Unit 1 :Programming in Java

Java

- Java is a **programming language** and a platform.
- Java is a high level, robust, object-oriented and secure programming language.
- Java was developed by *Sun Microsystems* (which is now the subsidiary of Oracle) in the year 1995.
- ***James Gosling*** is known as the father of Java.
- Before Java, its name was ***Oak***. Since Oak was already a registered company, so James Gosling and his team changed the name from Oak to Java.

Types of Java Applications

1) Standalone Application
2) Web Application
3) Enterprise Application
4) Mobile Application

History of Java :

- The team initiated this project to develop a language for digital devices such as set-top boxes, television, etc.
- Originally C++ was considered to be used in the project, but the idea was rejected for several reasons(For instance C++ required more memory).
- The history of Java starts with the Green Team.
- **James Gosling, Mike Sheridan**, and **Patrick Naughton** initiated the Java language project in June 1991. The small team of sun engineers called **Green Team**.
- Initially it was designed for small, embedded systems in electronic appliances like set-top boxes.
- Firstly, it was called **"Greentalk"** by James Gosling, and the file extension was .gt.
- After that, it was called **Oak** and was developed as a part of the Green project. Oak is a symbol of strength and chosen as a national tree of many countries like the U.S.A., France, Germany, Romania, etc.
- In 1995, Oak was renamed as **"Java"** because it was already a trademark by Oak Technologies.

Why Java Programming named "Java"?

- The team gathered to choose a new name. The suggested words were "dynamic", "revolutionary", "Silk", "jolt", "DNA", etc. They wanted something that reflected the essence of the technology: revolutionary, dynamic, lively, cool, unique, and easy to spell, and fun to say.

- Java is an island in Indonesia where the first coffee was produced (called Java coffee). It is a kind of espresso bean. Java name was chosen by James Gosling while having a cup of coffee nearby his office.

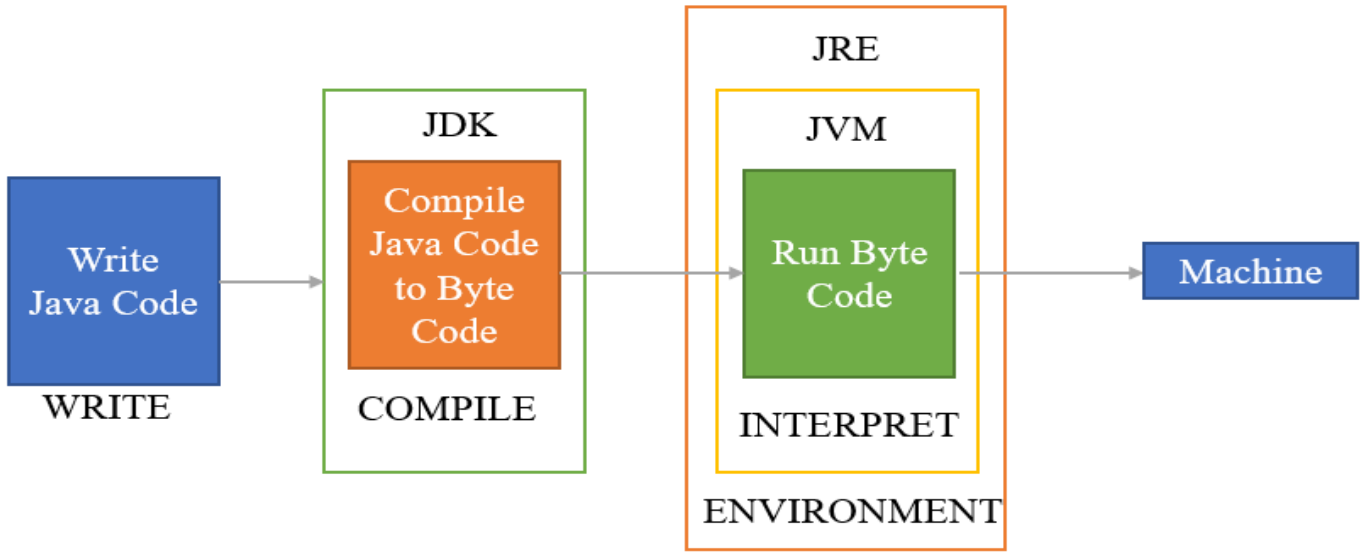
Java Architecture:

- **Java Architecture** is a collection of components, i.e., **JVM, JRE,** and **JDK**.
- It integrates the process of interpretation and compilation.
- It defines all the processes involved in creating a Java program.
- **Java Architecture** explains each and every step of how a program is compiled and executed.

Components of Java Architecture:

The Java architecture includes the three main components:

1. Java Virtual Machine (JVM)
2. Java Runtime Environment (JRE)
3. Java Development Kit (JDK)



JDK	JRE	JVM
The full form of JDK is Java Development Kit.	The full form of JRE is Java Runtime Environment.	The full form of JVM is Java Virtual Machine.
JDK is a software development kit to develop applications in Java.	It is a software bundle which provides Java class libraries with necessary components to run Java code.	JVM executes Java byte code and provides an environment for executing it.
JDK is platform dependent.	JRE is also platform dependent.	JVM is highly platform dependent.
It contains tools for developing, debugging, and monitoring java code.	It contains class libraries and other supporting files that JVM requires to execute the program.	Software development tools are not included in JVM.
It is the superset of JRE	It is the subset of JDK.	JVM is a subset of JRE.
The JDK enables developers to create Java programs that can be executed and run by the JRE and JVM.	The JRE is the part of Java that creates the JVM.	It is the Java platform component that executes source code.
JDK comes with the installer.	JRE only contain environment to execute source code.	JVM bundled in both software JDK and JRE.

Compiling:

- Compiling a Java program means taking the programmer-readable text in your program file (also called source code) and converting it to bytecodes, which are platform-independent instructions for the Java VM.

- The Java compiler is invoked at the command line on Unix and DOS shell operating systems as follows:

javac Simple.java

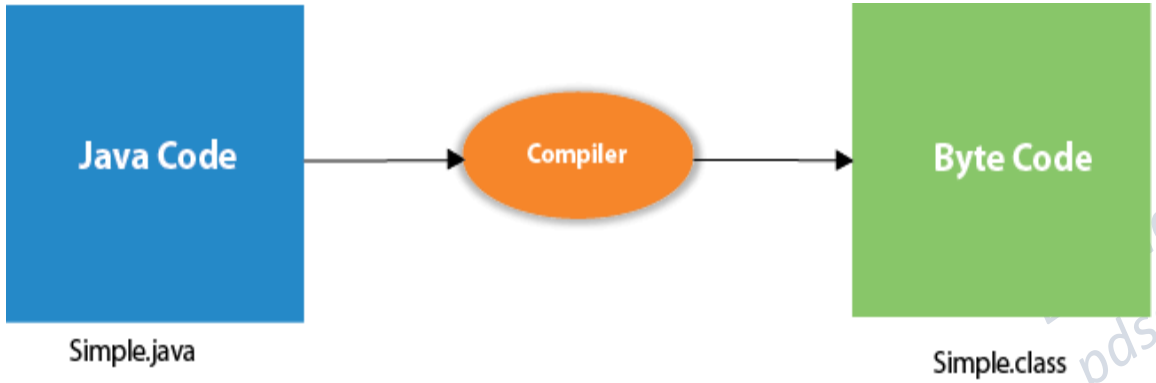
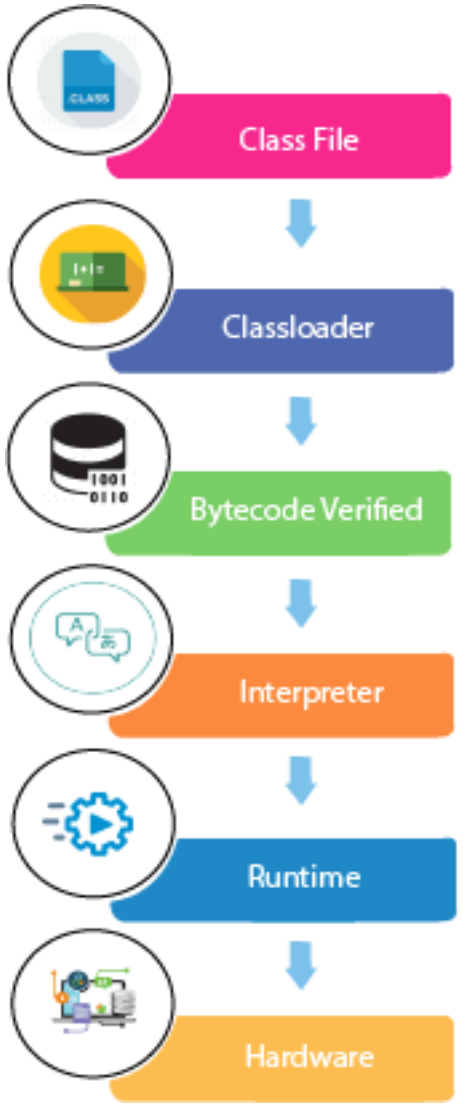


Fig. Compile Time

Run a Java program :

- ❖ The class files generated by the compiler are independent of the machine or the OS, which allows them to be run on any system.
- **Classloader:** It is the subsystem of JVM that is used to load class files.
- **Bytecode Verifier:** Checks the code fragments for illegal code that can violate access rights to objects.
- **Interpreter:** Read bytecode stream then execute the instructions.

Command to run the Java program:
java Simple



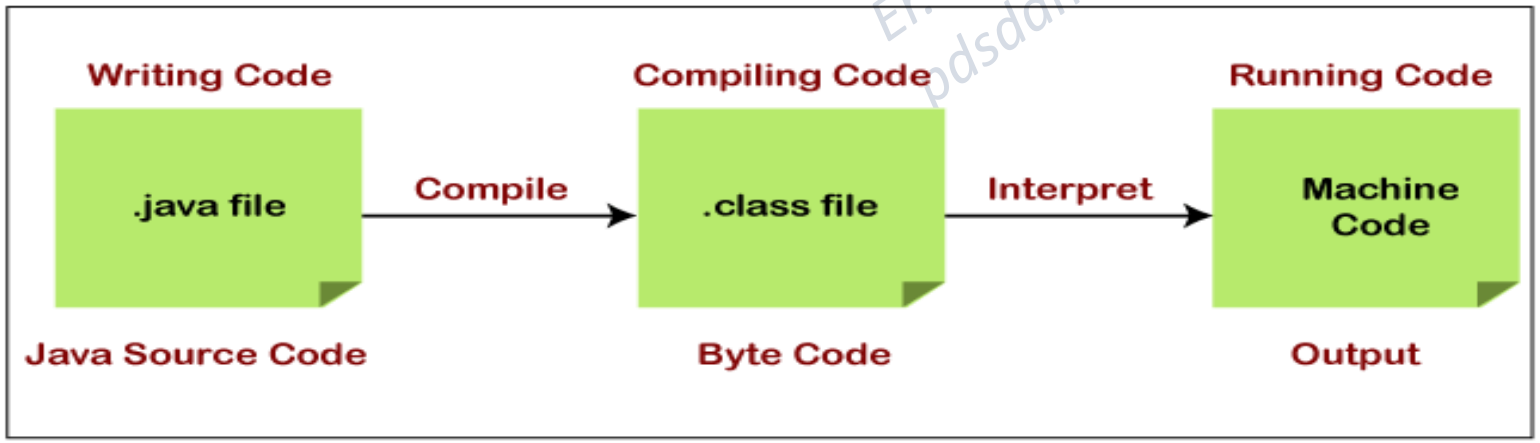
Interpreting and running Java Program

- Interpreting and running a Java program is process of **executing the Java Program**. In this process the **Java VM** is invoked and it takes the byte code and interprets it. In this process the byte code is converted to platform-dependent machine codes so that your computer can understand and run the program.

- Once your program successfully compiles into Java bytecodes, you can interpret and run your applications on any Java VM or JVM enabled web browser as applet.

How does the Java interpreter work?

- To convert the byte code into machine code, we deploy the .class file on the JVM. The JVM converts that code into machine code using the Java interpreter. The JVM uses the interpreter at runtime, after that it execute the code on the host machine.



Interpreter	Compiler
It translates the code instruction by instruction.	It translates the entire program at once.
Its execution is slower.	Its execution is faster.
Its compile time is less.	It takes more time to compile the code.
It does not generate the intermediate object code.	It generates the intermediate object code.
It compiles the program until an error is found.	All the errors show once at the end of the compilation.
Python, PHP, Ruby, and Perl use an interpreter.	Java, C++, Scala, and C uses a compiler.

Features of Java/ Java Buzzwords

1. Simple, Readable, and Easy to Learn

- ❖ Java was designed to be straightforward and easy to understand.
- ❖ Its syntax is similar to other C-style languages like C++ and C#, making it accessible to developers with experience in those languages.

2. Platform Independence

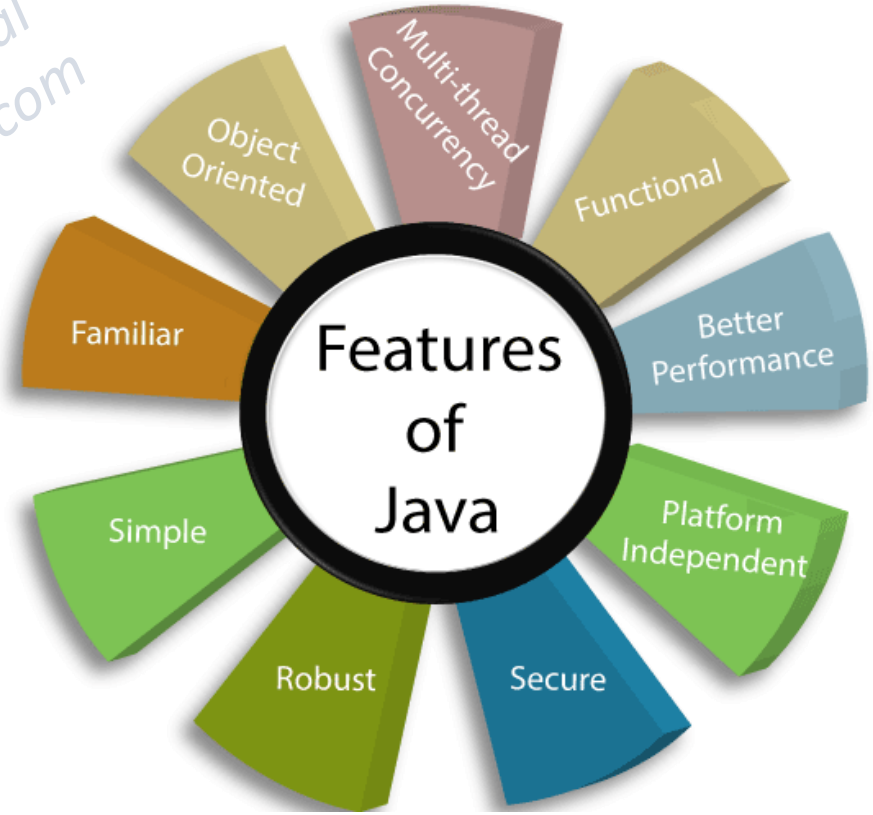
- ➢ Java is designed to be platform-independent, meaning you can write code on one platform (like Windows) and run it on another (like Linux) without modification.
- ➢ This is achieved through the use of the Java Virtual Machine (JVM), which translates Java code into bytecode that can be executed on any platform with a compatible JVM.

3. Portable

- ➢ Java is portable because it facilitates you to carry the Java bytecode to any platform. It doesn't require any implementation.

4. High-performance

- ➢ Java is faster than other traditional interpreted programming languages because Java bytecode is "close" to native code.
- ➢ It is still a little bit slower than a compiled language (e.g., C++). Java is an interpreted language that is why it is slower than compiled languages, e.g., C, C++, etc.



5. Object-Oriented:

- Java is an object-oriented programming language, which means it is based on the concept of objects that encapsulate data and behavior. This approach promotes modularity, reusability, and easier maintenance of code.
- The basic concepts of OOPs are:
  Object, Class, Inheritance,Polymorphism,Abstraction,Encapsulation

6. Multithreaded:

- Java supports multithreading, allowing developers to create applications that can efficiently execute multiple tasks concurrently.

7. Secure:

- Java has built-in security features that make it a suitable choice for building applications that need to run securely over networks or the internet.
- Its security model includes features like sandboxing, classloaders, and access controls.

8. Robust and Reliable:

- Java's strict compile-time checking and runtime exception handling contribute to its robustness. This helps catch errors early in the development process and promotes stable and reliable software.

9. Dynamic:

- Java supports dynamic class loading, allowing classes to be loaded at runtime. This feature is crucial for creating flexible and extensible applications.

Path and ClassPath variables:

Path

- An environment variable is used by the operating system to find the executable files.
- PATH setting up an environment for the operating system. Operating System will look in this PATH for executables.
- In path variable, we must place .\bin folder path
- PATH is used by CMD prompt to find binary files.

Classpath

- An environment variable is used by the Java compiler to find the path of classes.
- Classpath setting up the environment for Java. Java will use to find compiled classes.
- In classpath, we must place .\lib\jar file or directory path in which .java file is available.
- CLASSPATH is used by the compiler and JVM to find library files.

Setting up your Computer for Java Environment:

1. User variables :

JAVA_HOME : path of jdk (C:\Program Files\Java\jdk version)

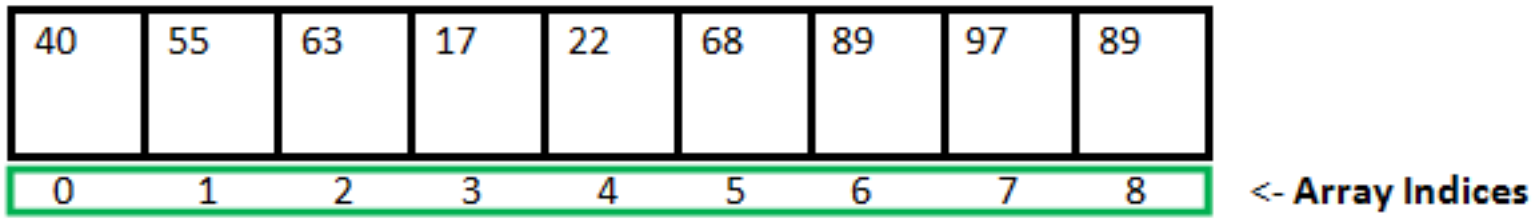
2. System variables :

- Select Path variable
- Click on edit
- Add path of both JDK and JRE

e.g. C:\Program Files\Java\jdk version\bin
C:\Program Files\Java\jre version\bin

Arrays:

- An array is a collection of similar types of data.
- Arrays are used to store multiple values in a single variable.
- A Java array variable can also be declared like other variables with [] after the data type.
- The variables in the array are ordered, and each has an index beginning from 0.
- Java array can be also be used as a static field, a local variable, or a method parameter.
- The **size** of an array must be specified by int or short value and not long.
- An array can contain primitives (int, char, etc.) and object (or non-primitive) references of a class depending on the definition of the array.
- In the case of primitive data types, the actual values are stored in contiguous memory locations. In the case of class objects, the actual objects are stored in a heap segment.



Array Length = 9
First Index = 0
Last Index = 8

Types:

1. Single Dimensional Array
2. Multidimensional Array

1. Single Dimensional Array

Syntax to Declare an Array in Java :

1. dataType[] arrayName; (or)
2. dataType []arrayName; (or)
3. dataType arrayName[];

- dataType - it can be primitive data types like int, char, double, byte, etc. or Java objects
- arrayName - it is an identifier

Instantiation of an Array in Java :

arrayName = **new** datatype[size];

Initialize Arrays in Java :

In Java, we can initialize arrays during declaration. For example,

```
//declare and initialize and array

int[] age = {12, 4, 5, 2, 5};
```

- ***Note that we have not provided the size of the array. In this case, the Java compiler automatically specifies the size by counting the number of elements in the array (i.e. 5).***

❖ We can also initialize arrays in Java, using the index number. For example,

```
int[] age = new int[5];
// initialize array
age[0] = 12;
age[1] = 4;
age[2] = 5;
 ..
```

2. Multidimensional Array

❖ In Java, a multidimensional array is an array of arrays. This means that each element of a multidimensional array is itself an array.

❖ Data is stored in row and column-based index (also known as matrix form).

Syntax to Declare Multidimensional Array in Java

1.dataType[][] arrayRefVar; (or)
2.dataType [][]arrayRefVar; (or)
3.dataType arrayRefVar[][]; (or)
4.dataType []arrayRefVar[];

Example to instantiate Multidimensional Array in Java

```
int[][] arr=new int[3][3];//3 row and 3 column
```

Declaring and initializing 2D array

```
int arr[][]={{1,2,3},{2,4,5},{4,4,5}};
```

Jagged Array in Java

- In Java, a jagged array is a special type of multidimensional array where each row can have a different number of columns.
- In other words, it is an array of arrays with different number of columns.

Instantiate

```
int[][] jaggedArray = new int[3][];
```

Declaring

```
int[][] jaggedArray = {{1,2,3,4}, {5,6}, {7,8,9}};
```

Loop statements / Looping statement

- **Looping statement** are the statements execute one or more statement repeatedly several number of times.
- In java programming language there are three types of loops; while, for and do-while.

Types :

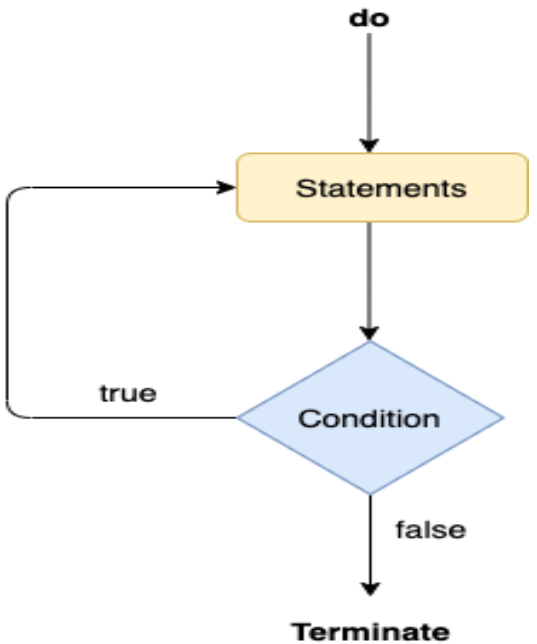
- do while loop
- while loop
- for loop
- for-each loop

a. do while loop:

- The do while loop checks the condition at the end of the loop after executing the loop statements.
- When the number of iteration is not known and we have to execute the loop at least once, we can use do-while loop.
- It is also known as the exit-controlled loop since the condition is not checked in advance.

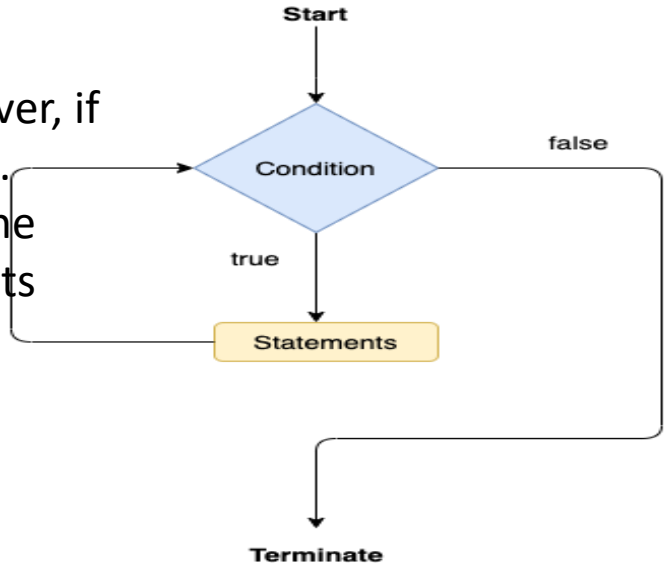
Syntax :

```
do
{
//statements
} while (condition);
```



b. while loop

- The while loop is also used to iterate over the number of statements multiple times. However, if we don't know the number of iterations in advance, it is recommended to use a while loop.
- It is also known as the entry-controlled loop since the condition is checked at the start of the loop. If the condition is true, then the loop body will be executed; otherwise, the statements after the loop will be executed.

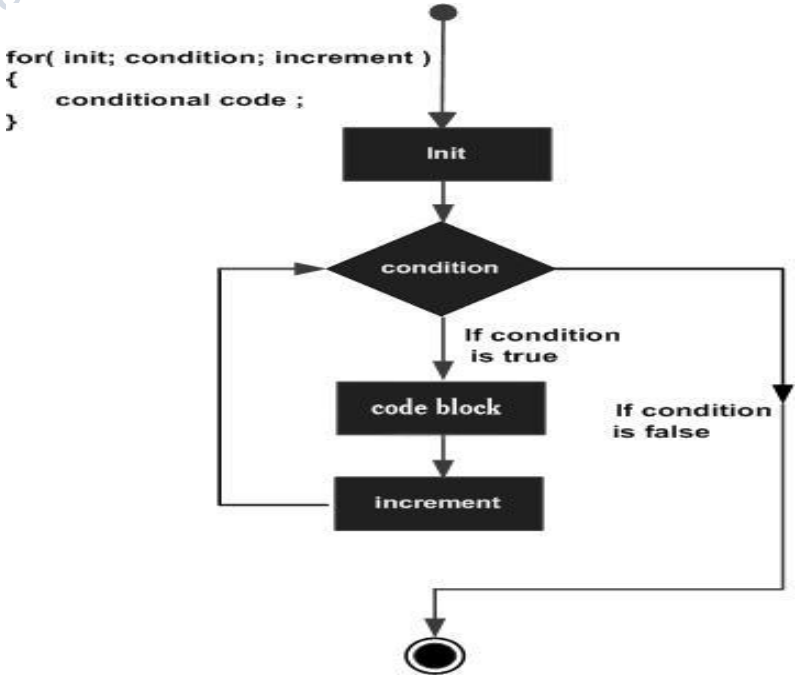


Syntax :

```
while(condition){
 //looping statements
}
```

c. for loop :

- The for statement consumes the initialization, condition and increment/decrement in one line thereby providing a shorter, easy to debug structure of looping.



Syntax:

for (*statement 1*; *statement 2*; *statement 3*) {
 // code block to be executed
 }

Statement 1 is executed (one time) before the execution of the code block.
Statement 2 defines the condition for executing the code block.
Statement 3 is executed (every time) after the code block has been executed.

d. for-each loop / Enhanced For Loop

- The Java for-each loop or enhanced for loop is introduced since Java5.
- It provides an alternative approach to traverse the array or collection in Java.
- It is mainly used to traverse the array or collection elements.
- It is known as the for-each loop because it traverses each element one by one.
- In Java, the **for-each** loop is used to iterate through elements of **arrays** and collections (like **ArrayList**).

Advantages

- It makes the code more readable.
- It eliminates the possibility of programming errors.

Drawback

- it cannot traverse the elements in reverse order.
- you do not have the option to skip any element because it does not work on an index basis.

for-each Loop Sytnax:

```
for(dataType item : array) {
...
}
```

Here,

array - an array or a collection

item - each item of array/collection is assigned to this variable

dataType - the data type of the array/collection

Fundamentals of Classes:

- A class is a group of objects which have common properties.
- It is a template or blueprint from which objects are created.
- It is a logical entity.
- It can't be physical.

A class in Java can contain:

- **Fields**
- **Methods**
- **Constructors**
- **Blocks**
- **Nested class and interface**

```java
public class Employee {

    //Fields, data members, Instance variables
    private int employeeId;
    private String employeeName;
    private String employeeAddress;

    //Constructors
    public Employee(int employeeId, String employeeName, String employeeAddress) {
        this.employeeId = employeeId;
        this.employeeName = employeeName;
        this.employeeAddress = employeeAddress;
    }
    //Methods
    public String showEmployee() {
        return "Id : "+employeeId + "\nName : "+employeeName + "\nAddress : "+employeeAddress;
    }


    public static void main(String[] args) {
        //Creating a class instance
        Employee employee = new Employee(10, "Hari Dahal", "Baneshwor 31, Kathmandu");

        //Calling methods
        String employeeDetails = employee.showEmployee();
        System.out.println(employeeDetails);
    }
}
```

Fig. General form of class

Object :

- Any entity that has state and behavior is known as an object. For example, a chair, pen, table, keyboard, bike, etc. It can be physical or logical.
- An Object can be defined as an instance of a class.
- An object contains an address and takes up some space in memory. Objects can communicate without knowing the details of each other's data or code. The only necessary thing is the type of message accepted and the type of response returned by the objects.

Create Object in Java :

1. Using new keyword:

- Using the **new** keyword is the most popular way to create an object or instance of the class.
- When we create an instance of the class by using the new keyword, it allocates memory (heap) for the newly created **object** and also returns the **reference** of that object to that memory.

- The syntax for creating an object is:

```
ClassName object = new ClassName();
```

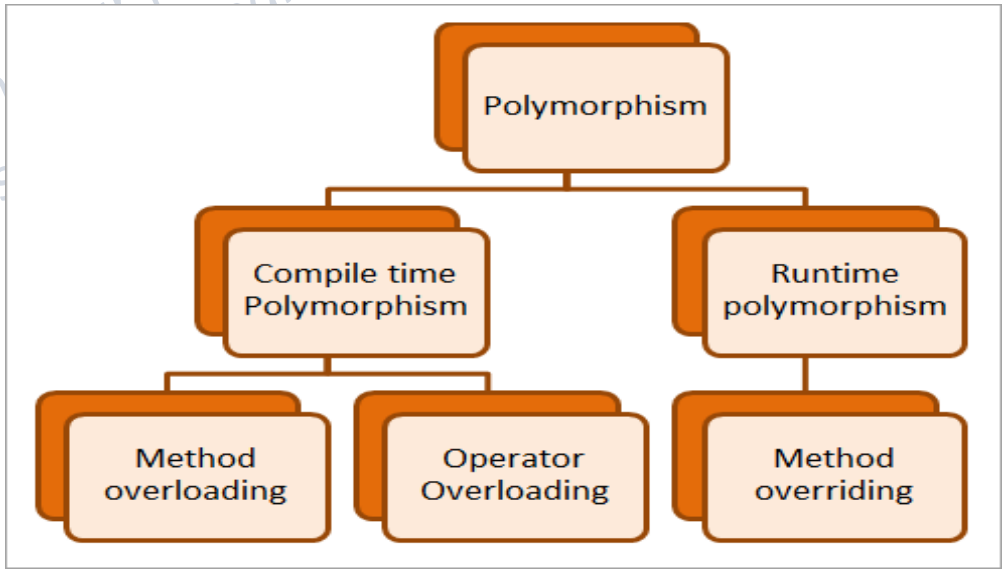
Polymorphism :

- The word polymorphism means having many forms.
- In simple words, we can define polymorphism as the ability of a message to be displayed in more than one form.
- In Java, polymorphism refers to the ability of a class to provide different implementations of a method, depending on the type of object that is passed to the method.

1. Compile-time polymorphism :

- Compile-time polymorphism is also known as "**Static polymorphism**".
- As the name suggests, the compile-time polymorphism is performed at compile-time.
- This type of polymorphism is achieved by **function overloading** or **operator overloading**.

- ***Note: Java doesn't support the Operator Overloading.***



A. Method Overloading:

- When there are multiple functions with the same name but different parameters then these functions are said to be **overloaded**.

- There are two ways to overload the method in java:

i. By changing number of arguments
ii. By changing the data type

Note: In Java, Method Overloading is not possible by changing the return type of the method only.

2. Runtime polymorphism:

- **Runtime polymorphism** or **Dynamic Method Dispatch** is a process in which a call to an overridden method is resolved at runtime rather than compile-time.
- In this process, an overridden method is called through the reference variable of a superclass.
- The determination of the method to be called is based on the object being referred to by the reference variable.
- Runtime polymorphism in Java is achieved by using "**method overriding**".

Upcasting:

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

Rules of Runtime Polymorphism:

- Methods of child and parent class must have the same name.
- Methods of child and parent class must have the same parameter.
- IS-A relationship is mandatory (inheritance).

Limitations of Runtime Polymorphism:

- One cannot override the private methods of a parent class.
- One cannot override Final methods.
- One cannot override static methods.

A. Method Overriding:

- Method overriding is a technique by which a method in the parent class is redefined or overridden in the child class.
- When the method is overridden in a class, the dynamic method dispatch technique resolves the overridden method call at runtime and not at compile time.

Rules for Java Method Overriding

1. The method must have the same name as in the parent class
2. The method must have the same parameter as in the parent class.
3. There must be an IS-A relationship (inheritance).

Access Privileges

❖ In Java, access privileges, also known as access modifiers, are keywords that determine the visibility and accessibility of classes, methods, fields, and other members within a class or between different classes.

Java provides four main access privileges:

Default: Default has scope only inside the same package
Public: Public has scope that is visible everywhere
Protected: Protected has scope within the package and all sub classes
Private: Private has scope only within the classes

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

Inheritance :

- Inheritance is one of the Basic Concepts of OOPs in which one object acquires the properties and behaviors of the parent object. It's creating a parent-child relationship between two classes.
- It helps to reuse the code and establish a relationship between different classes.

As we can see in the image :
A child inherits the properties from his father.

Similarly, in Java, there are two classes:

1. Parent class (Super or Base class)
2. Child class (Subclass or Derived class)

A class which inherits methods and values from the superclass known as Child class.

whereas

A class from which the child class inherits all the methods and values known as Parent Class.

extends Keyword :

- The extends keyword in Java indicates that the child class inherits or acquires all the properties of the parent class.
- This keyword basically establishes a relationship of an inheritance among classes.
- If a class extends another class, then we say that it has acquired all the properties and behavior of the parent class.
- We use the extends keyword in Java between two class names that we want to connect in the Inheritance relationship.

Note :

- ***It is not possible to extend multiple classes in Java because there is no support for multiple inheritances in Java. And therefore we cannot write multiple class names after the extended keyword.***

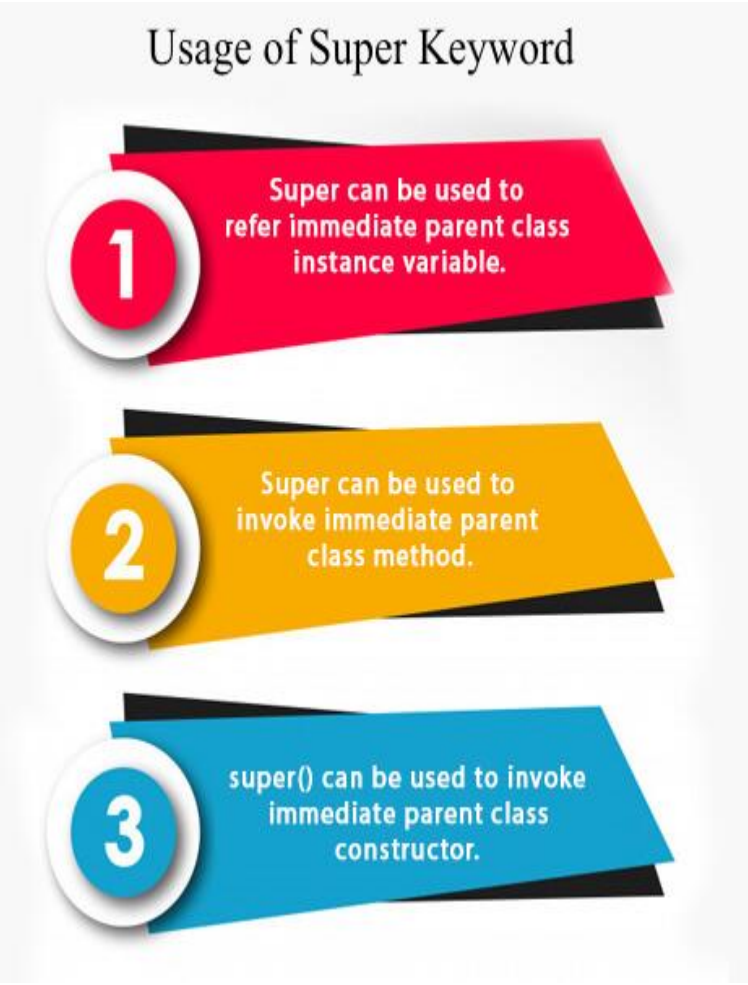
Syntax:

class ParentClass{ ...}

class ChildClass **extends** ParentClass { ... }

Super Keyword:

- The **super** keyword in java is a reference variable that is used to refer parent class objects.
- It is majorly used in the following contexts:



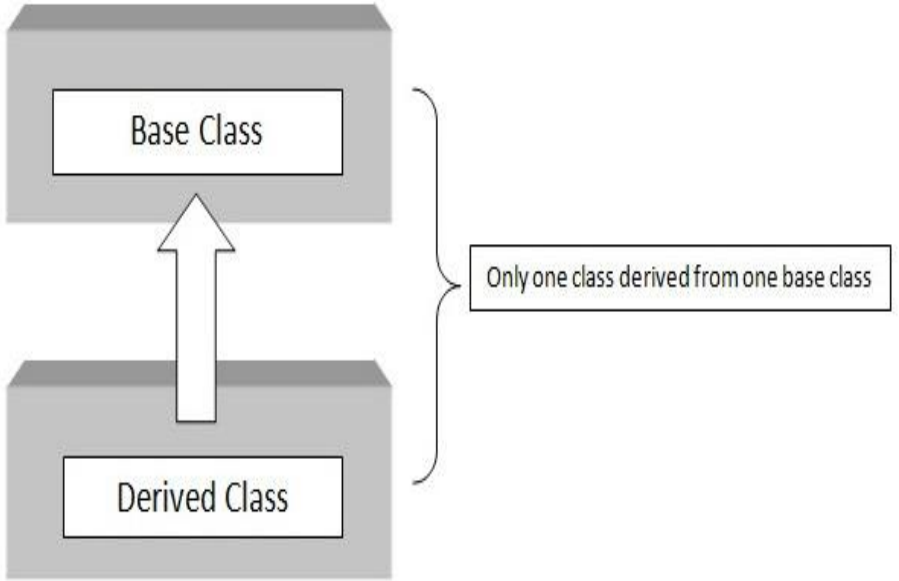
super	super()
The super keyword in Java is a reference variable that is used to refer parent class objects.	The super() in Java is a reference variable that is used to refer parent class constructors.
super can be used to call parent class' variables and methods.	super() can be used to call parent class' constructors only.
The variables and methods to be called through super keyword can be done at any time,	Call to super() must be first statement in Derived(Student) Class constructor.
If one does not explicitly invoke a superclass variables or methods, by using super keyword, then nothing happens	If a constructor does not explicitly invoke a superclass constructor by using super(), the Java compiler automatically inserts a call to the no-argument constructor of the superclass.

Different types of Inheritance in Java :

1. Single Inheritance
2. Multilevel Inheritance
3. Hierarchical Inheritance
4. Multiple Inheritance(Only through Interfaces)
5. Hybrid Inheritance

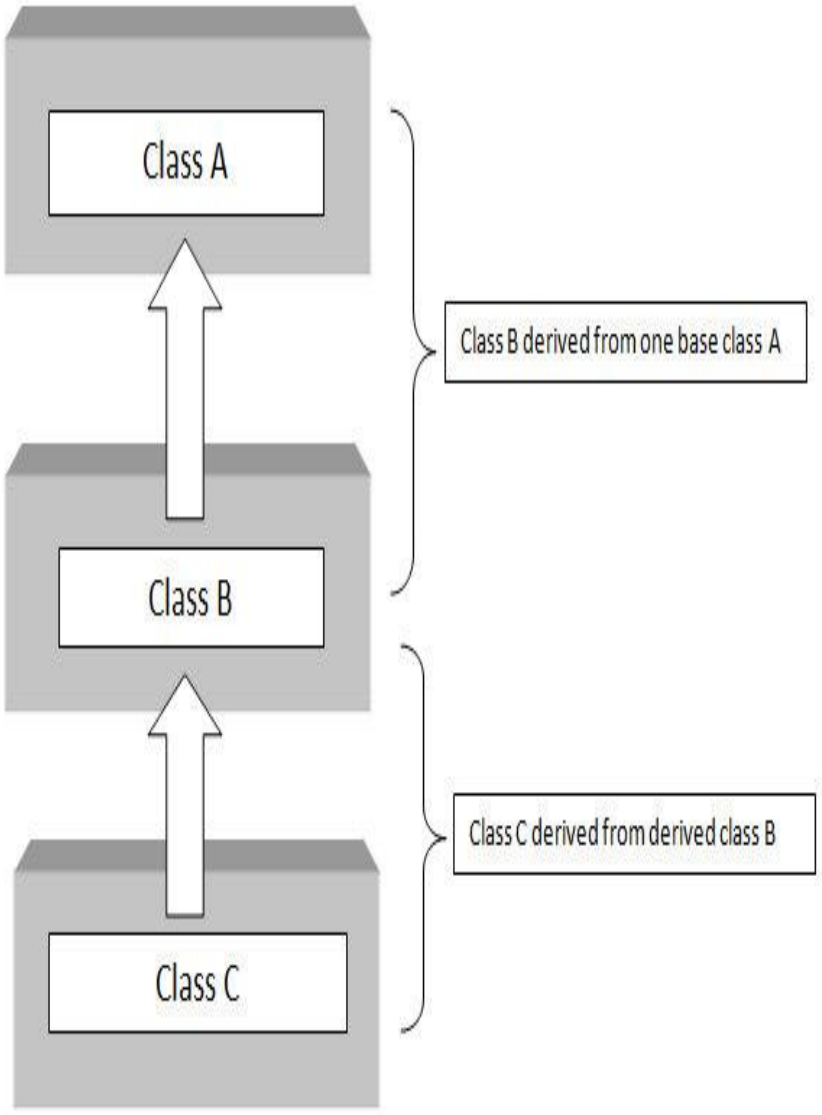
Single Inheritance :

- In single inheritance, a single child class inherits data and methods from its parent class.
- In this case, a child class can access all the methods and the variables of the parent class.



Multilevel Inheritance :

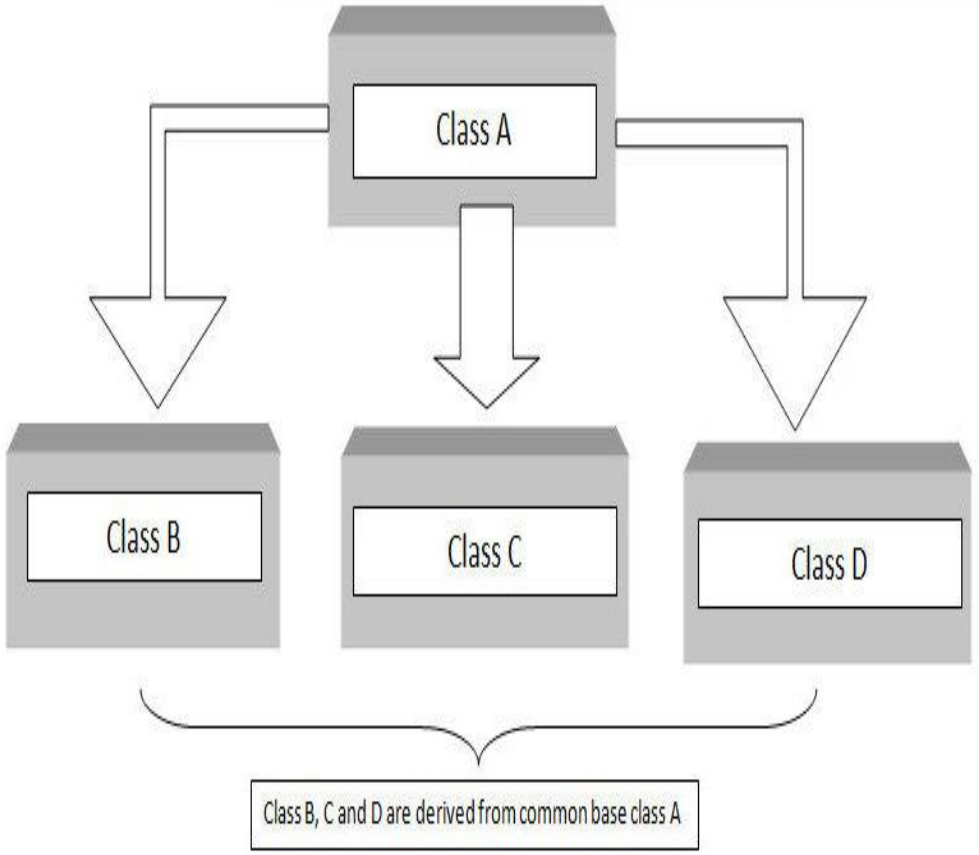
- The multi-level inheritance includes the involvement of at least two or more than two classes.
- One class inherits the features from a parent class and the newly created sub-class becomes the base class for another new class.



Hierarchical Inheritance:

- If more than one class is inherited from the base class, it's known as hierarchical inheritance.

- In hierarchical inheritance, all features that are common in child classes are included in the base class.



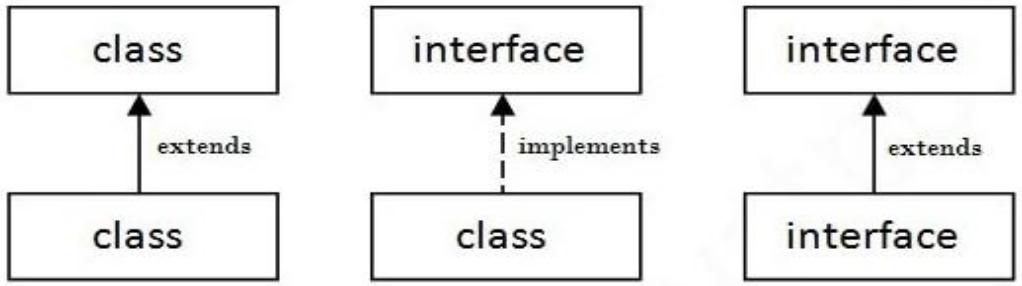
Java Interface :

- Interface can be defined as a container that stores the signatures of the methods to be implemented in the code segment.

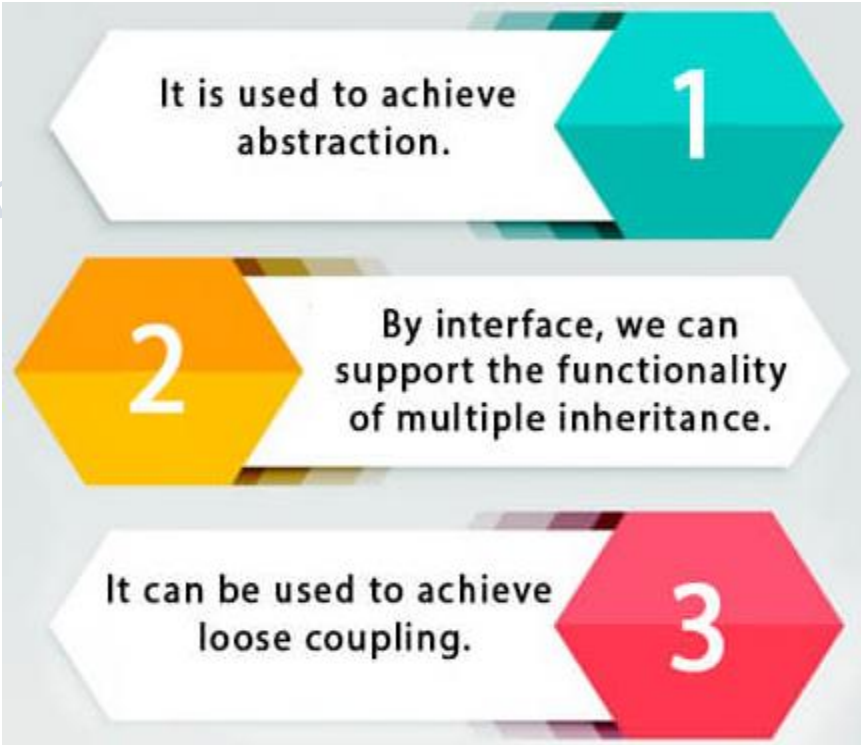
or,

- An **Interface in Java** programming language is defined as an abstract type used to specify the behavior of a class.
- A Java interface contains static constants and abstract methods.
- All methods in the interface are implicitly public and abstract.
- In Java, interfaces are declared using the interface keyword.
- Java Interface also **represents the IS-A relationship**.

- As shown in the figure given below :
  - a class extends another class
  - an interface extends another interface,
  - a **class implements an interface**.



Reasons to use interface



Note : ***The Java compiler adds public and abstract keywords before the interface method. Moreover, it adds public, static and final keywords before data members.***

Class	Interface
The keyword used to create a class is “class”	The keyword used to create an interface is “interface”
A class can be instantiated i.e, objects of a class can be created.	An Interface cannot be instantiated i.e, objects cannot be created.
Classes does not support multiple inheritance.	Interface supports multiple inheritance.
It can be inherit another class.	It cannot inherit a class.
It can be inherited by another class using the keyword ‘extends’.	It can be inherited by a class by using the keyword ‘implements’ and it can be inherited by an interface using the keyword ‘extends’.
It can contain constructors.	It cannot contain constructors.
It cannot contain abstract methods.	It contains abstract methods only.
Variables and methods in a class can be declared using any access specifier(public, private, default, protected)	All variables and methods in a interface are declared as public.
Variables in a class can be static, final or neither.	All variables are static and final.

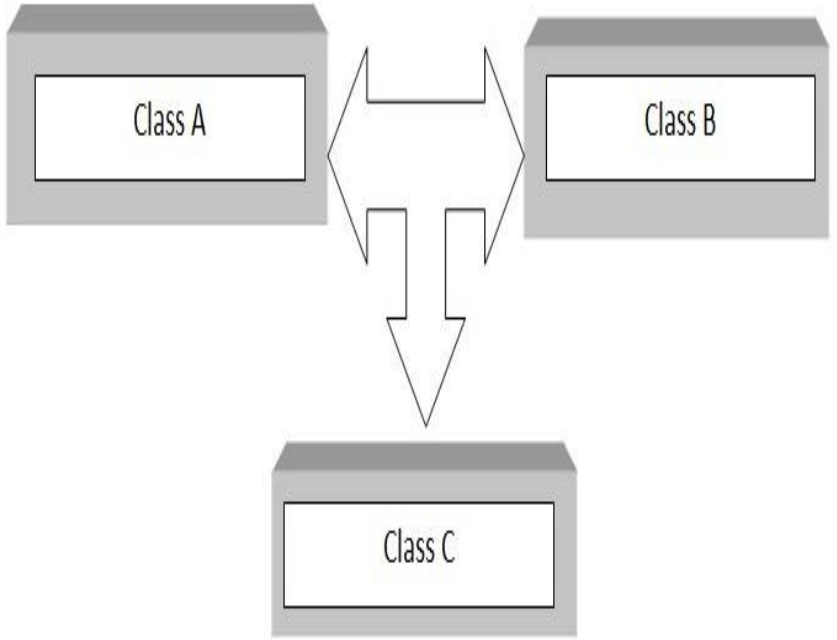
Multiple Inheritance :

- When one class extends more than one classes then this is called **multiple inheritance.** For example: class C extends class A and B then this type of inheritance is known as **multiple inheritance.**

- Multiple inheritance is not supported by Java because of ambiguity problem. This means that a class cannot extend more than one class.

public class C extends A, B{}

- To achieve multiple inheritance in Java, we must use the interface.



Hybrid Inheritance :

- Hybrid inheritance in Java is a combination of two or more types of inheritances.
- The purpose of using hybrid inheritance in Java is to modularize the codebase into well-defined classes and provide code reusability.

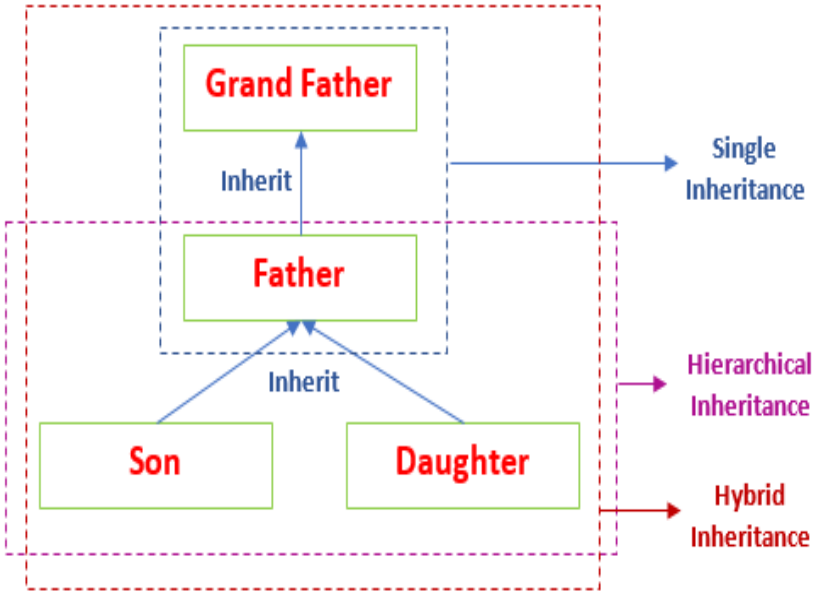


Fig 1.

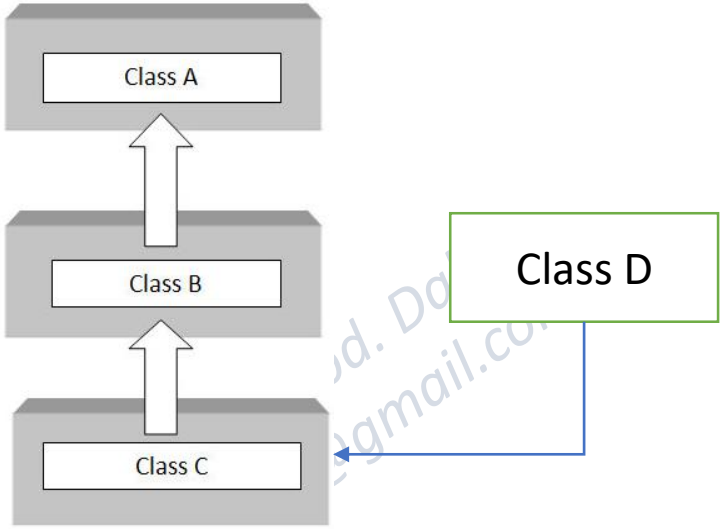


Fig 2.

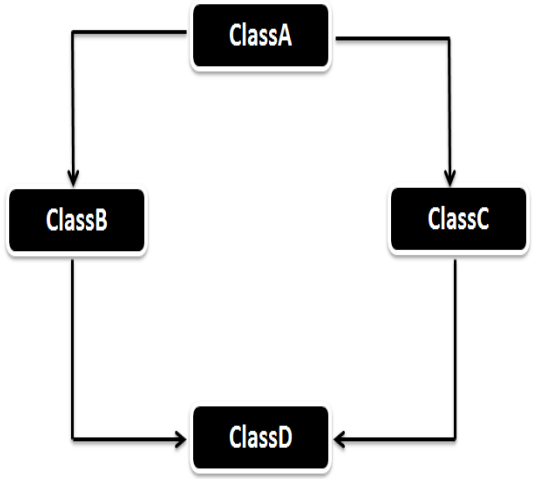


Fig 3.

Final Class in Java :

- In java, the final keyword can be used with variable, method, or class.
- A class that is declared with the final keyword is known as the final class.
- A final class can't be inherited by subclasses.

Syntax:

accessModifier **final** **class** className

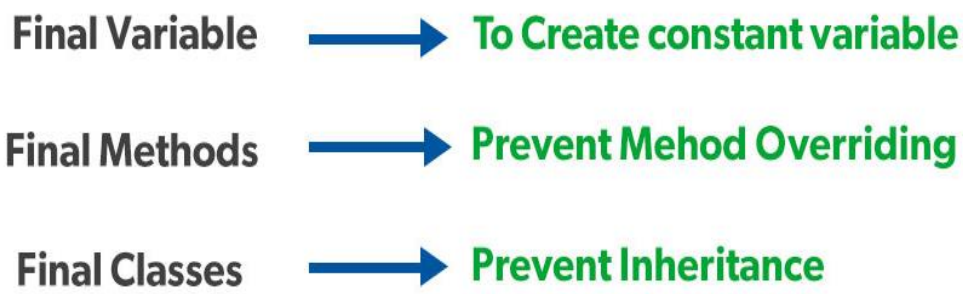
```
{
// Body of class
}
```

When to use a final class in java?

1. A final class is introduced in JDK to prevent the inheritance of class. Suppose I have a class that having some personal or secured information and you want that class should not be extended by any class. Then you should use the final keyword with class.

2. To create an immutable class then the final keyword is mandatory. It means an immutable class is always a final class.

3. The following are different contexts where final is used



Java Packages :

- A package as the name suggests is a pack(group) of classes, interfaces and other packages.
- In java we use packages to organize our classes and interfaces.

Types of packages in Java :

1. Built-in Packages (packages from the Java API) :

- Built-in packages or predefined packages are those that come along as a part of JDK (Java Development Kit) to simplify the task of Java programmer.
- They consist of a huge number of predefined classes and interfaces that are a part of Java API's.
- Some of the commonly used built-in packages are :

- **java.lang,**
- **java.io,**
- **java.util,**
- **java.applet, etc.**

2. User-defined Packages (create your own packages) :

- User-defined packages are those which are developed by users in order to group related classes, interfaces, and sub-packages.

How to Create a package?

Creating a package is a simple task as follows :

- Choose the name of the package
- Include the package command as the first line of code in your Java Source File.
- The Source file contains the classes, interfaces, etc you want to include in the package
- Compile to create the Java packages

Following statement creates a package named texas***:***

package texas;

Note: *If you omit the package statement, the class names are put into the default package, which has no name. Though the default package is fine for short programs, it is inadequate for real applications.*

import Keyword :

- The import keyword is used to import a package, class or interface.

How to access package from another package?
There are three ways to access the package from outside the package.

1. import package.*;'
- If you use package.* then all the classes and interfaces of this package will be accessible but not subpackages.
- The **import** keyword is used to make the classes and interface of another package accessible to the current package.

2. import package.classname;
- If you import package.classname then only declared class of this package will be accessible.

3. fully qualified name :
- If you use fully qualified name, then only declared class of this package will be accessible. Now there is no need to import.
- But you need to use fully qualified name every time when you are accessing the class or interface.
- It is generally used when two packages have same class name e.g. java.util and java.sql packages contain Date class.

Packages are used for:

- Preventing naming conflicts.
  For example, there can be two classes with name Employee in two packages,
  college.staff.cse.Employee and
  college.staff.ee.Employee

- Making searching/locating and usage of classes, interfaces, enumerations and annotations easier.

- Providing controlled access: protected and default have package level access control. A protected member is accessible by classes in the same package and its subclasses. A default member (without any access specifier) is accessible by classes in the same package only.

- Packages can be considered as data encapsulation (or data-hiding).

Exceptions:

- Exception is an unwanted or unexpected event, which occurs during the execution of a program, i.e. at run time, that disrupts the normal flow of the program's instructions.
- Exceptions can be caught and handled by the program.
- When an exception occurs within a method, it creates an object. This object is called the exception object. It contains information about the exception, such as the name and description of the exception and the state of the program when the exception.

Major reasons why an exception Occurs :

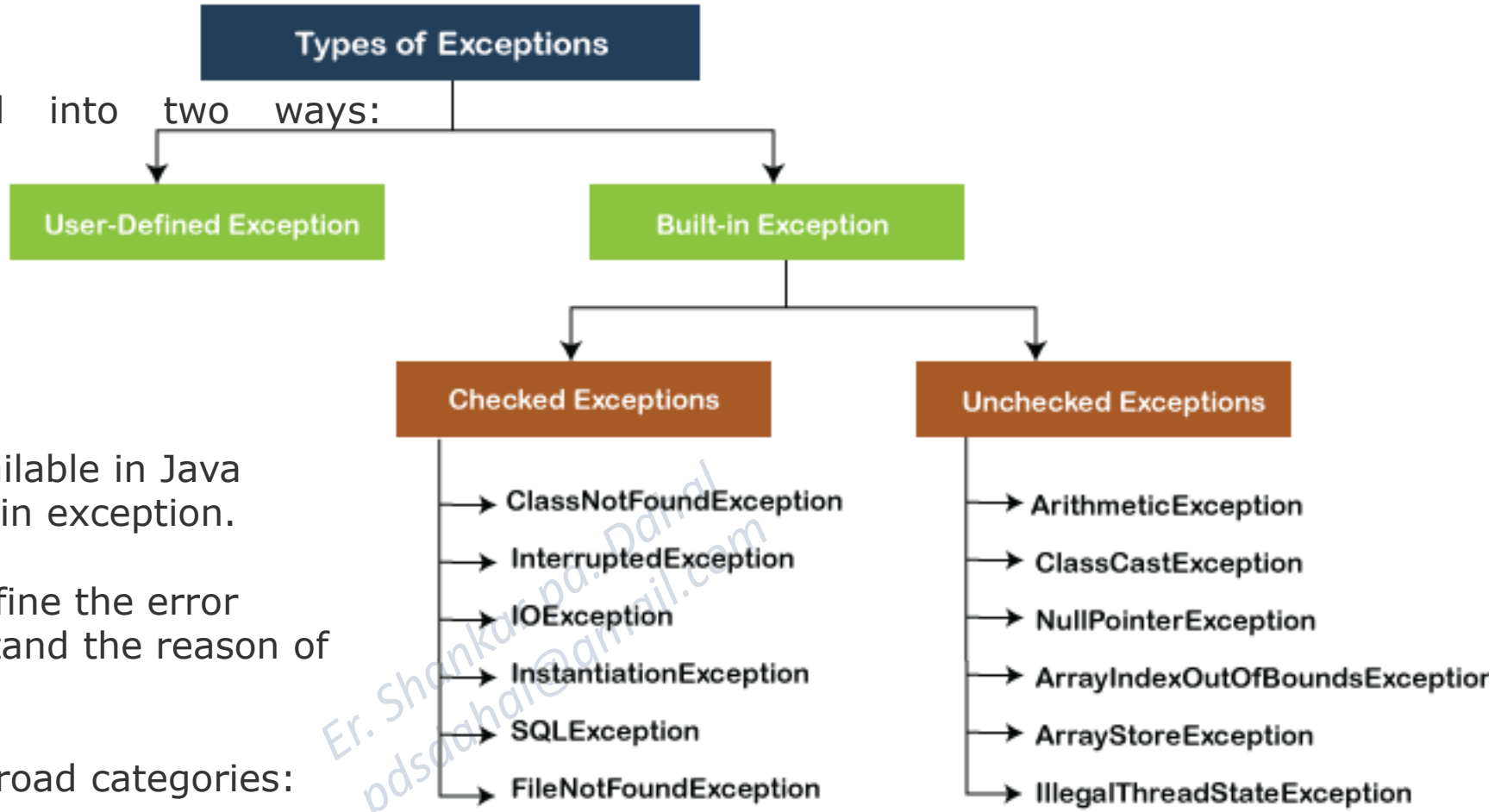
- Invalid user input
- Device failure
- Loss of network connection
- Physical limitations (out of disk memory)
- Code errors
- Opening an unavailable file

Exceptions Types in Java

Exceptions can be categorized into two ways:

1.Built-in Exceptions

2. User-Defined Exceptions



1. Built-in Exceptions

- Exceptions that are already available in Java libraries are referred to as built-in exception.
- These exceptions are able to define the error situation so that we can understand the reason of getting this error.
- It can be categorized into two broad categories:

  a. Checked Exception
  b. Unchecked Exception

1. Built-in Exceptions :

a. Checked Exception :

- Checked exceptions are called compile-time exceptions because these exceptions are checked at compile-time by the compiler.

b. Unchecked Exception :

- The unchecked exceptions are just opposite to the checked exceptions.
- The compiler will not check these exceptions at compile time, but they are checked at runtime.
- In simple words, if a program throws an unchecked exception, and even if we didn't handle or declare it, the program would not give a compilation error.

2. User-Defined Exceptions:

- Sometimes, the built-in exceptions in Java are not able to describe a certain situation. In such cases, users can also create exceptions, which are called user-defined Exceptions.

Basic Scenarios of Exceptions:

1. A scenario where ArithmeticException occurs

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

2. A scenario where NullPointerException occurs

- If we have a null value in any variable, performing any operation on the variable throws a NullPointerException.

3. A scenario where NumberFormatException occurs

- If the formatting of any variable or number is mismatched, it may result into NumberFormatException. Suppose we have a string variable that has characters; converting this variable into digit will cause NumberFormatException.

4. A scenario where ArrayIndexOutOfBoundsException occurs

- When an array exceeds to it's size, the ArrayIndexOutOfBoundsException occurs. there may be other reasons to occur ArrayIndexOutOfBoundsException.

Exception Handling in Java :

- The **Exception Handling in Java** is one of the powerful *mechanism to handle the runtime errors* so that the normal flow of the application can be maintained.

- Java exception handling is managed via five keywords: try, catch, throw, throws and finally.

1. 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.

2. 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.

3. finally :

- The "finally" block is used to execute the necessary code of the program.
- It is executed whether an exception is handled or not.
- Putting cleanup code in a finally block is always a good practice, even when no exceptions are anticipated.

4. throw :

- The "throw" keyword is used to throw an exception.

5. 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.

Java try-catch block :

- The try...catch block in Java is used to handle exceptions and prevents the abnormal termination of the program.
- The try block includes the code that might generate an exception.
- The catch block includes the code that is executed when there occurs an exception inside the try block.

Syntax

```
try {
        // Block of code to try
} catch(Exception e) {
        // Block of code to handle errors
}
```


Java try...finally block

- We can also use the try block along with a finally block.
- finally block is executed whether an exception is handled or not.

Syntax :

```
try {
}finally {
}
```

Java try – catch - finally block :

Syntax:

```
try {
        //statements that may cause an exception
}catch (Exception e) {
        //statements that will execute if exception occurs
}finally {
        //statements that execute whether the exception occurs or not
}
```


Why use Java finally block?

- finally block in Java can be used to put "**cleanup**" code such as closing a file, closing connection, etc.
- The important statements to be printed can be placed in the finally block.

Java Multi-catch Block :

- A try block can be followed by one or more catch blocks.
- Each catch block must contain a different exception handler. So, if you have to perform different tasks at the occurrence of different exceptions, use java multi-catch block.

Note :

- Starting from Java 7.0, it is possible for a single catch block to catch multiple exceptions by separating each with | (pipe symbol) in the catch block.
- Catching multiple exceptions in a single catch block reduces code duplication and increases efficiency. The bytecode generated while compiling this program will be smaller than the program having multiple catch blocks as there is no code redundancy.

Syntax :

```
try {
        //block of code

}catch(ExceptionType1 | ExceptionType2    | ExceptionType3 ex) {

        //block of code
}
```

Java Nested try block :

❖ When a try catch block is present in another try block then it is called the nested try catch block.

Syntax:

```
....
//Main try block
try {
   statement 1;
   statement 2;
   //try-catch block inside another try block
   try {
      statement 3;
      statement 4;
      //try-catch block inside nested try block
      try {
         statement 5;
         statement 6;
      }
      catch(Exception e2) {
         //Exception Message
      }
   }
   catch(Exception e1) {
       //Exception Message
   }

}
//Catch of Main(parent) try block
catch(Exception e3) {
      //Exception Message
}
....
```

Java throw :

- throw keyword is used to create an exception object manually.
- When we use throw, programmer is responsible to create an exception object.
- In case of throw keyword, we can throw only single exception.
- throw keyword is used with the method.

Syntax:

```
throw new exception_class("error message");
```

Java throws :

- throws is a keyword used in the method signature used to declare an exception which might get thrown by the function while executing the code.

- throws keyword is followed by exception class names.

- It indicates the caller method that given exception can occur, so we have to handle it either using try catch block or again declare it by using throws keyword.

- In case of throws keyword, we can declare multiple exceptions.

Syntax:

```
return_type method_name() throws exception_class_name_1, exception_class_name_2, ... {

}
```

Thread :

- A thread is a lightweight sub-process, the smallest unit of processing and also has separate paths of execution.
- All Java programs have at least one thread, known as the main thread, which is created by the Java Virtual Machine (JVM) at the program's start, when the main() method is invoked.
- A single-threaded application has only one Java thread and can handle only one task at a time. To handle multiple tasks in parallel, multi-threading is used.

Multithreading :

- It is a process of executing two or more threads simultaneously to maximum utilization of CPU.
- Multithreaded applications execute two or more threads run concurrently. Hence, it is also known as Concurrency in Java. Each thread runs parallel to each other.
- Multiple threads don't allocate separate memory area. Hence they save memory. Also, context switching between threads takes less time.

Life Cycle of a Thread

- A thread goes through various stages in its life cycle.
- For example, a thread is born, started, runs, and then dies. The following diagram shows the complete life cycle of a thread.

1. New :

- A new thread begins its life cycle in the new state.
- It remains in this state until the program starts the thread.
- It is also referred to as a **born thread**.

2. Runnable :

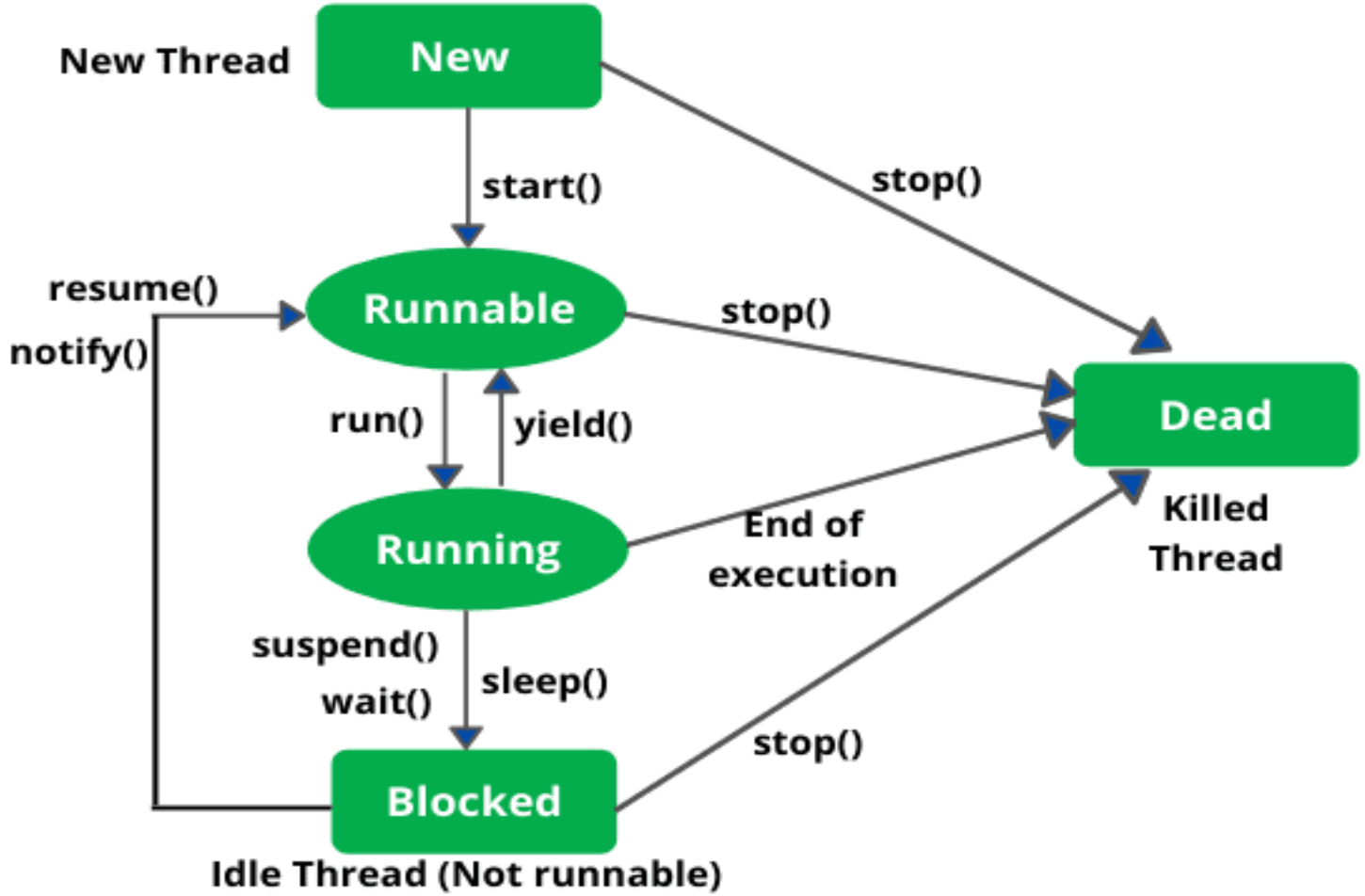
- After a newly born thread is started, the thread becomes runnable.
- A thread in this state is considered to be executing its task.

3. Running :

- When the thread scheduler selects the thread then, that thread would be in a running state.

4. Non-Runnable (Blocked)

The thread is still alive in this state, but currently, it is not eligible to run.



5. Terminated/Dead :

A thread terminates because of either of the following reasons:

- Because it exits normally. This happens when the code of the thread has been entirely executed by the program.
- Because there occurred some unusual erroneous event, like segmentation fault or an unhandled exception.

Create/ Instantiate/ Start New Threads :

We can create Threads in java using two ways, namely :

- By extending Thread Class
- By Implementing a Runnable interface

In both the cases run() method should be implemented.

1. By extending Thread Class

Syntax:

```
class DemoThread extends Thread{
    public void run() {
        //block  of code
    }
}
```

2. By Implementing a Runnable interface

Syntax:

```
class DemoThread implements Runnable{

   public void run() {
        //block of code
    }
}
```

Thread Priorities :

- A component of Java that decides which thread to run or execute and which thread to wait is called a **thread scheduler in Java**.
- In Java, a thread is only chosen by a thread scheduler if it is in the runnable state. However, if there is more than one thread in the runnable state, it is up to the thread scheduler to pick one of the threads and ignore the other ones.
- There are some criteria that decide which thread will execute first. There are two factors for scheduling a thread i.e. **Priority** and **Time of arrival**.

Priority :

- Priority of each thread lies between 1 to 10. If a thread has a higher priority, it means that thread has got a better chance of getting picked up by the thread scheduler.

- In most cases, the thread scheduler schedules the threads according to their priority (known as preemptive scheduling). But it is not guaranteed because it depends on JVM specification that which scheduling it chooses.
- Java programmer can also assign the priorities of a thread explicitly in a Java program.

- The default priority is set to 5 as excepted.
- Minimum priority is set to 1.
- Maximum priority is set to 10.

Here 3 constants are defined in it namely as follows:

- public static int NORM_PRIORITY
- public static int MIN_PRIORITY
- public static int MAX_PRIORITY

Setter & Getter Method of Thread Priority :

public final int getPriority():

- The java.lang.Thread.getPriority() method returns the priority of the given thread.

public final void setPriority(int newPriority):

- The java.lang.Thread.setPriority() method updates or assign the priority of the thread to newPriority.
- The method throws IllegalArgumentException if the value newPriority goes out of the range, which is 1 (minimum) to 10 (maximum).

Synchronization :

- Multi-threaded programs may often come to a situation where multiple threads try to access the same resources and finally produce erroneous and unforeseen results. So, it needs to be made sure by some synchronization method that only one thread can access the resource at a given point in time.

- In General, when two or more threads need access to a shared resource, they need some way to ensure that the resource will be used by only one thread at a time. The process by which this is achieved is called **synchronization**.

Why use Synchronization?

The synchronization is mainly used to :

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

Types:
There are two types of thread synchronization :

1. Mutual Exclusive

2. Cooperation (Inter-thread communication in java)

Mutual Exclusive :

- Mutual Exclusive helps keep threads from interfering with one another while sharing data.
- It can be achieved by using the following three ways:

a. By Using Synchronized Method
b. By Using Synchronized Block
c. By Using Static Synchronization

a. By Using Synchronized Method :

1. If you declare any method as synchronized, it is known as synchronized method.
2. Synchronized method is used to lock an object for any shared resource.

3. When a thread invokes a synchronized method, it automatically acquires the lock for that object and releases it when the thread completes its task.

4. If the object lock is not available, the calling thread is blocked, and it has to wait until the lock becomes available.

- Once a thread completes the execution of code inside the synchronized method, it releases object lock and allows other thread waiting for this lock to proceed.

- That is once a thread completes its work using synchronized method, it will hand over to the next thread that is ready to use the same resource.

Syntax :

```
synchronized Access_Modifiers Return_Type MehthodName(Parameters) {

}
```

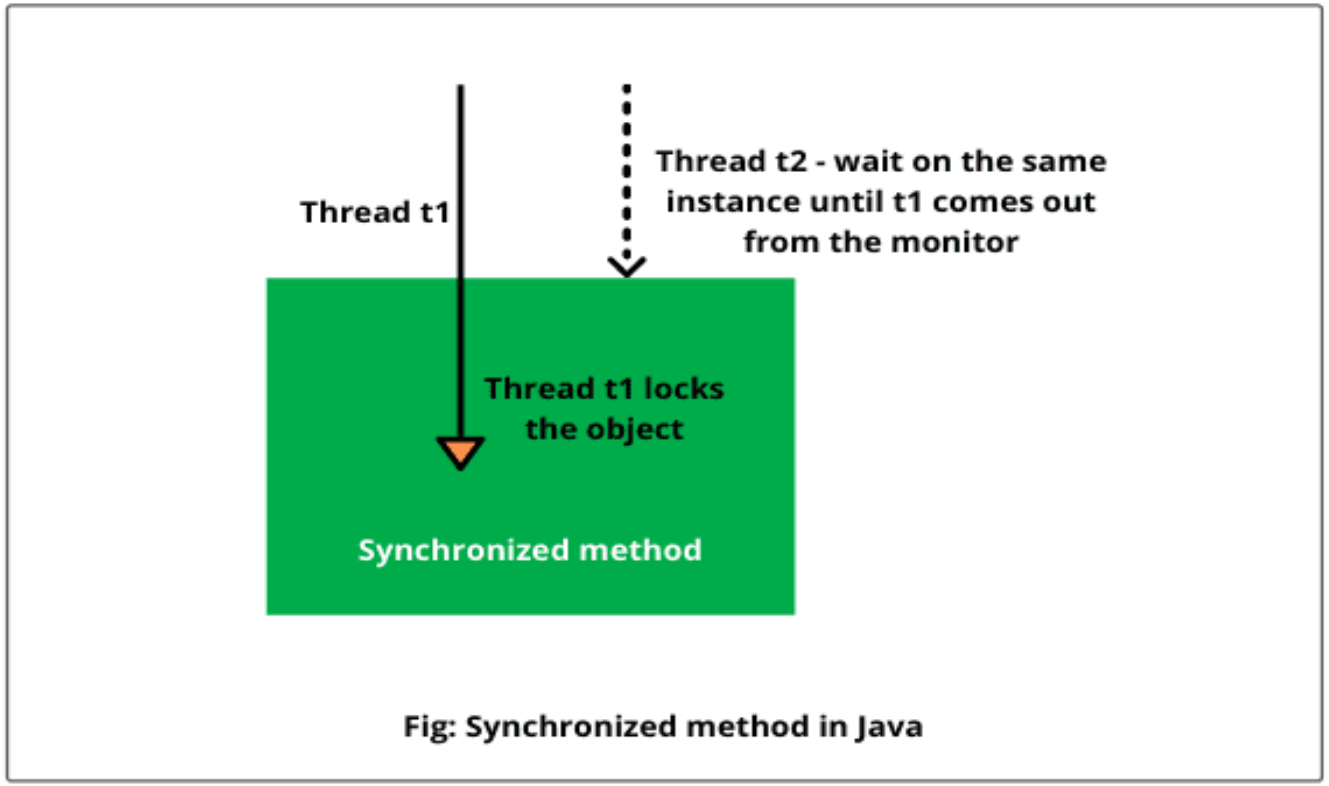


Fig: Synchronized method in Java

b. By Using Synchronized Block

- Synchronized block can be used to perform synchronization on any specific resource of the method.
- Suppose we have 100 lines of code in our method, but we want to synchronize only 10 lines, in such cases, we can use synchronized block.
- If we put all the codes of the method in the synchronized block, it will work same as the synchronized method.

- **Syntax :**

```
synchronized (object reference expression) {

}
```

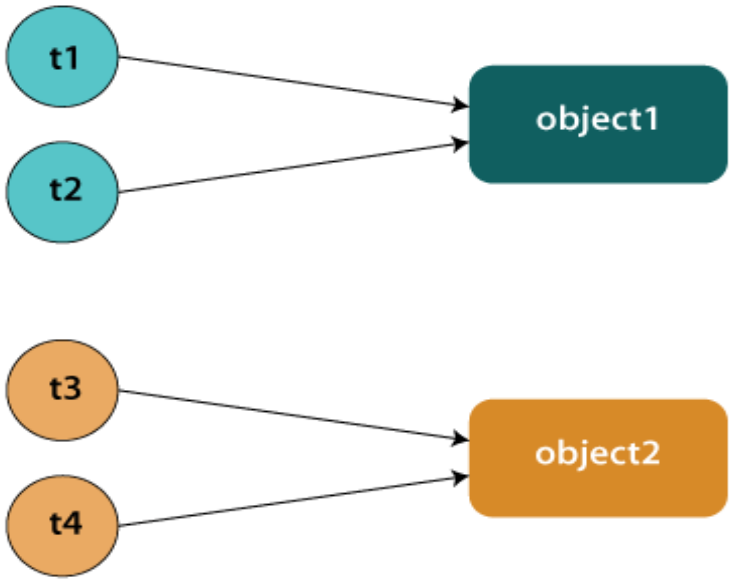
c. By Using Static Synchronization

- Suppose there are two objects of a shared class named object1 and object2.
- In case of synchronized method and synchronized block there cannot be interference between **t1 and t2** or **t3 and t4** because t1 and t2 both refers to a common object that have a single lock. But there can be interference between t1 and t3 or t2 and t4 because t1 acquires another lock and t3 acquires another lock.
- We don't want interference between t1 and t3 or t2 and t4. Static synchronization solves this problem.
- If you make any static method as synchronized, the lock will be on the class not on object.

Syntax :

```
static synchronized Access_Modifiers Return_Type MehthodName(Parameters) {

}
```

2. Co-operation (Inter-thread communication in java)

- Inter-thread communication is a mechanism in which a thread releases the lock and enter into paused state and another thread acquires the lock and continue to executed.

- It is implemented by following methods of **Object class**:

1. wait() :

- The wait() method causes current thread to release the lock and wait until either another thread invokes the notify() method or the notifyAll() method for this object, or a specified amount of time has elapsed.

2. notify() :

- This method is used to wake up a single thread and releases the object lock.

3. notifyAll() :

- This method is used to wake up all threads that are in waiting state.

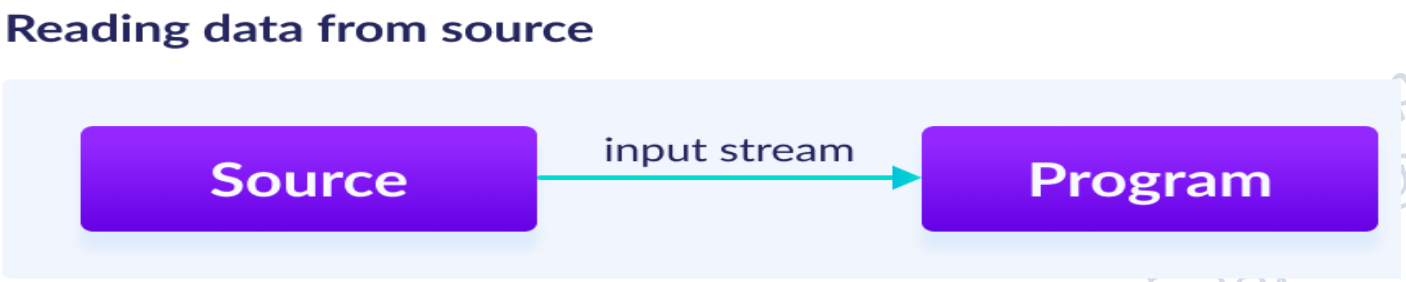
Java.io Package in Java :

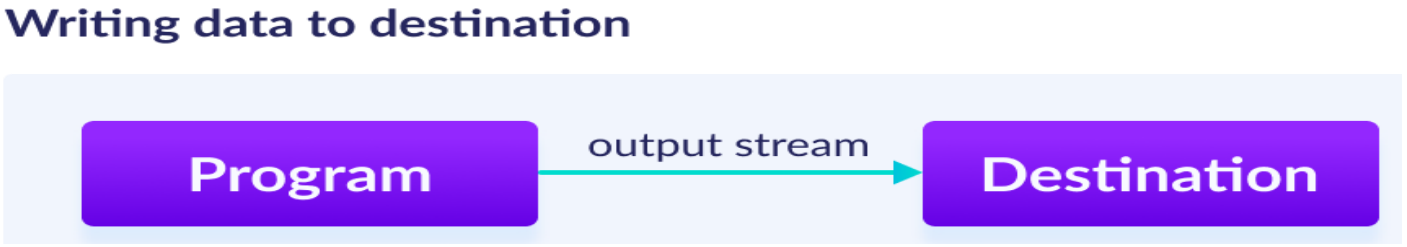
❖ This package provides for system input and output through data streams, serialization and the file system.

Streams:

❖ A stream can be defined as a sequence of data. There are two kinds of Streams :

1. **InPutStream** – The **InputStream** is used to read data from a source.
2. **OutPutStream** – The **OutputStream** is used for writing data to a destination.





Types of Streams

It can be classified into:

Byte Stream :

- Byte stream is used to read and write a single byte (8 bits) of data.
- All byte stream classes are derived from base abstract classes called InputStream and OutputStream.

Character Stream :

- Character stream is used to read and write a single character of data.
- All the character stream classes are derived from base abstract classes Reader and Writer.

Predefined Streams :

In Java, 3 streams are created for us automatically. All these streams are attached with the console.

1) **System.out**: standard output stream
2) **System.in**: standard input stream
3) **System.err**: standard error stream

Byte Stream classes:

- Though there are many classes related to byte streams but the most frequently used classes are, **FileInputStream** and **FileOutputStream**.

Stream Class	Meaning
BufferedInputStream	Buffered input stream
BufferedOutputStream	Buffered output stream
ByteArrayInputStream	Input stream that reads from a byte array
ByteArrayOutputStream	Output stream that writes to a byte array
DataInputStream	An input stream that contains methods for reading the Java standard data types
DataOutputStream	An output stream that contains methods for writing the Java standard data types
FileInputStream	Input stream that reads from a file
FileOutputStream	Output stream that writes to a file
FilterInputStream	Implements **InputStream**
FilterOutputStream	Implements **OutputStream**
InputStream	Abstract class that describes stream input
ObjectInputStream	Input stream for objects
ObjectOutputStream	Output stream for objects
OutputStream	Abstract class that describes stream output
PipedInputStream	Input pipe
PipedOutputStream	Output pipe
PrintStream	Output stream that contains **print( )** and **println( )**
PushbackInputStream	Input stream that supports one-byte "unget," which returns a byte to the input stream
RandomAccessFile	Supports random access file I/O
SequenceInputStream	Input stream that is a combination of two or more input streams that will be read sequentially, one after the other

- You can write byte-oriented as well as character-oriented data through FileOutputStream class.

Note : *for character-oriented data, it is preferred to use* ***FileWriter*** *than FileOutputStream****.***

- In Java, we can copy the contents of one file to another file. This can be done by the **FileInputStream** and **FileOutputStream** classes.

Character Stream classes:

- Though there are many classes related to character streams, but the most frequently used classes are, **FileReader** and **FileWriter**.

Stream Class	Meaning
BufferedReader	Buffered input character stream
BufferedWriter	Buffered output character stream
CharArrayReader	Input stream that reads from a character array
CharArrayWriter	Output stream that writes to a character array
FileReader	Input stream that reads from a file
FileWriter	Output stream that writes to a file
FilterReader	Filtered reader
FilterWriter	Filtered writer
InputStreamReader	Input stream that translates bytes to characters
LineNumberReader	Input stream that counts lines
OutputStreamWriter	Output stream that translates characters to bytes
PipedReader	Input pipe
PipedWriter	Output pipe
PrintWriter	Output stream that contains **print( )** and **println( )**
PushbackReader	Input stream that allows characters to be returned to the input stream
Reader	Abstract class that describes character stream input
StringReader	Input stream that reads from a string
StringWriter	Output stream that writes to a string
Writer	Abstract class that describes character stream output

Java Files and Directories :

- The File class from the java.io package, allows us to work with files.
- The File class is an abstract representation of file and directory pathname. A pathname can be either absolute or relative.
- The File class have several methods for working with directories and files such as creating new directories or files, deleting and renaming directories or files, listing the contents of a directory etc.

- To use the File class, create an object of the class, and specify the filename or directory name:

```
File myObj = new File("filename.txt"); // Specify the filename
```

- A directory is a collection of files and subdirectories. A directory inside a directory is known as subdirectory.

The File class has many useful methods for getting information about files. Below are some:

Method	Type	Description
canRead()	Boolean	Tests whether the file is readable or not
canWrite()	Boolean	Tests whether the file is writable or not
createNewFile()	Boolean	Creates an empty file
delete()	Boolean	Deletes a file
exists()	Boolean	Tests whether the file exists
getName()	String	Returns the name of the file
getAbsolutePath()	String	Returns the absolute pathname of the file
length()	Long	Returns the size of the file in bytes
list()	String[]	Returns an array of the files in the directory
mkdir()	Boolean	Creates a directory

Reading and Writing Files :

- Java provides a number of classes and methods that allow you to read and write files. In Java, all files are byte-oriented, and Java provides methods to read and write bytes from and to a file.

FileInputStream(String *fileName*) throws **FileNotFoundException** :

- The FileInputStream class of the java.io package can be used to read data (in bytes) from files.
- It extends the InputStream abstract class.

Syntax:

```
FileInputStream fileInputStream = new FileInputStream(String fileNamePath);
```


FileOutputStream(String *fileName*) throws **FileNotFoundException**

- The FileOutputStream class of the java.io package can be used to write data (in bytes) to the files.
- It extends the OutputStream abstract class.

Syntax:

```
FileOutputStream output = new FileOutputStream(String path, boolean value);
```

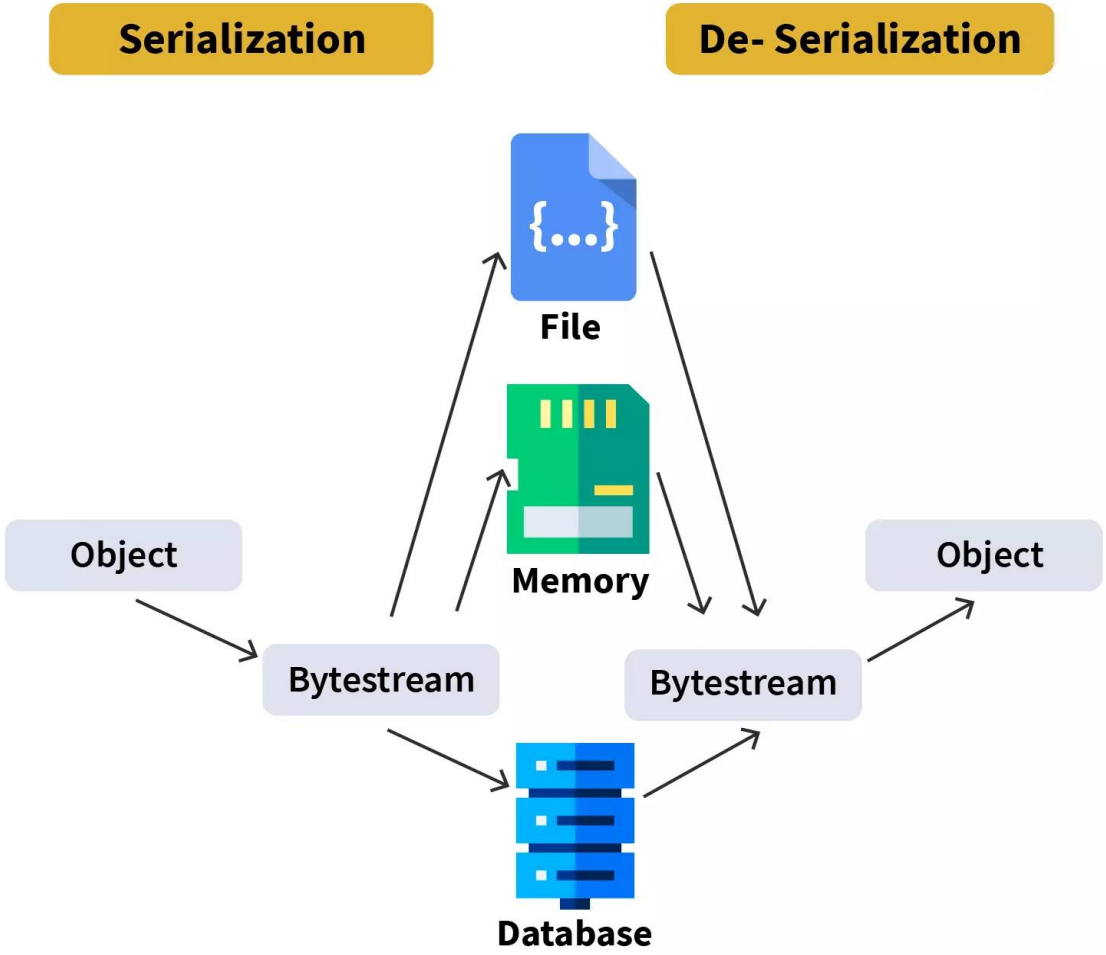
Serialization Interface:

- The Serializable interface is present in java.io package.
- 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.

Why Do We Need Serialization in Java?

- Serialization mechanism is usually used when there is a need to send your data (objects) over the network or to store in files. Now the hardware components like network infrastructure, hard disk etc understands only bytes and bits, not the java objects.

- So, Serialization is used in this case which translates Java object's state to byte-stream to send it over the network or save it in file.



How Do We Serialize An Object?

- Serialization in java can be implemented using java.io.Serializable interface.

- For serializing the object, we will be using the **writeObject**() method of **ObjectOutputStream** class.

```
FileOutputStream fileOutputStream = new FileOutputStream(”fileName");
ObjectOutputStream objectOutputStream = new ObjectOutputStream(fileOutputStream);
```

- For deserializing the object, we will be using the **readObject**() method of **ObjectInputStream** class.

```
FileInputStream fileInputStream = new FileInputStream(”fileName");
ObjectInputStream objectOutputStream = new ObjectInputStream(fileInputStream);
```


SerialVersionUID in Java

- The serialization process at runtime associates an id with each Serializable class which is known as SerialVersionUID. It is used to verify the sender and receiver of the serialized object. The sender and receiver must be the same. To verify it, SerialVersionUID is used. The sender and receiver must have the same SerialVersionUID, otherwise, InvalidClassException will be thrown when you deserialize the object.

- Syntax:

```
private static final long serialVersionUID = 1L;
```