep():** This Method is used to pause the execution of current thread for a specified time in Milliseconds. Here, Thread does not lose its ownership of the monitor and resume's it's execution
**Wait():** This method is defined in object class. It tells the calling thread (a.k.a Current Thread) to wait until another thread invoke's the notify() or notifyAll() method for this object, The thread waits until it reobtains the ownership of the monitor and Resume's Execution.

| | |
|---|---|
| Wait() is not a static method. | Sleep() is a static method. |

# ArrayList , linked List , HashSet , HashMap and their methods.

| ArrayList | LinkedList |
|---|---|
| 1) ArrayList internally uses a **dynamic array** to store the elements. | LinkedList internally uses a **doubly linked list** to store the elements. |
| 2) Manipulation with ArrayList is **slow** because it internally uses an array. If any element is removed from the array, all the other elements are shifted in memory. | Manipulation with LinkedList is **faster** than ArrayList because it uses a doubly linked list, so no bit shifting is required in memory. |
| 3) An ArrayList class can **act as a list** only because it implements List only. | LinkedList class can **act as a list and queue** both because it implements List and Deque interfaces. |
| 4) ArrayList is **better for storing and accessing** data. | LinkedList is **better for manipulating** data. |
| 5) The memory location for the elements of an ArrayList is contiguous. | The location for the elements of a linked list is not contagious. |
| 6) Generally, when an ArrayList is initialized, a default capacity of 10 is assigned to the ArrayList. | There is no case of default capacity in a LinkedList. In LinkedList, an empty list is created when a LinkedList is initialized. |
| 7) To be precise, an ArrayList is a resizable array. | LinkedList implements the doubly linked list of the list interface. |

```
import java.util.*;
class TestArrayLinked{
 public static void main(String args[]){

  List<String> al=new ArrayList<String>();//creating arraylist
  al.add("Ravi");//adding object in arraylist
```

```java
al.add("Vijay");
al.add("Ravi");
al.add("Ajay");

List<String> al2=new LinkedList<String>();//creating linkedlist
al2.add("James");//adding object in linkedlist
al2.add("Serena");
al2.add("Swati");
al2.add("Junaid");

System.out.println("arraylist: "+al);
System.out.println("linkedlist: "+al2);
}
}
```
**Output:**

```
arraylist: [Ravi,Vijay,Ravi,Ajay]
linkedlist: [James,Serena,Swati,Junaid]
```

HashMap and HashSet:

| **Basis** | **HashMap** | **HashSet** |
|---|---|---|
| **Definition** | Java HashMap is a hash table based implementation of Map interface. | HashSet is a Set. It creates a collection that uses a hash table for storage. |
| **Implementation** | HashMap implements **Map, Cloneable,** and **Serializable** interface es. | HashSet implements **Set, Cloneable, Serializable, Iterable** and **Collection** interfaces. |
| **Stores** | In HashMap we store a **key-value pair**. It maintains the mapping of key and value. | In HashSet, we store **objects**. |

| | | |
|---|---|---|
| **Duplicate values** | It does not allow **duplicate keys**, but **duplicate values** are **allowed**. | It does not allow **duplicate values**. |
| **Null values** | It can contain a **single null key** and **multiple null values**. | It can contain **a single null value**. |
| **Method of insertion** | HashMap uses the **put()** method to add the elements in the HashMap. | HashSet uses the **add()** method to add elements in the HashSet. |
| **Performance** | HashMap is **faster/ than HashSet because values are associated with a unique key.** | HashSet is **slower** than HashMap because the member object is used for calculating hashcode value, which can be same for two objects. |
| **The Number of objects** | Only **one** object is created during the add operation. | There are **two** objects created during put operation, one for **key** and one for **value**. |
| **Storing Mechanism** | HashMap internally uses **hashing** to store objects. | HashSet internally uses a **HashMap** object to store objects. |
| **Uses** | Always prefer when we do not maintain the **uniqueness**. | It is used when we need to maintain the **uniqueness** of data. |
| **Example** | **{a->4, b->9, c->5}** Where **a, b, c** are **keys** and **4, 9, 5** are **values** associated with key. | **{6, 43, 2, 90, 4}** It denotes a set. |

```
1. HashMap<String, Integer> hm= new HashMap<String, Integer>();
2. HashSet<String> hs= new HashSet<String>();
```

JDBC connectivity steps or API components, JDBC Drivers

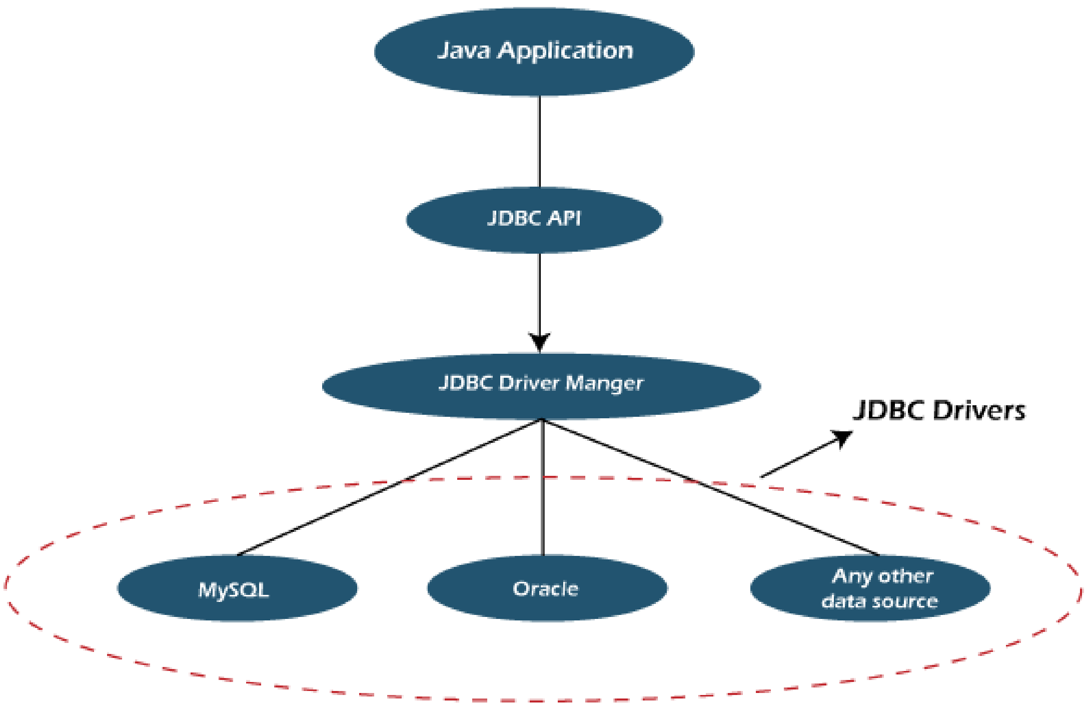
- Import the Packages.

- Load the drivers using the forName() method.
- Register the drivers using DriverManager.
- Establish a connection using the Connection class object.
- Create a statement.
- Execute the query.
- Close the connections.

**The JDBC API:** It allows the Java programs to perform the execution of the SQL statements and then get the results.

A few of the crucial interfaces and classes defined in the JDBC API are the following:

- Drivers
- DriverManager
- Statement
- Connection
- CallableStatement
- PreparedStatement
- ResultSet
- SQL data



-

--→JDBC DRIVERS :- https://www.javatpoint.com/jdbc-driver

# Junit annotations , Assert Functions , Fixtures

The JUnit annotations are predefined texts provided by Java API, which helps JVM to recognize the type of method or class for testing. In other words, they are used to specify or declare the methods or classes with few properties like testing, disabling tests, ignoring tests, etc. We need to install the JUnit dependencies first to use JUnit annotations for testing. You can use Maven to install these dependencies. Check out this article for more details about installing Maven dependencies.

| Annotations | Description |
|---|---|
| @Test | Represents the method or class as a test block, also accepts parameters. |
| @Before | The method with this annotation gets executed before all the other tests. |
| @After | The method with this annotation gets executed after all the other tests are executed. |
| @Ignore | It is used to ignore a few test statements during execution. |
| @Disabled | Used to disable the tests from execution, but the corresponding reports of the tests are still generated. |

**Assert** is a method useful in determining Pass or Fail status of a test case, The assert methods are provided by the class org.junit.Assert which extends java.lang.Object class.

There are various types of assertions like Boolean, Null, Identical etc.

## JUnit Assert methods

### Boolean

If you want to test the boolean conditions (true or false), you can use following assert methods

1. assertTrue(condition)
2. assertFalse(condition)

Here the condition is a boolean value.

### Null object

If you want to check the initial value of an object/variable, you have the following methods:

1. assertNull(object)
2. assertNotNull(object)

Here object is Java object e.g. assertNull(actual);

## Identical

If you want to check whether the objects are identical (i.e. comparing two references to the same java object), or different.

1. assertSame(expected, actual), It will return true if expected == actual
2. assertNotSame(expected, actual)

## Assert Equals

If you want to test equality of two objects, you have the following methods

- assertEquals(expected, actual)

It will return true if: expected.equals( actual ) returns true.

## Assert Array Equals

If you want to test equality of arrays, you have the following methods as given below:

- assertArrayEquals(expected, actual)

Above method must be used if arrays have the same length, for each valid value for i, you can check it as given below:

- assertEquals(expected[i],actual[i])
- assertArrayEquals(expected[i],actual[i])

## Fail Message

If you want to throw any assertion error, you have fail() that always results in a fail verdict.

- Fail(message);

**JUnit** is an open source unit testing tool and used to test small/large units of code. To run the JUnit test you don’t have to create a class object or define the main method. Junit provides assertion library which is used to evaluate the test result. Annotations of JUnit are used to run the test method. JUnit is also used to run the Automation suite having multiple test cases.

It is used by developers to implement unit testing in Java, and accelerate programming speed and increase the quality of code. JUnit test framework provides the following important features:

- **Fixtures:** Fixed state of a set of objects used as a baseline for running tests. The purpose of a test fixture is to ensure that there is a well-known and fixed environment in which tests are run so that results are repeatable.
- Test suites: Bundles a few unit test cases and runs them together. Both @RunWith and @Suite annotation are used to run the suite test.
- Test runners: Test runner is used for executing the test cases.
- JUnit classes: JUnit classes are important classes, used in writing and testing JUnits. Some of the important classes are Assert, TestCase, TestResult.

## CGI, web container

**The Common Gateway Interface (CGI)** is a standard for writing programs that can interact through a Web server with a client running a Web browser. These programs allow a Web developer to deliver dynamic information (usually in the form of HTML) via the browser. A CGI program can be written in any language, including Java, that can be executed by your Web server. CGI programs are commonly used to add search engines, guest-book applications, database-query engines, interactive-user forums, and other interactive applications to Web sites.

In very basic terms, a CGI program must interpret the information sent to it, process the information in some way, and generate a response that will be sent back to the client.

| Basis | Servlet | CGI |
|---|---|---|
| Approach | It is thread based i.e. for every new request new thread is created. | It is process-based i.e. for every new request new process is created. |
| Language Used | The codes are written in JAVA programming language. | The codes are written any programming language. |
| Object-Oriented | Since codes are written in Java, it is object oriented and the user will get the benefits of OOPs | Since codes are written in any language, all the languages are not object-oriented thread-based. So, the user will not get the benefits of OOPs |
| Portability | It is portable. | It is not portable. |
| Persistence | It remains in the memory until it is not explicitly destroyed. | It is removed from the memory after the completion of the process-basedrequest. |
| Server Independent | It can use any of the web-server. | It can use the web-server that supports it. |
| Data Sharing | Data sharing is possible. | Data sharing is not possible. |
| Link | It links directly to the server. | It does not link the web server directly to the server. |
| HTTP server | It can read and set HTTP servers. | It can neither read nor set HTTP servers. |
| Cost | Construction and destruction of new threads are not costly. | Construction and destruction of the new processes are costly. |
| Speed | Its can speed is slower. | It can speed is faster. |

| Basis | Servlet | CGI |
|---|---|---|
| Platform dependency | It can be Platform dependent | It can be not Platform dependent. |

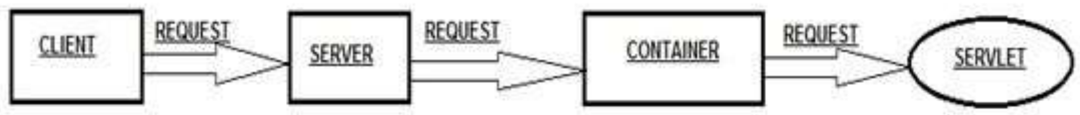
**Web Container** is a java application that controls servlet. Servlet does not have a main() method, So they require a container to load them.

Container is a place where servlet gets deployed.

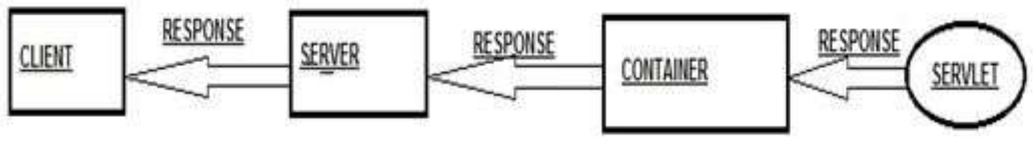
When a client sends a request to web server that contains a servlet, the server sends that request to the container rather than to servlet directly. Container then finds out the requested servlet and pass the Http Request and response to servlet and loads the servlet methods i.e. doGet() or do Post().

An example of a web container is Tomcat.

**1. Request made by client to server**



**2. Response received  by client**

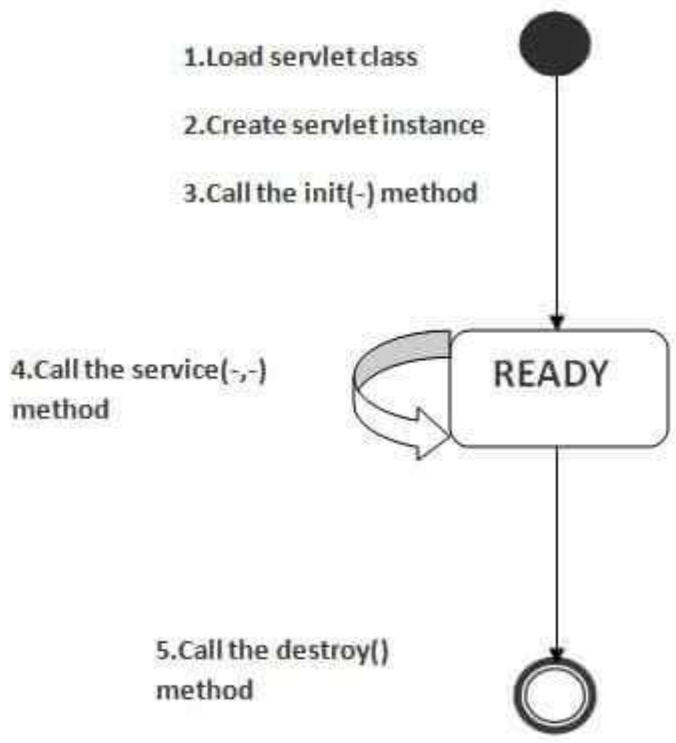


# Servlet life cycle, Servlet Context, Servlet Config Page context

the life cycle of the servlet:

1. Servlet class is loaded.
2. Servlet instance is created.

3. init method is invoked.
4. service method is invoked.
5. destroy method is invoked.


1.Load servlet class
2.Create servlet instance
3.Call the init(-) method
4.Call the service(-,-) method
READY
5.Call the destroy() method

As displayed in the above diagram, there are three states of a servlet: new, ready and end. The servlet is in new state if servlet instance is created. After invoking the init() method, Servlet comes in the ready state. In the ready state, servlet performs all the tasks. When the web container invokes the destroy() method, it shifts to the end state.

## 1) Servlet class is loaded

The classloader is responsible to load the servlet class. The servlet class is loaded when the first request for the servlet is received by the web container.

## 2) Servlet instance is created

The web container creates the instance of a servlet after loading the servlet class. The servlet instance is created only once in the servlet life cycle.

## 3) init method is invoked

The web container calls the init method only once after creating the servlet instance. The init method is used to initialize the servlet. It is the life cycle method of the javax.servlet.Servlet interface. Syntax of the init method is given below:

```
1. public void init(ServletConfig config) throws ServletException
```

## 4) service method is invoked

The web container calls the service method each time when request for the servlet is received. If servlet is not initialized, it follows the first three steps as described above then calls the service method. If servlet is initialized, it calls the service method. Notice that servlet is initialized only once. The syntax of the service method of the Servlet interface is given below:

```
1. public void service(ServletRequest request, ServletResponse response)
2.   throws ServletException, IOException
```

## 5) destroy method is invoked

The web container calls the destroy method before removing the servlet instance from the service. It gives the servlet an opportunity to clean up any resource for example memory, thread etc. The syntax of the destroy method of the Servlet interface is given below:

```
1. public void destroy()
```

## ServletConfig

(I) It is Servlet specific

(II) Parameters are as name-value pair in (init-param) inside (Servlet)

(III) Obtained by getServletConfig().

(IV) Each Servlet have its own ServletConfig Object

(V) Use only when one Servlet needs information shared by it.

```
(web-app)
 (context-param)
      |
      |
(/context param)
(Servlet)
 (init-param)
  (Param-name)
  (Param-value)
 (/init-param)
(/Servlet)
(/web-app)
```

## Servlet Context

(I) It is for whole application

(II) Parameters are as name-value pair in (Context-param) outside of (Servlet) inside (web-app)

(III) Obtained by getServletContext().

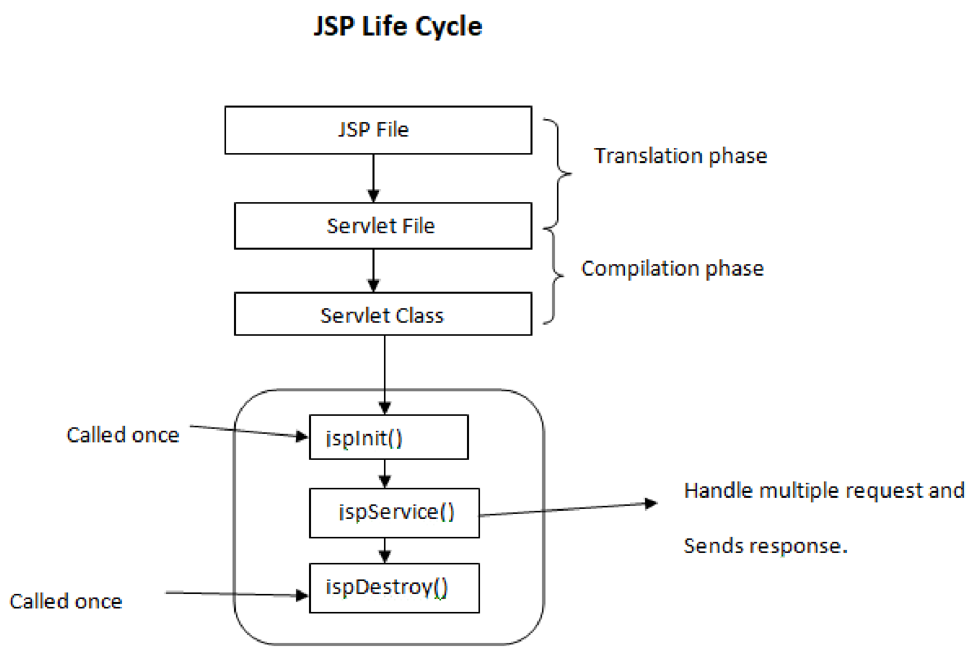
(IV) Only one is present for different Servlets.

(V) Use when whole application needs information shared by it.

```
(web-app)
 (Context-param)
  (param-name) Google (/param-name)
  (param-value) www.google.com (/param-value)
 (/Context-param)
 (Servlet)
   —
   —
   —
 (/Servlet)
(/web-app)
```

# JSP life cycle , JSP Directives tags, Action tags, Scripting tags

A Java Server Page life cycle is defined as the process that started with its creation which later translated to a servlet and afterward servlet lifecycle comes into play. This is how the process goes on until its destruction.



Following steps are involved in the JSP life cycle:

**Translation of JSP page to Servlet :**

This is the first step of the JSP life cycle. This translation phase deals with the Syntactic correctness of JSP. Here test.jsp file is translated to test.java.

**Compilation of JSP page :**

Here the generated java servlet file (test.java) is compiled to a class file (test.class).

**Classloading :**

Servlet class which has been loaded from the JSP source is now loaded into the container.

**Instantiation :**

Here an instance of the class is generated. The container manages one or more instances by providing responses to requests.

**Initialization :**

jspInit() method is called only once during the life cycle immediately after the generation of Servlet instance from JSP.

**Request processing :**

_jspService() method is used to serve the raised requests by JSP. It takes request and response objects as parameters. This method cannot be overridden.

**JSP Cleanup :**

In order to remove the JSP from the use by the container or to destroy the method for servlets jspDestroy()method is used. This method is called once, if you need to perform any cleanup task like closing open files, releasing database connections jspDestroy() can be overridden.

JSP directives are the elements of a JSP source code that guide the web container on how to translate the JSP page into it's respective servlet.
**Syntax: @**

<%@ directive attribute = "value"%>

Directives can have a number of attributes which you can list down as **key-value pairs** and separated by commas. The blanks between the @ symbol and the directive name, and between the last attribute and the closing %>, are optional.
**Different types of JSP directives :**

- import
- contentType
- extends
- info
- buffer
- language
- isELIgnored
- isThreadSafe
- autoFlush
- session
- pageEncoding
- errorPage
- isErrorPage

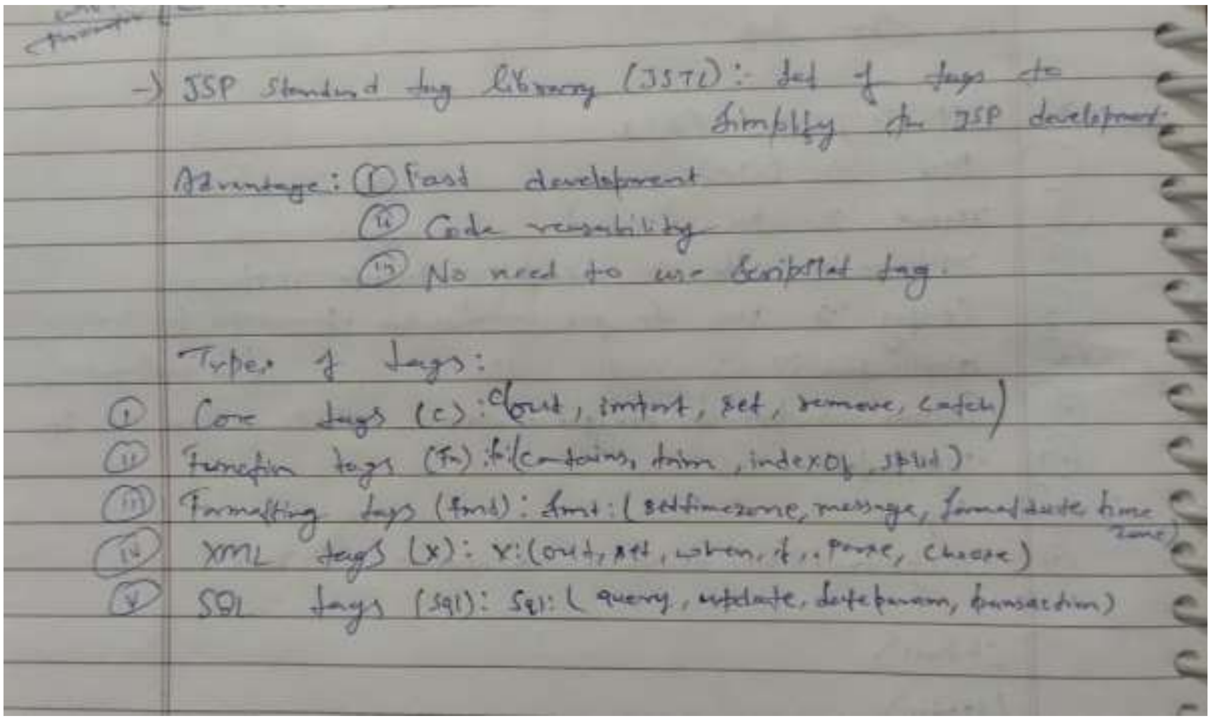
There are many JSP action tags or elements. Each JSP action tag is used to perform some specific tasks. The action tags are used to control the flow between pages and to use Java Bean. The Jsp action tags are given below.

| JSP Action Tags | Description |
|---|---|
| jsp:forward | forwards the request and response to another resource. |
| jsp:include | includes another resource. |
| jsp:useBean | creates or locates bean object. |
| jsp:setProperty | sets the value of property in bean object. |
| jsp:getProperty | prints the value of property of the bean. |
| jsp:plugin | embeds another components such as applet. |
| jsp:param | sets the parameter value. It is used in forward and include mostly. |
| jsp:fallback | can be used to print the message if plugin is working. It is used in jsp:plugin. |

The scripting elements provides the ability to insert java code inside the jsp. There are three types of scripting elements:

- scriptlet tag
- expression tag
- declaration tag

## JSTL (5 tags)  atleast 5 methods under each tag



-> JSP Standard tag library (JSTL): set of tags to simplify the JSP development.

Advantage: (i) Fast development
(ii) Code reusability
(iii) No need to use Scriptlet tag.

Types of tags:

(i) Core tags (c): c(out, import, set, remove, catch)
(ii) Function tags (fn): fn(contains, trim, indexOf, split)
(iii) Formatting tags (fmt): fmt: (settimezone, message, formatdate, time zone)
(iv) XML tags (x): x: (out, set, when, if, parse, choose)
(v) SQL tags (sql): sql: (query, update, dateparam, transaction)

# XML, Structure of XML.

An **XML (EXtensible Markup Language) Document** contains declarations, elements, text, and attributes. It is made up of entities (storing units) and It tells us the structure of the data it refers to. It is used to provide a standard format of data transmission. As it helps in message delivery, it is not always stored physically, i.e. in a disk but generated dynamically but its structure always remains the same.

### XML Tree Structure

An XML document has a self descriptive structure. It forms a tree structure which is referred as an XML tree. The tree structure makes easy to describe an XML document.

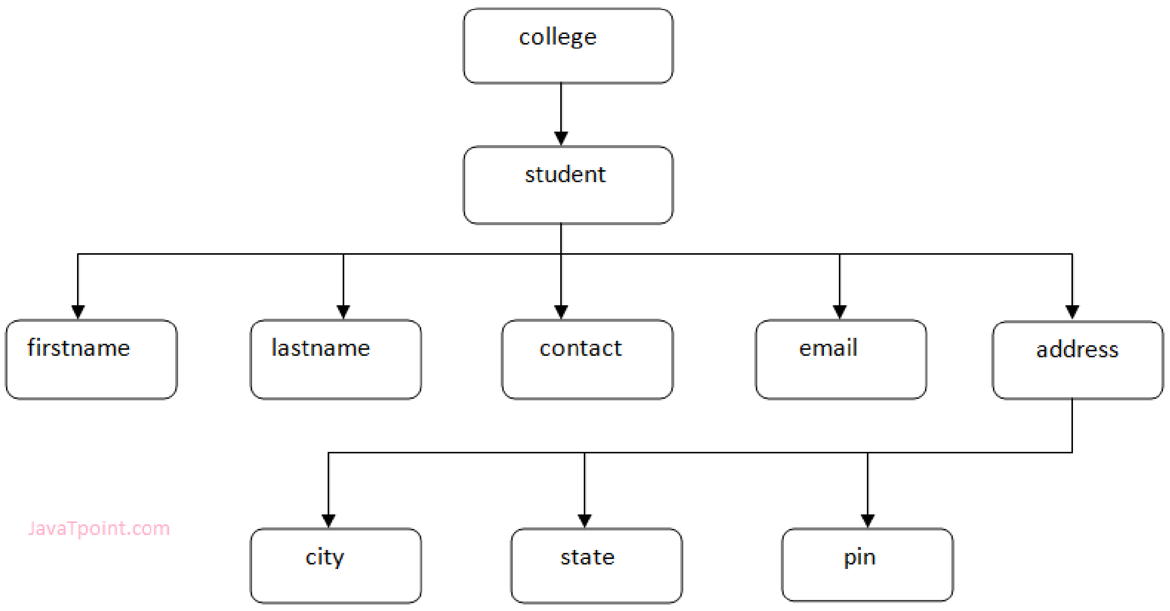
A tree structure contains root element (as parent), child element and so on. It is very easy to traverse all succeeding branches and sub-branches and leaf nodes starting from the root.

## Example of an XML document

```
<?xml version="1.0"?>
<college>
 <student>
    <firstname>Tamanna</firstname>
     <lastname>Bhatia</lastname>
    <contact>09990449935</contact>
```

```
        <email>tammanabhatia@abc.com</email>
        <address>
            <city>Ghaziabad</city>
            <state>Uttar Pradesh</state>
            <pin>201007</pin>
        </address>
    </student>
</college>
```

Let's see the tree-structure representation of the above example.



## Different XML Parsers their importance

An XML parser is a software library or package that provides interfaces for client applications to work with an XML document. The XML Parser is designed to read the XML and create a way for programs to use XML.

XML parser validates the document and check that the document is well formatted.

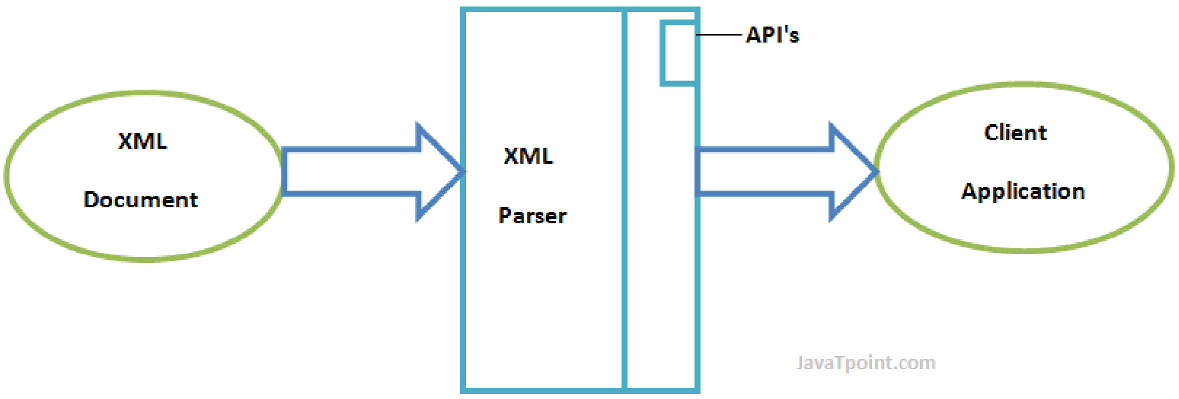
The goal of a parser is to transform XML into a readable code.

To ease the process of parsing, some commercial products are available that facilitate the breakdown of XML document and yield more reliable results.

Some commonly used parsers are listed below –

- **MSXML (Microsoft Core XML Services)** – This is a standard set of XML tools from Microsoft that includes a parser.
- **System.Xml.XmlDocument** – This class is part of .NET library, which contains a number of different classes related to working with XML.
- **Java built-in parser** – The Java library has its own parser. The library is designed such that you can replace the built-in parser with an external implementation such as Xerces from Apache or Saxon.
- **Saxon** – Saxon offers tools for parsing, transforming, and querying XML.
- **Xerces** – Xerces is implemented in Java and is developed by the famous open source Apache Software Foundation.

Let's understand the working of XML parser by the figure given below:



# DOM , SAX parsers and their interfaces

These are the two main types of XML Parsers:

1. DOM
2. SAX

## DOM (Document Object Model)

A DOM document is an object which contains all the information of an XML document. It is composed like a tree structure. The DOM Parser implements a DOM API. This API is very simple to use.

### Features of DOM Parser

A DOM Parser creates an internal structure in memory which is a DOM document object and the client applications get information of the original XML document by invoking methods on this document object.

DOM Parser has a tree based structure.

### Advantages

1) It supports both read and write operations and the API is very simple to use.

2) It is preferred when random access to widely separated parts of a document is required.

### Disadvantages

1) It is memory inefficient. (consumes more memory because the whole XML document needs to loaded into memory).

2) It is comparatively slower than other parsers.

---

## SAX (Simple API for XML)

A SAX Parser implements SAX API. This API is an event based API and less intuitive.

### Features of SAX Parser

It does not create any internal structure.

Clients does not know what methods to call, they just overrides the methods of the API and place his own code inside method.

It is an event based parser, it works like an event handler in Java.

### Advantages

1) It is simple and memory efficient.

2) It is very fast and works for huge documents.

### Disadvantages

1) It is event-based so its API is less intuitive.

2) Clients never know the full information because the data is broken into pieces.

# DTD

## What is DTD

DTD stands for **Document Type Definition**. It defines the legal building blocks of an XML document. It is used to define document structure with a list of legal elements and attributes.

## Purpose of DTD

Its main purpose is to define the structure of an XML document. It contains a list of legal elements and define the structure with the help of them.

## Description of DTD

**<!DOCTYPE employee :** It defines that the root element of the document is employee.

**<!ELEMENT employee:** It defines that the employee element contains 3 elements "firstname, lastname and email".

**<!ELEMENT firstname:** It defines that the firstname element is #PCDATA typed. (parse-able data type).

**<!ELEMENT lastname:** It defines that the lastname element is #PCDATA typed. (parse-able data type).

**<!ELEMENT email:** It defines that the email element is #PCDATA typed. (parse-able data type).