

Introduction to Java

Java is a **class-based, object-oriented programming language** that is designed to have as few implementation dependencies as possible. It is intended to let application developers **write once, and run anywhere (WORA)**, meaning that compiled **Java code** can run on all platforms that support **Java** without the need for recompilation. **Java** was first released in 1995 and is widely used for developing applications for desktop, web, and mobile devices. **Java** is known for its **simplicity, robustness, and security features**, making it a popular choice for **enterprise-level applications**.

Java was developed by **James Gosling** at **Sun Microsystems Inc** in May 1995 and later acquired by **Oracle Corporation**. It is a **simple programming language**. **Java** makes writing, compiling, and debugging programming easy. It helps to create reusable code and modular programs. **Java** is a **class-based, object-oriented programming language** and is designed to have as few implementation dependencies as possible. A **general-purpose programming language** made for developers to **write once run anywhere** that is compiled **Java code** can run on all platforms that support **Java**. **Java applications** are compiled to **byte code** that can run on any **Java Virtual Machine**

History of Java

The history of Java is very interesting. Java was originally designed for interactive television, but it was too advanced technology for the digital cable television industry at the time. The history of Java starts with the Green Team. Java team members (also known as Green Team), initiated this project to develop a language for digital devices such as set-top boxes, televisions, etc. However, it was best suited for internet programming. Later, Java technology was incorporated by Netscape.

The principles for creating Java programming were "Simple, Robust, Portable, Platform-independent, Secured, High Performance, Multithreaded, Architecture Neutral, Object-Oriented, Interpreted, and Dynamic". **Java** was developed by James Gosling, who is known as the father of Java, in 1995. James Gosling and his team members started the project in the early '90s.

Currently, Java is used in internet programming, mobile devices, games, e-business solutions, etc. Following are given significant points that describe the history of Java.

1) **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.

3) Firstly, it was called "**Greentalk**" by James Gosling, and the file extension was .gt.

4) After that, it was called **Oak** and was developed as a part of the Green project.

Why Java was named as "Oak"?



5) **Why Oak?** Oak is a symbol of strength and chosen as a national tree of many countries like the U.S.A., France, Germany, Romania, etc.

6) In 1995, Oak was renamed as "**Java**" because it was already a trademark by Oak Technologies.

Why Java Programming named "Java"?

7) Why had they chose the name Java for Java language? 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.

According to James Gosling, "Java was one of the top choices along with **Silk**". Since Java was so unique, most of the team members preferred Java than other names.

8) 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.

9) Notice that Java is just a name, not an acronym.

10) Initially developed by James Gosling at [Sun Microsystems](#) (which is now a subsidiary of Oracle Corporation) and released in 1995.

11) In 1995, Time magazine called **Java one of the Ten Best Products of 1995**.

12) JDK 1.0 was released on January 23, 1996. After the first release of Java, there have been many additional features added to the language. Now Java is being used in Windows applications, Web applications, enterprise applications, mobile applications, cards, etc. Each new version adds new features in Java.

Java Version History

Many java versions have been released till now. The current stable release of Java is Java SE 10.

1. JDK Alpha and Beta (1995)
2. JDK 1.0 (23rd Jan 1996)
3. JDK 1.1 (19th Feb 1997)
4. J2SE 1.2 (8th Dec 1998)
5. J2SE 1.3 (8th May 2000)
6. J2SE 1.4 (6th Feb 2002)
7. J2SE 5.0 (30th Sep 2004)
8. Java SE 6 (11th Dec 2006)
9. Java SE 7 (28th July 2011)
10. Java SE 8 (18th Mar 2014)
11. Java SE 9 (21st Sep 2017)
12. Java SE 10 (20th Mar 2018)
13. Java SE 11 (September 2018)
14. Java SE 12 (March 2019)
15. Java SE 13 (September 2019)

- 16. Java SE 14 (Mar 2020)
- 17. Java SE 15 (September 2020)
- 18. Java SE 16 (Mar 2021)
- 19. Java SE 17 (September 2021)
- 20. Java SE 18 (to be released by March 2022)

Since Java SE 8 release, the Oracle corporation follows a pattern in which every even version is release in March month and an odd version released in September month.

C++ vs Java

There are many differences and similarities between the **C++ programming** language and **Java**. A list of top differences between C++ and Java are given below:

Comparison Index	C++	Java
Platform-independent	C++ is platform-dependent.	Java is platform-independent.
Mainly used for	C++ is mainly used for system programming.	Java is mainly used for application programming. It is widely used in Windows-based, web-based, enterprise, and mobile applications.
Design Goal	C++ was designed for systems and applications programming. It was an extension of the C programming language .	Java was designed and created as an interpreter for printing systems but later extended as a support network computing. It was designed to be easy to use and accessible to a broader audience.
Goto	C++ supports the goto statement.	Java doesn't support the goto statement.

Operator Overloading	C++ supports operator overloading .	Java doesn't support operator overloading.
Pointers	C++ supports pointers . You can write a pointer program in C++.	Java supports pointer internally. However, you can't write the pointer program in java. It means java has restricted pointer support in java.
Compiler and Interpreter	C++ uses compiler only. C++ is compiled and run using the compiler which converts source code into machine code so, C++ is platform dependent.	Java uses both compiler and interpreter. Java source code is converted into bytecode at compilation time. The interpreter executes this bytecode at runtime and produces output. Java is interpreted that is why it is platform-independent.
Call by Value and Call by reference	C++ supports both call by value and call by reference.	Java supports call by value only. There is no call by reference in java.
Structure and Union	C++ supports structures and unions.	Java doesn't support structures and unions.
Thread Support	C++ doesn't have built-in support for threads. It relies on third-party libraries for thread support.	Java has built-in thread support.
Documentation comment	C++ doesn't support documentation comments.	Java supports documentation comment (/** ... */) to create documentation for java source code.
Virtual Keyword	C++ supports virtual keyword so that we can decide whether or not to override a function.	Java has no virtual keyword. We can override all non-static methods by default. In other words, non-static methods are virtual by default.

unsigned right shift >>>	C++ doesn't support >>> operator.	Java supports unsigned right shift >>> operator that fills zero at the top for the negative numbers. For positive numbers, it works same like >> operator.
Inheritance Tree	C++ always creates a new inheritance tree.	Java always uses a single inheritance tree because all classes are the child of the Object class in Java. The Object class is the root of the inheritance tree in java.
Hardware	C++ is nearer to hardware.	Java is not so interactive with hardware.
Object-oriented	C++ is an object-oriented language. However, in the C language, a single root hierarchy is not possible.	Java is also an object-oriented language. However, everything (except fundamental types) is an object in Java. It is a single root hierarchy as everything gets derived from java.lang.Object.

The Java Development Kit (JDK) includes many tools for software development, including:

- **Basic tools:** These include:
 - **appletviewer:** Runs and debugs applets without a web browser
 - **jar:** Creates and manages Java Archive (JAR) files
 - **javac:** Compiles Java source code files into bytecode files
 - **javadoc:** Generates API documentation from Java source code
 - **jdb:** Debugs Java programs
- **Troubleshooting tools:** These include:
 - **jinfo:** Prints configuration information for a process, core file, or remote debug server

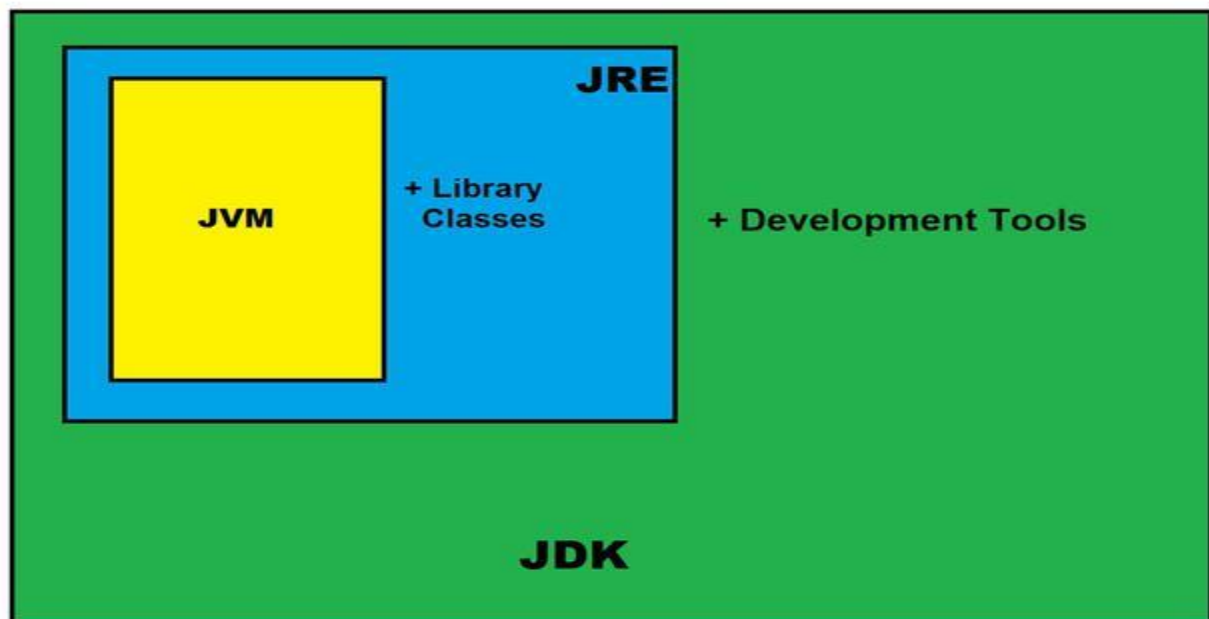
- **jmap**: Prints shared object memory maps or heap memory details for a process, core file, or remote debug server
 - **jsadebugd**: Attaches to a process or core file and acts as a debug server
 - **jstack**: Prints a stack trace of threads for a process, core file, or remote debug server
 - **JDK Mission Control (JMC)**: Manages, monitors, profiles, and troubleshoots Java applications
 - **VisualVM**: Integrates several command-line JDK tools and performance and memory profiling capabilities
 - **wsimport**: Generates portable JAX-WS artifacts for invoking a web service
 - **xjc**: Accepts an XML schema and generates Java classes
- You can install the JDK from the Oracle's official Download Page. After installation, you can verify that the JDK is installed correctly by running the `java -version` command in the command prompt.

The JDK is a development environment for building applications, applets, and components using the Java programming language. The JDK includes tools useful for developing and testing programs written in the Java programming language and running on the Java platform.

The Java Development Kit (JDK) is a cross-platformed software development environment that offers a collection of tools and libraries necessary for developing Java-based software applications and applets. It is a core package used in Java, along with the **JVM (Java Virtual Machine)** and the JRE (Java Runtime Environment).

Beginners often get confused with JRE and JDK, if you are only interested in running Java programs on your machine then you can easily do it using Java Runtime Environment. However, if you would like to develop a Java-based software application then along with JRE you may need some additional necessary tools, which is called JDK.

JDK=JRE+Development Tools



JAVA Development Kit (JDK)

The Java Development Kit is an implementation of one of the Java Platform:

- [Standard Edition](#) (Java SE),
- [Java Enterprise Edition](#) (Java EE),
- [Micro Edition](#) (Java ME),

Contents of JDK

The JDK has a private Java Virtual Machine (JVM) and a few other resources necessary for the development of a Java Application.

JDK contains:

- Java Runtime Environment (JRE),
- An interpreter/loader (Java),
- A compiler (javac),
- An archiver (jar) and many more.

The Java Runtime Environment in JDK is usually called Private Runtime because it is separated from the regular JRE and has extra content. The Private Runtime in JDK contains a JVM and all the class libraries present in the production environment, as well as additional libraries useful to developers, e.g, internationalization libraries and the IDL libraries.

Most Popular JDKs:

- **Oracle JDK:** the most popular JDK and the main distributor of Java11,
- **OpenJDK:** Ready for use: JDK 15, JDK 14, and JMC,
- **Azul Systems Zing:** efficient and low latency JDK for Linux os,

- **Azul Systems:** based Zulu brand for Linux, Windows, Mac OS X,
- **IBM J9 JDK:** for AIX, Linux, Windows, and many other OS,
- **Amazon Corretto:** the newest option with the no-cost build of OpenJDK and long-term support.

Set-Up:

Setting up JDK in your development environment is super easy, just follow the below simple steps.

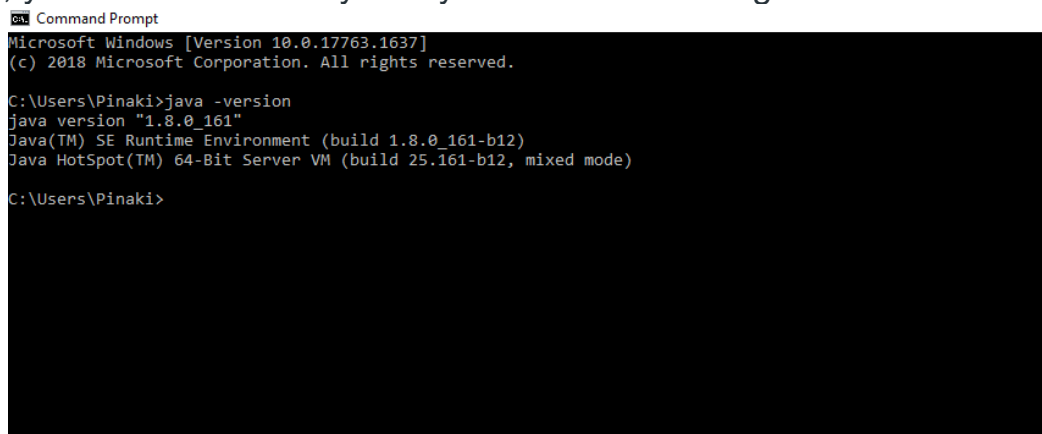
Installation of JDK

- Go to this Oracle's official Download Page through this [link](#)
- Select the latest JDK version and click Download and add it to your classpath.
- Just check the JDK software is installed or not on your computer at the correct location, for example, at C:\Program Files\Java\jdk11.0.9.

Set JAVA_HOME for Windows:

- Right-click My Computer and select Properties.
- Go to the Advanced tab and select Environment Variables, and then edit JAVA_HOME to point to the exact location where your JDK software is stored, for example, C:\Program Files\Java\jdk11.0.9 is the default location in windows.

Java maintains backward compatibility, so don't worry just download the latest release and you will get all the old and many new features. After Installing the JDK and JRE adds the java command to your command line. You can verify this through the command prompt by the **java -version** command. In some cases, you need to restart your system after installing the JDK.



```

C:\Users\Pinaki>java -version
java version "1.8.0_161"
Java(TM) SE Runtime Environment (build 1.8.0_161-b12)
Java HotSpot(TM) 64-Bit Server VM (build 25.161-b12, mixed mode)
C:\Users\Pinaki>

```

JDK Version

Compile and Run Java Code using JDK:

You can use the JDK compiler to convert your Java text file into an executable program. Your Java text segment is converted into **bytecode** after compilation which carries the **.class** extension.

First, create a Java text file and save it using a name. Here we are saving the file as Hello.java.

Java

```
class Hello{  
    public static void main (String[] args) {  
        System.out.println("Hello Geek!");  
    }  
}
```

After that just simply use the **javac** command, which is used for the compilation purpose in Java. Please don't forget to provide the full path of your java text file to the command line else you will get an error as "The system cannot find the path specified",

Your command should be similar to the given below example where Hello is the file name and the full path to the file is specified before the file name. The path and javac.exe should be inside the quotes.

"C:\Program Files\Java\jdk-11.0.9\bin\javac.exe" Hello.java

You can notice now that the *Hello.class* file is being created in the same directory as Hello.java. Now you can run your code by simply using the **java** **Hello** command, which will give you the desired result according to your code. Please remember that you don't have to include the .class to run your code.

C:\Users\Pinaki\Documents>java Hello

(Output:) Hello Geek!

The Jar component:

JDK contains many useful tools and among them, the most popular after javac is the jar tool. The jar file is nothing but a full pack of Java classes. After creating the .class files, you can put them together in a .jar, which compresses and structures them in a predictable fashion. Now, let's convert our Hello.class to a jar file.

Before proceeding, please note that you should be in the same directory where the Hello.java file was saved. Now type the command given below in the command line.

Important Components of JDK

Below there is a comprehensive list of mostly used components of Jdk which are very useful during the development of a java application.

Component	Use
javac	Java compiler converts source code into Java bytecode
java	The loader of the java apps.
javap	Class file disassembler,
javadoc	Documentation generator,
jar	Java Archiver helps manage JAR files.

Java class file

A **Java class file** is a [file](#) (with the `.class` [filename extension](#)) containing [Java bytecode](#) that can be executed on the [Java Virtual Machine \(JVM\)](#). A Java class file is usually produced by a [Java compiler](#) from [Java programming language source files](#) (`.java` files) containing Java [classes](#) (alternatively, other [JVM languages](#) can also be used to create class files). If a source file has more than one class, each class is compiled into a separate class file. Thus, it is called a `.class` file because it contains the bytecode for a single class.

JVMs are available for many [platforms](#), and a class file compiled on one platform will execute on a JVM of another platform. This makes Java applications [platform-independent](#).

History

[\[edit\]](#)

On 11 December 2006, the class file format was modified under [Java Specification Request](#) (JSR) 202.

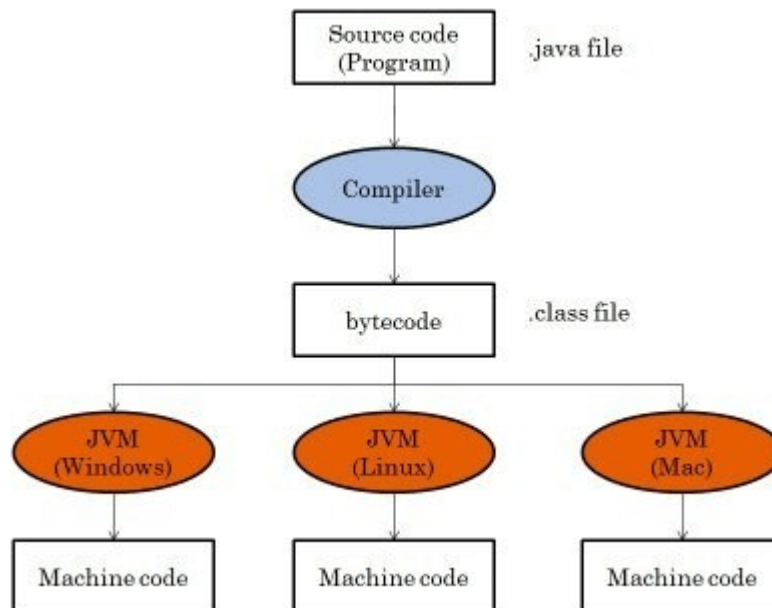
bytecode in Java

Java bytecode is the instruction set of the Java virtual machine (JVM), the language to which Java and other JVM-compatible source code is compiled. Each instruction is represented by a single byte, hence the name bytecode, making it a compact form of data. Bytecode is essentially the machine level language which runs on the Java Virtual Machine.

Whenever a class is loaded, it gets a stream of bytecode per method of the class. Whenever that method is called during the execution of a program, the bytecode for that method gets invoked. Javac not only compiles the program but also generates the bytecode for the program. Thus, we have realized that the bytecode implementation makes Java a **platform-independent** language.

How does it works?

When we write a program in Java, firstly, the compiler compiles that program and a bytecode is generated for that piece of code. When we wish to run this .class file on any other platform, we can do so. After the first compilation, the bytecode generated is now run by the Java Virtual Machine and not the processor in consideration. This essentially means that we only need to have basic java installation on any platforms that we want to run our code on. Resources required to run the bytecode are made available by the Java Virtual Machine, which calls the processor to allocate the required resources. JVM's are stack-based so they use stack implementation to read the codes.



JVM (Java Virtual Machine)

JVM (Java Virtual Machine) is an abstract machine. It is a specification that provides runtime environment in which java bytecode can be executed.

JVMs are available for many hardware and software platforms (i.e. JVM is platform dependent).

What is JVM

It is:

1. **A specification** where working of Java Virtual Machine is specified. But implementation provider is independent to choose the algorithm. Its implementation has been provided by Oracle and other companies.
2. **An implementation** Its implementation is known as JRE (Java Runtime Environment).
3. **Runtime Instance** Whenever you write java command on the command prompt to run the java class, an instance of JVM is created.

What it does

The JVM performs following operation:

- Loads code
- Verifies code
- Executes code
- Provides runtime environment

JVM provides definitions for the:

- Memory area
- Class file format
- Register set
- Garbage-collected heap
- Fatal error reporting etc.

Java Identifiers

All Java **variables** must be **identified** with **unique names**.

These unique names are called **identifiers**.

In Java, identifiers are names given to various elements in a program, such as variables, methods, classes, interfaces, packages, and objects. They are used to identify and differentiate between different elements within the code

Identifiers can be short names (like x and y) or more descriptive names (age, sum, totalVolume).

Note: It is recommended to use descriptive names in order to create understandable and maintainable code:

```
int minutesPerHour = 60;  
  
int m = 60;
```

The general rules for naming variables are:

- Names can contain letters, digits, underscores, and dollar signs
- Names must begin with a letter
- Names should start with a lowercase letter, and cannot contain whitespace
- Names can also begin with \$ and _
- Names are case-sensitive ("myVar" and "myvar" are different variables)
- Reserved words (like Java keywords, such as **int** or **boolean**) cannot be used as names

Java If-else Statement

The **Java if** statement is used to test the condition. It checks **boolean** condition: *true* or *false*. There are various types of if statement in Java.

- if statement
- if-else statement
- if-else-if ladder

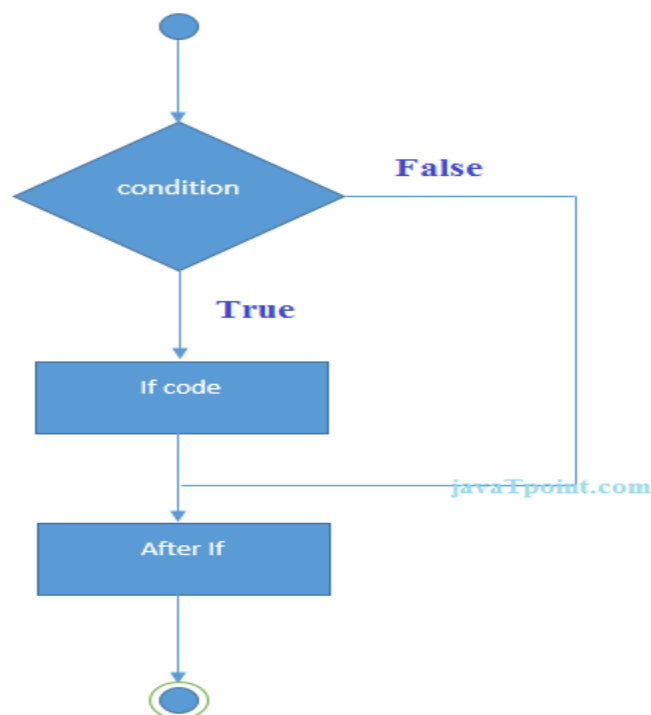
- nested if statement

Java if Statement

The Java if statement tests the condition. It executes the *if block* if condition is true.

Syntax:

```
if(condition){  
    //code to be executed  
}
```



Example:

```
//Java Program to demonstate the use of if statement.  
public class IfExample {  
    public static void main(String[] args) {  
        //defining an 'age' variable
```

```
int age=20;  
//checking the age  
if(age>18){  
    System.out.print("Age is greater than 18");  
}  
}  
}
```

Output:

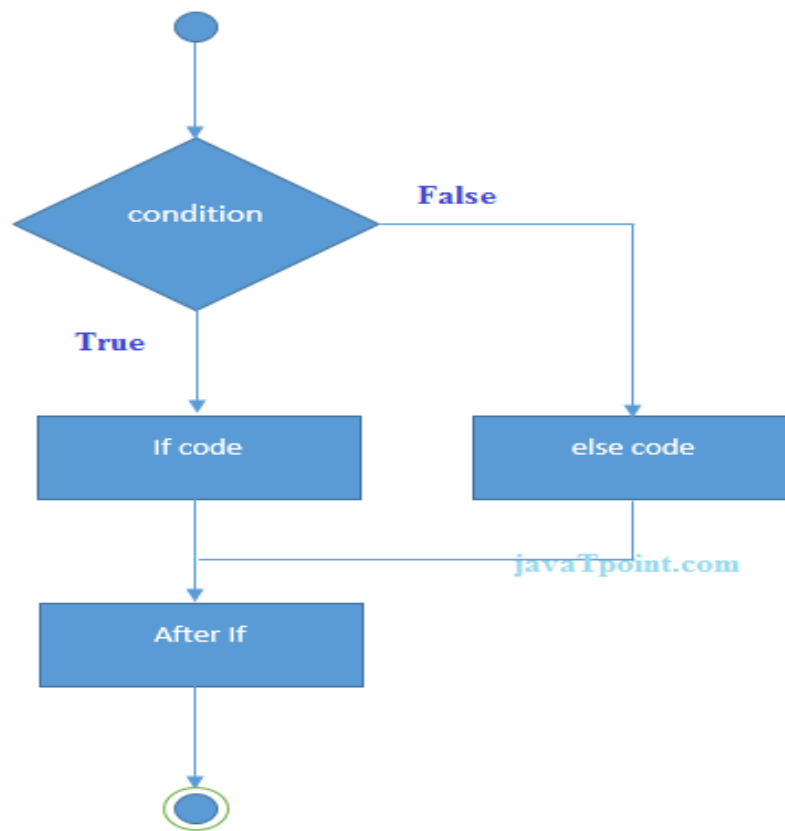
```
Age is greater than 18
```

if-else Statement

The Java if-else statement also tests the condition. It executes the *if block* if condition is true otherwise *else block* is executed.

Syntax:

```
if(condition){  
    //code if condition is true  
}  
else{  
    //code if condition is false  
}
```

Example:

//A Java Program to demonstrate the use of if-else statement.

//It is a program of odd and even number.

```
public class IfElseExample {  
    public static void main(String[] args) {  
        //defining a variable  
        int number=13;  
        //Check if the number is divisible by 2 or not  
        if(number%2==0){  
            System.out.println("even number");  
        }else{  
            System.out.println("odd number");  
        }  
    }  
}
```

Output:

odd number

Leap Year Example:

A year is leap, if it is divisible by 4 and 400. But, not by 100.

```
public class LeapYearExample {  
    public static void main(String[] args) {  
        int year=2020;  
        if(((year % 4 ==0) && (year % 100 !=0)) || (year % 400==0)){  
            System.out.println("LEAP YEAR");  
        }  
        else{  
            System.out.println("COMMON YEAR");  
        }  
    }  
}
```

Output:

LEAP YEAR

Using Ternary Operator

We can also use ternary operator (?:) to perform the task of if...else statement. It is a shorthand way to check the condition. If the condition is true, the result of ? is returned. But, if the condition is false, the result of : is returned.

Example:

```
public class IfElseTernaryExample {  
    public static void main(String[] args) {  
        int number=13;
```

```
//Using ternary operator
String output=(number%2==0?"even number":"odd number");
System.out.println(output);
}
}
```

Output:

odd number

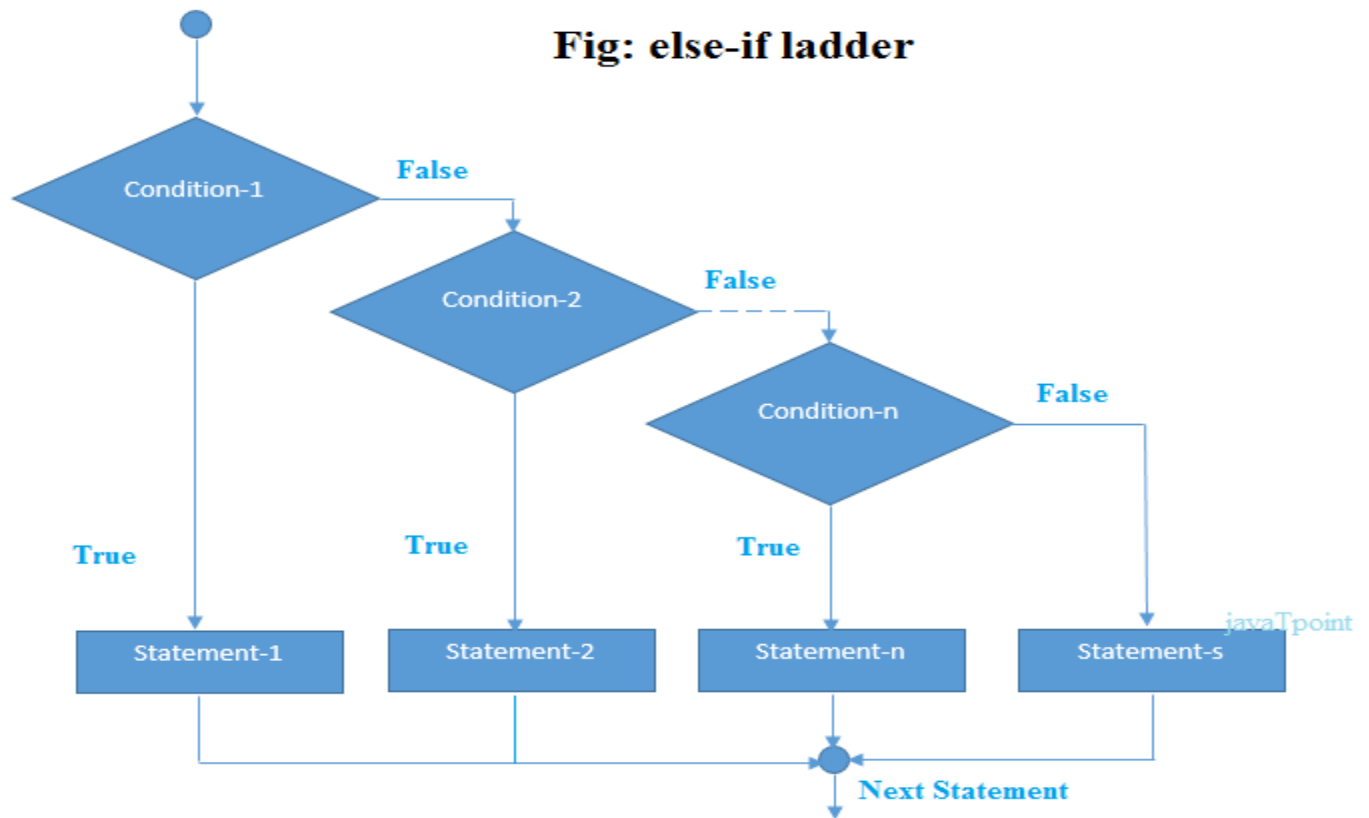
Java if-else-if ladder Statement

The if-else-if ladder statement executes one condition from multiple statements.

Syntax:

```
if(condition1){
//code to be executed if condition1 is true
}else if(condition2){
//code to be executed if condition2 is true
}
else if(condition3){
//code to be executed if condition3 is true
}
...
else{
//code to be executed if all the conditions are false
}
```

Fig: else-if ladder



Example:

//Java Program to demonstrate the use of If else-if ladder.

//It is a program of grading system for fail, D grade, C grade, B grade, A grade and A+.

```
public class IfElseIfExample {  
    public static void main(String[] args) {  
        int marks=65;  
  
        if(marks<50){  
            System.out.println("fail");  
        }  
        else if(marks>=50 && marks<60){  
            System.out.println("D grade");  
        }  
        else if(marks>=60 && marks<70){  
            System.out.println("C grade");  
        }  
    }  
}
```

```

}
else if(marks>=70 && marks<80){
    System.out.println("B grade");
}
else if(marks>=80 && marks<90){
    System.out.println("A grade");
}else if(marks>=90 && marks<100){
    System.out.println("A+ grade");
}else{
    System.out.println("Invalid!");
}
}
}

```

Output:

C grade

Program to check POSITIVE, NEGATIVE or ZERO:

```

public class PositiveNegativeExample {
public static void main(String[] args) {
    int number=-13;
    if(number>0){
        System.out.println("POSITIVE");
    }else if(number<0){
        System.out.println("NEGATIVE");
    }else{
        System.out.println("ZERO");
    }
}
}
}

```

Output:

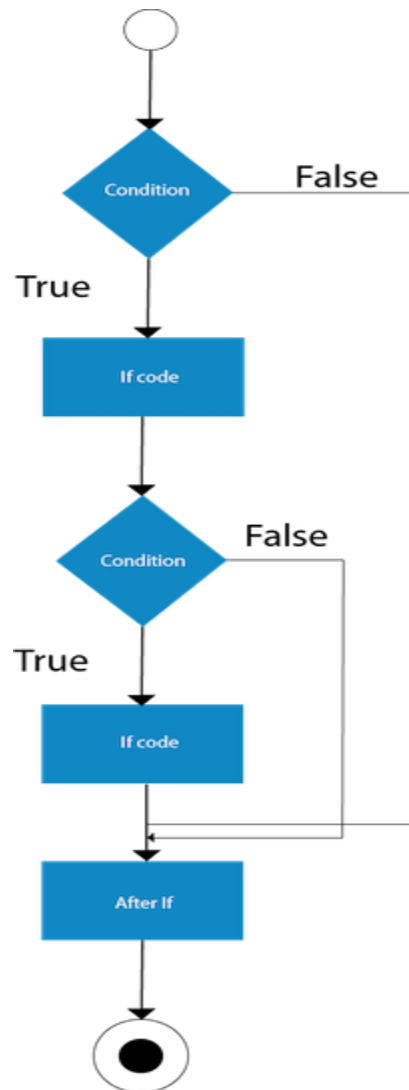
NEGATIVE

Java Nested if statement

The nested if statement represents the *if block within another if block*. Here, the inner if block condition executes only when outer if block condition is true.

Syntax:

```
if(condition){  
    //code to be executed  
    if(condition){  
        //code to be executed  
    }  
}
```



Example:

//Java Program to demonstrate the use of Nested If Statement.

```
public class JavaNestedIfExample {  
    public static void main(String[] args) {  
        //Creating two variables for age and weight  
        int age=20;  
        int weight=80;  
        //applying condition on age and weight  
        if(age>=18){  
            if(weight>50){  
                System.out.println("You are eligible to donate blood");  
            }  
        }  
    }  
}
```

Output:

```
You are eligible to donate blood
```

Example 2:

//Java Program to demonstrate the use of Nested If Statement.

```
public class JavaNestedIfExample2 {  
    public static void main(String[] args) {  
        //Creating two variables for age and weight  
        int age=25;  
        int weight=48;  
        //applying condition on age and weight  
        if(age>=18){  
            if(weight>50){  
                System.out.println("You are eligible to donate blood");  
            } else{  
                System.out.println("You are not eligible to donate blood");  
            }  
        }  
    }  
}
```

```
    }  
  } else{  
    System.out.println("Age must be greater than 18");  
  }  
} }
```

Output:

```
You are not eligible to donate blood
```

Java Simple for Loop

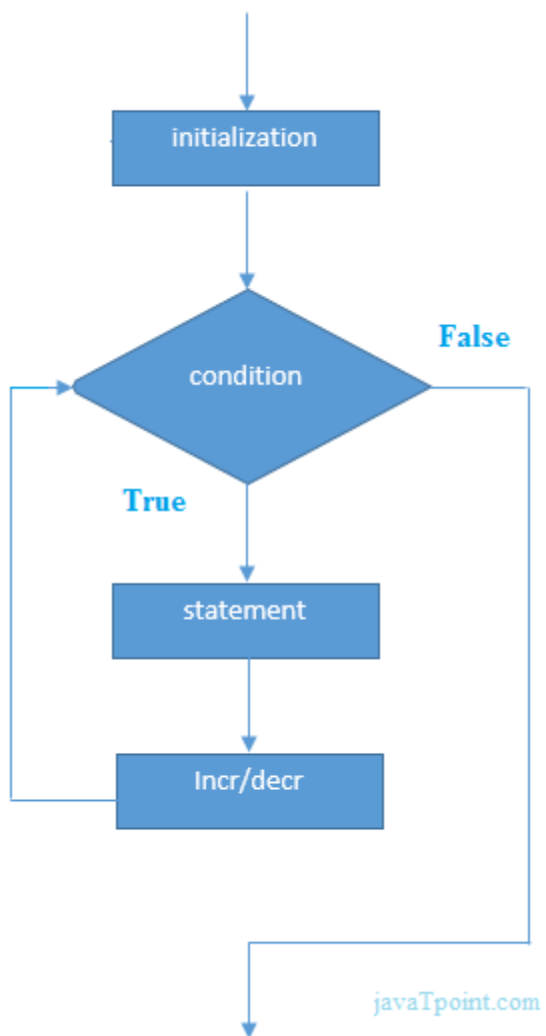
A simple for loop is the same as **C/C++**. We can initialize the **variable**, check condition and increment/decrement value. It consists of four parts:

1. **Initialization:** It is the initial condition which is executed once when the loop starts. Here, we can initialize the variable, or we can use an already initialized variable. It is an optional condition.
2. **Condition:** It is the second condition which is executed each time to test the condition of the loop. It continues execution until the condition is false. It must return boolean value either true or false. It is an optional condition.
3. **Increment/Decrement:** It increments or decrements the variable value. It is an optional condition.
4. **Statement:** The statement of the loop is executed each time until the second condition is false.

Syntax:

```
for(initialization; condition; increment/decrement){  
  //statement or code to be executed  
}
```

Flowchart:



Example:

ForExample.java

//Java Program to demonstrate the example of for loop
//which prints table of 1

```
public class ForExample {  
public static void main(String[] args) {  
    //Code of Java for loop  
    for(int i=1;i<=10;i++){  
        System.out.println(i);  
    }  
}
```

```
}  
}
```

Output:

```
1  
2  
3  
4  
5  
6  
7  
8  
9  
10
```

Java While Loop

The [Java while loop](#) is used to iterate a part of the [program](#) repeatedly until the specified Boolean condition is true. As soon as the Boolean condition becomes false, the loop automatically stops.

Syntax:

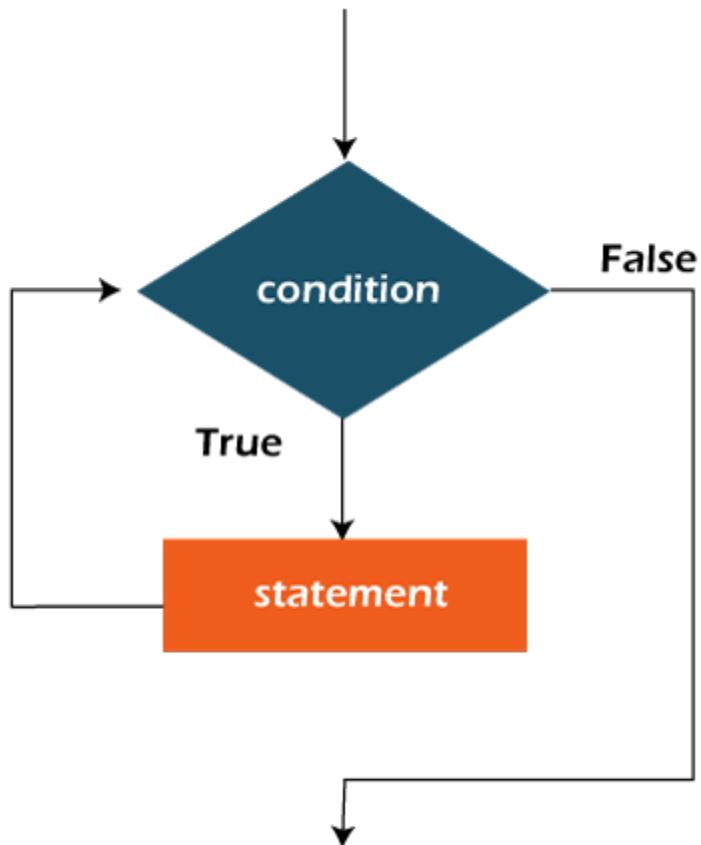
```
while (condition){  
  //code to be executed  
  Increment / decrement statement  
}
```

The different parts of do-while loop:

1. Condition: It is an expression which is tested. If the condition is true, the loop body is executed and control goes to update expression. When the condition becomes false, we exit the while loop.

Flowchart of Java While Loop

Here, the important thing about while loop is that, sometimes it may not even execute. If the condition to be tested results into false, the loop body is skipped and first statement after the while loop will be executed.



Example:

In the below example, we print integer values from 1 to 10. Unlike the for loop, we separately need to initialize and increment the variable used in the condition (here, i). Otherwise, the loop will execute infinitely.

WhileExample.java

```
public class WhileExample {  
    public static void main(String[] args) {  
        int i=1;  
        while(i<=10){  
            System.out.println(i);  
            i++;  
        }  
    }  
}
```

Output:

```
1
2
3
4
5
6
7
8
9
10
```

Java do-while Loop

The Java *do-while loop* is used to iterate a part of the program repeatedly, until the specified condition is true. If the number of iteration is not fixed and you must have to execute the loop at least once, it is recommended to use a do-while loop.

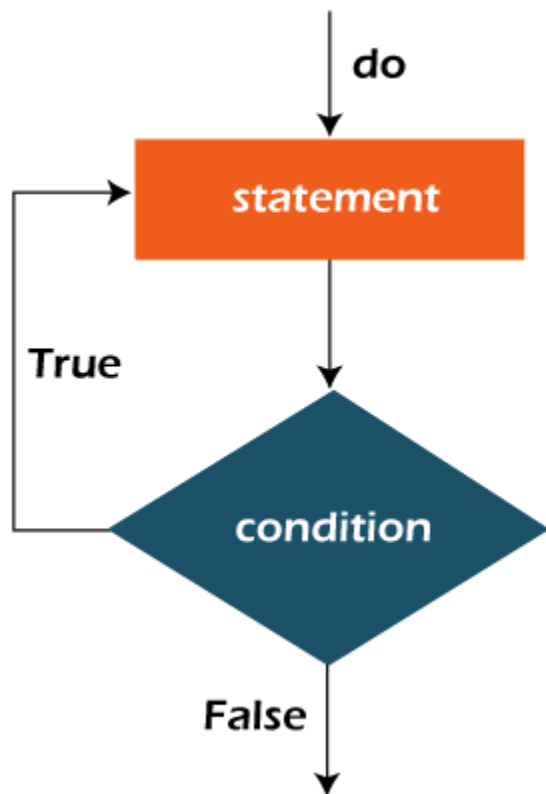
Java do-while loop is called an **exit control loop**. Therefore, unlike while loop and for loop, the do-while check the condition at the end of loop body. The Java *do-while loop* is executed at least once because condition is checked after loop body.

Syntax:

```
do{
//code to be executed / loop body
//update statement
}while (condition);
```

Note: The do block is executed at least once, even if the condition is false.

Flowchart of do-while loop:



Example:

In the below example, we print integer values from 1 to 10. Unlike the for loop, we separately need to initialize and increment the variable used in the condition (here, i). Otherwise, the loop will execute infinitely.

DoWhileExample.java

```
public class DoWhileExample {  
    public static void main(String[] args) {  
        int i=1;  
        do{  
            System.out.println(i);  
            i++;  
        }while(i<=10);  
    }  
}
```

}

Output:

```
1
2
3
4
5
6
7
8
9
10
```

Java Arrays

Normally, an array is a collection of similar type of elements which has contiguous memory location.

Java array is an object which contains elements of a similar data type. Additionally, The elements of an array are stored in a contiguous memory location. It is a data structure where we store similar elements. We can store only a fixed set of elements in a Java array.

Array in Java is index-based, the first element of the array is stored at the 0th index, 2nd element is stored on 1st index and so on.

Advantages

Code Optimization: It makes the code optimized, we can retrieve or sort the data efficiently.

Random access: We can get any data located at an index position.

Disadvantages

Size Limit: We can store only the fixed size of elements in the array. It doesn't grow its size at runtime. To solve this problem, collection framework is used in Java which grows automatically.

/Java Program to illustrate how to declare, instantiate, initialize

```
//and traverse the Java array.  
class Testarray{  
public static void main(String args[]){  
int a[]=new int[5];//declaration and instantiation  
a[0]=10;//initialization  
a[1]=20;  
a[2]=70;  
a[3]=40;  
a[4]=50;  
//traversing array  
for(int i=0;i<a.length;i++)//length is the property of array  
System.out.println(a[i]);  
}}
```

Output:

```
10  
20  
70  
40  
50
```

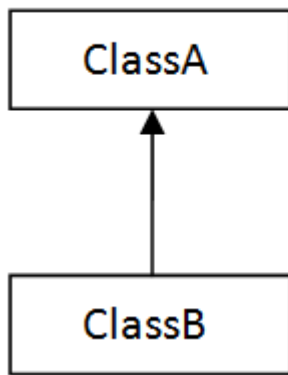
Java Inheritance (Subclass and Superclass)

In Java, it is possible to inherit attributes and methods from one class to another. We group the "inheritance concept" into two categories:

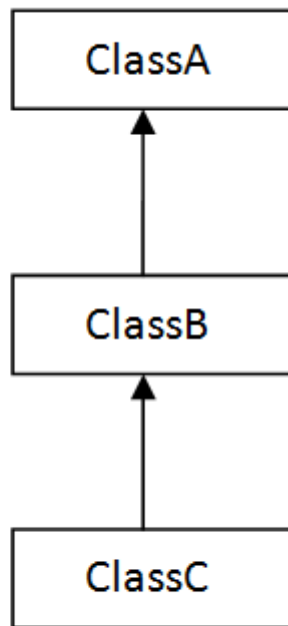
- **subclass** (child) - the class that inherits from another class
- **superclass** (parent) - the class being inherited from

To inherit from a class, use the **extends** keyword.

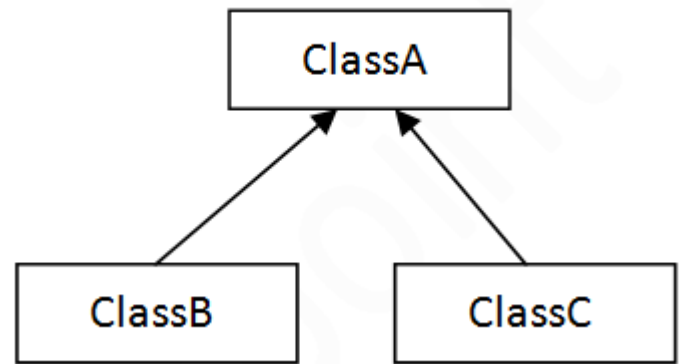
```
class Employee{  
float salary=40000;
```



1) Single



2) Multilevel



3) Hierarchical

```

}
class Programmer extends Employee{
    int bonus=10000;
    public static void main(String args[]){
        Programmer p=new Programmer();
        System.out.println("Programmer salary is:"+p.salary);
        System.out.println("Bonus of Programmer is:"+p.bonus);
    }
}

```

```

Programmer salary is:40000.0
Bonus of programmer is:10000

```

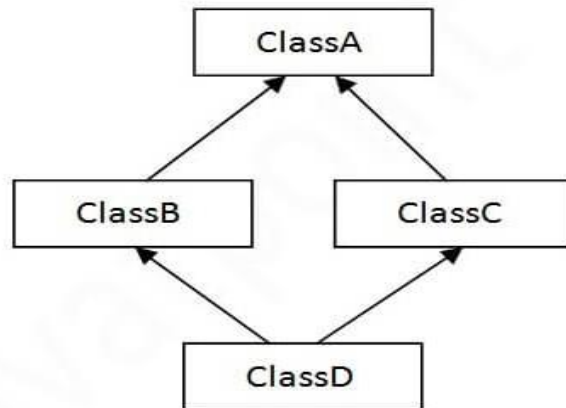
Types of inheritance in java

On the basis of class, there can be three types of inheritance in java: single, multilevel and hierarchical.

In java programming, multiple and hybrid inheritance is supported through interface only. We will learn about interfaces later.

Note: Multiple inheritance is not supported in Java through class.

When one class inherits multiple classes, it is known as multiple inheritance. For Example:



5) Hybrid

Single Inheritance Example

When a class inherits another class, it is known as a *single inheritance*. In the example given below, Dog class inherits the Animal class, so there is the single inheritance.

File: TestInheritance.java

```
class Animal{
void eat(){System.out.println("eating...");}
}
class Dog extends Animal{
void bark(){System.out.println("barking...");}
}
class TestInheritance{
public static void main(String args[]){
Dog d=new Dog();
d.bark();
d.eat();
}}
```

Output:

```
barking...
eating...
```

Multilevel Inheritance Example

When there is a chain of inheritance, it is known as *multilevel inheritance*. As you can see in the example given below, BabyDog class inherits the Dog class which again inherits the Animal class, so there is a multilevel inheritance.

File: TestInheritance2.java

```
class Animal{
void eat(){System.out.println("eating...");}
}
class Dog extends Animal{
void bark(){System.out.println("barking...");}
}
```

```
class BabyDog extends Dog{
void weep(){System.out.println("weeping...");}
}
class TestInheritance2{
public static void main(String args[]){
BabyDog d=new BabyDog();
d.weep();
d.bark();
d.eat();
}}
```

Output:

```
weeping...
barking...
eating...
```

Hierarchical Inheritance Example

When two or more classes inherits a single class, it is known as *hierarchical inheritance*. In the example given below, Dog and Cat classes inherits the Animal class, so there is hierarchical inheritance.

File: TestInheritance3.java

```
class Animal{
void eat(){System.out.println("eating...");}
}
class Dog extends Animal{
void bark(){System.out.println("barking...");}
}
class Cat extends Animal{
void meow(){System.out.println("meowing...");}
}
class TestInheritance3{
public static void main(String args[]){
Cat c=new Cat();
c.meow();
}
```

```
c.eat();  
//c.bark();//C.T.Error  
}}
```

Output:

```
meowing...  
eating...
```

hybrid inheritance

```
//parent class  
class GrandFather  
{  
  public void showG()  
  {  
    System.out.println("He is grandfather.");  
  }  
}  
//inherits GrandFather properties  
class Father extends GrandFather  
{  
  public void showF()  
  {  
    System.out.println("He is father.");  
  }  
}  
//inherits Father properties  
class Son extends Father  
{  
  public void showS()  
  {  
    System.out.println("He is son.");  
  }  
}  
//inherits Father properties  
public class Daughter extends Father  
{  
  public void showD()  
  {  
    System.out.println("She is daughter.");  
  }  
}
```

```

public static void main(String args[])
{
//Daughter obj = new Daughter();
//obj.show();
Son obj = new Son();
obj.showS(); // Accessing Son class method
obj.showF(); // Accessing Father class method
obj.showG(); // Accessing GrandFather class method
Daughter obj2 = new Daughter();
obj2.showD(); // Accessing Daughter class method
obj2.showF(); // Accessing Father class method
obj2.showG(); // Accessing GrandFather class method
}
}

```

Output:

```

He is son.
He is father.
He is grandfather.
She is daughter.
He is father.
He is grandfather.

```

Method Overriding in Java

In Java, method overriding occurs when a subclass (child class) has the same method as the parent class. In other words, method overriding occurs when a subclass provides a particular implementation of a method declared by one of its parent classes.

```

//Java Program to demonstrate the real scenario of Java Method Overriding
//where three classes are overriding the method of a parent class.
//Creating a parent class.

```

```

class Bank{
int getRateOfInterest()
{
return 0;
}
}
//Creating child classes.
class SBI extends Bank
{

```

```

int getRateOfInterest()
{
    return 8;
}

class ICICI extends Bank
{
    int getRateOfInterest()
    {
        return 7;
    }
}

class AXIS extends Bank
{
    int getRateOfInterest()
    {
        return 9;
    }
}

//Test class to create objects and call the methods
class Test2
{
    public static void main(String args[])
    {
        SBI s=new SBI();
        ICICI i=new ICICI();
        AXIS a=new AXIS();
        System.out.println("SBI Rate of Interest: "+s.getRateOfInterest());
        System.out.println("ICICI Rate of Interest: "+i.getRateOfInterest());
        System.out.println("AXIS Rate of Interest: "+a.getRateOfInterest());
    }
}

```

Output:
SBI Rate of Interest: 8
ICICI Rate of Interest: 7
AXIS Rate of Interest: 9

Abstract class in Java

A class which is declared with the abstract keyword is known as an abstract class in [Java](#). It can have abstract and non-abstract methods (method with the body).

Before learning the Java abstract class, let's understand the abstraction in Java first.

Abstraction in Java

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.

Abstract class in Java

A class which is declared as abstract is known as an **abstract class**. It can have abstract and non-abstract methods. It needs to be extended and its method implemented. It cannot be instantiated.

Points to Remember

- An abstract class must be declared with an abstract keyword.
 - It can have abstract and non-abstract methods.
 - It cannot be instantiated.
 - It can have [constructors](#) and static methods also.
 - It can have final methods which will force the subclass not to change the body of the method.
-

Rules for Java Abstract class



Example of abstract class

1. **abstract class** A{

Abstract Method in Java

A method which is declared as abstract and does not have implementation is known as an abstract method.

Example of abstract method

1. **abstract void** printStatus();//no method body and abstract

Example of Abstract class that has an abstract method

In this example, Bike is an abstract class that contains only one abstract method run. Its implementation is provided by the Honda class.


```

abstract class Bike{
    abstract void run();
}
class Honda4 extends Bike{
void run(){System.out.println("running safely");}
public static void main(String args[]){
    Bike obj = new Honda4();
    obj.run();
}
}

```

running safely

File: TestAbstraction1.java

```

abstract class Shape{
abstract void draw();
}
//In real scenario, implementation is provided by others i.e. unknown by end user
class Rectangle extends Shape{
void draw(){System.out.println("drawing rectangle");}
}
class Circle1 extends Shape{
void draw(){System.out.println("drawing circle");}
}
//In real scenario, method is called by programmer or user
class TestAbstraction1{
public static void main(String args[]){
    Shape s=new Circle1();//In a real scenario, object is provided through method, e.g., getShape() method
    s.draw();
}
}

```

drawing circle

Another example of Abstract class in java

File: TestBank.java

```

abstract class Bank{
abstract int getRateOfInterest();
}

```

```
class SBI extends Bank{
int getRateOfInterest(){return 7;}
}
class PNB extends Bank{
int getRateOfInterest(){return 8;}
}
```

```
class TestBank{
public static void main(String args[]){
Bank b;
b=new SBI();
System.out.println("Rate of Interest is: "+b.getRateOfInterest()+" %");
b=new PNB();
System.out.println("Rate of Interest is: "+b.getRateOfInterest()+" %");
}}
```

Rate of Interest is: 7 %

Rate of Interest is: 8 %