

## Introduction to Java

- Java is a computer programming language. It enables programmers to write computer instructions using English-based commands instead of writing in numeric codes.
- It's known as a "**high-level**" language because humans can read and write easily. Like English, Java has a set of rules that determine how the instructions are written. These rules are known as its "syntax". Once a program has been written, the high-level instructions are translated into numeric codes that computers can understand and execute.
- Java is an **object-oriented** programming language.

## Java History

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 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 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".

Currently, Java is used in internet programming, mobile devices, games, e-business solutions, etc. Significant points describe the history of Java.



- **James Gosling, Mike Sheridan, and Patrick Naughton** initiated the Java language project in June 1991. The small team of sun engineers is called the **Green Team**.
- Originally designed for small, embedded systems in electronic appliances like set-top boxes.
- Firstly, it was called "**Green Talk**" 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.
- **Why Oak?** Oak is a symbol of strength and has been chosen as a national tree in many countries like the U.S.A., France, Germany, Romania, etc.
- In 1995, Oak was renamed "**Java**" because it was already a trademark by Oak Technologies.
- Java is an island of Indonesia where the first coffee was produced (java coffee).
- Notice that Java is just a name, not an acronym.
- Initially developed by James Gosling at Sun Microsystems (now a subsidiary of Oracle Corporation) and released in 1995.
- In 1995, Time magazine called **Java one of the Ten Best Products of 1995**.
- JDK 1.0 was released on (23 January 1996).

## Java Version History

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

JDK Alpha and Beta (1995)

JDK 1.0 (23rd Jan 1996)

JDK 1.1 (19th Feb 1997)

J2SE 1.2 (8th Dec 1998)

J2SE 1.3 (8th May 2000)

J2SE 1.4 (6th Feb 2002)

J2SE 5.0 (30th Sep 2004)

Java SE 6 (11th Dec 2006)

Java SE 7 (28th July 2011)

**Java SE 8-LTS-Long Term Support** (18th Mar 2014)

Java SE 9 (21st Sep 2017)

Java SE 10 (20th Mar 2018)

**Java SE 11-LTS (September 2018)**

Java SE 12 (March 2019)

Java SE 13 (September 2019)

Java SE 14 (Mar 2020)

Java SE 15 (September 2020)

Java SE 16 (Mar 2021)

**Java SE 17-LTS (September 2021)**

Java SE 18 (March 2022)

Java SE 19 (September 2022)

Java SE 20 (March 2023)

**Java SE 21-LTS (September 2023)**

Java SE 22 (March 2024)

Java SE 23 (September 24)

Java SE 24 (March 24)

**Java SE 25-LTS (September 25)**

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

**Where is it used?**

According to Sun, 3 billion devices run Java. There are many devices where Java is currently used.

Some of them are as follows:

1. Desktop Applications such as Acrobat Reader, media player, antivirus, etc.
2. Web Applications such as irctc.co.in etc.
3. Enterprise Applications such as banking applications.
4. Mobile Application
5. Smart Card, Debit, Credit Cards

6. Robotics
7. Games
8. Television set-top boxes, Coffee Machine, Toaster machine, etc.

## Types of Java Applications

There are mainly four types of applications that can be created using Java programming:

### 1) Desktop Application

It is also known as a desktop application or window-based application. We must install an application on every machine, such as a media player, antivirus, etc. AWT and Swing are used in Java to create standalone applications.

### 2) Web Application

An application that runs on the server side and creates dynamic pages is called a web application. Currently, servlet, jsp, struts, JSF, etc. technologies are used to create web applications in Java.

### 3) Enterprise Application

An application that is distributed in nature, such as banking applications, etc. It has the advantage of high-level security, load balancing, and clustering. In Java, EJB is used to create enterprise applications.

### 4) Mobile Application

An application that is created for mobile devices. Currently, Android and Java ME are used to create mobile applications.

## Java Editions/Platforms

There are four platforms or editions of Java:

**Java Standard Edition (JSE):** The Java Standard Edition (Java SE) is for building desktop applications and applets. These applications typically serve only a small number of users at one time. It is a Java programming platform. It includes Java programming APIs such as Java.lang, java.io, java.net, and java.util, java.sql, java.math etc. It includes core topics like OOPs, String, Regex, Exception, Inner classes, Multithreading, I/O Stream, Networking, AWT, Swing, Reflection, Collection, etc.

**Java Enterprise Edition (JEE):** It is an enterprise platform, which is mainly used to develop web and enterprise applications. It is built on the top of the Java SE platform. It includes topics like Servlet, JSP, Web Services, EJB, JPA, etc.

**Java Micro Edition (JME):** The Java Micro Edition is for mobile applications (e.g., cell phones, PDAs) and embedded devices (e.g., TV tuner boxes, printers).

**JAVA Card:** used in SIM Cards, ATM Cards, Credit Cards, and SMART Card reader programming.

## JAVA FEATURES

Java was designed with a few key principles in mind:

**1. Simple:**

- It resembles with C++.
- No need for header files like conio.h, iostream.h, etc.
- No pointers, structures, union, operator overloading, virtual base class, friend function, etc.
- The size of the basic interpreter and class support is about 40k bytes.
- The basic standard libraries and thread support add 175k bytes only.
- Java provides a bug-free system due to its strong memory management.
- It also has an automatic memory allocation and deallocation system.

**2. Object Oriented:**

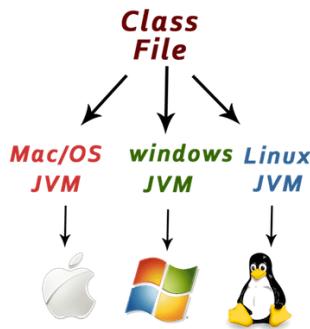
It supports all the features of OOPS. Some of the important object-oriented features are namely:

- Class
- Object
- Inheritance
- Interfaces
- Packages
- Data Abstraction
- Data Encapsulation
- Polymorphism

- Overloading
- Reusability

### **3. Platform Independent and Architecture Neutral:**

- The concept of write-once-run-anywhere (WORA- known as the Platform Independent) is one of the key features of Java that makes Java the most powerful language.
- The programs written on one platform can run on any platform, provided the platform must have the JVM.



- Java was written to be a portable language that does not care about the operating system or computer hardware.
- The size of primitive data types is independent.
- Architecture-neutral
- Java is architecture-neutral because there are no implementation-dependent features; for example, the size of primitive types is fixed.
- In C programming, the int data type occupies 2 bytes of memory for 32-bit architecture and 4 bytes of memory for 64-bit architecture. For example, Borland C++ takes 4 bytes, and Turbo C++ takes 2. However, JAVA occupies 4 bytes of memory for 32- and 64-bit architectures in Java.

### **4. Distributed:**

- Java supports many networking protocols like TCP, IP, HTTP, and FTP, and Java has an extensive library of these protocols.
- Java applications can open and access objects across the internet via urls. Internet programmers can call functions on these protocols and access the files from any remote machine on the internet rather than writing codes on their local system.

- Java supports networking capabilities like socket connection, common gateway Interface (CGI) scripting, Servlet, JSP, and JSF- make server-side processing in Java efficient.
- Java is distributed because it facilitates users in creating distributed applications in Java. RMI and EJB are used to create distributed applications. This feature of Java allows us to access files by calling the methods from any machine on the internet.

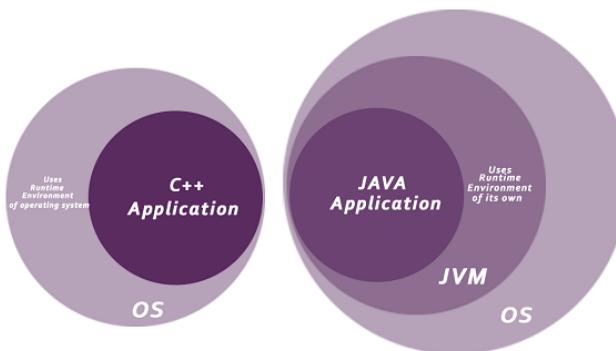
#### **5. Reliable and Robust:**

- Java has a strong memory allocation and automatic garbage collection mechanism that solves all the memory management problems.
- It provides powerful exception handling, which captures errors and eliminates any risk of crashing.
- Provide compile-time and run-time checking -The compiler checks the program to see whether there are any errors, and the interpreter checks any run-time errors and makes the system secure from the crash.

#### **6. Secure:**

- Java does not support a pointer like C/C++. Hence, the memory locations of the system cannot be accessed through a JAVA program, so any program developed in Java cannot be used to hack a system.
- Java uses the public key encryption system to allow the Java applications to transmit over the internet in a secure encrypted form.
- The byte code Verifier checks the classes after loading.

Java programs run inside the virtual machine, so Java is probably the most secure programming language to date.

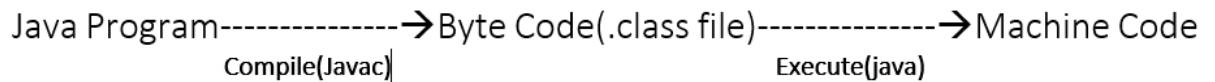


## 7. Portable:

- The feature Write-once-run-anywhere (WORA) means the Java compiler generates byte code instructions that can be implemented on any machine. Java is portable because it allows you to carry the Java bytecode to any platform. It doesn't require any implementation.
  - The size of JAVAC and JAVA is approximately less than 100KB.



## **8. Compiled and Interpreted:**



## 9. 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, so it is slower than compiled languages, e.g., C, C++, etc.

## 10. Popular:

- Java has rich libraries like applets, servlets, etc that make it popular.
  - In C++ Library → Customize → Application
  - In Java Library → Application

## 11. Multithreaded:

- Java language provides an environment by which several tasks can be initiated and managed easily such a feature is called multithreading. Multithreading means a single program having different threads executing independently at the same time.

- Multiple threads execute instructions according to the program code in a process or a program.
- The main advantage of multi-threading is that it doesn't occupy memory for each thread. It shares a common memory area.
- Example: In the MS Word program, create a different thread for functions like Save, Scroll, Line Spacing, font size, auto-save, spell check, delete, object insert, etc.
- Multithreading works similarly to multiple processes run on one computer.

## **12. Dynamic and Extensible:**

- Java is a dynamic language. It supports dynamic loading of classes. It means classes are loaded on demand.
- Java can dynamically link in new classes, libraries, methods, and objects. Java programs support functions written in another language, such as c or C++. These functions are known as native methods; this facility enables the programmers to use the efficient functions available in this language; native methods are linked dynamically at runtime.

The Sun Microsystems team successfully combined these key principles, and Java's popularity can be traced to its robust, secure, easy-to-use, and portable language.

## **Java Virtual Machine (JVM):**

JVM (Java Virtual Machine) is an abstract machine. It is called a virtual machine because it doesn't physically exist.

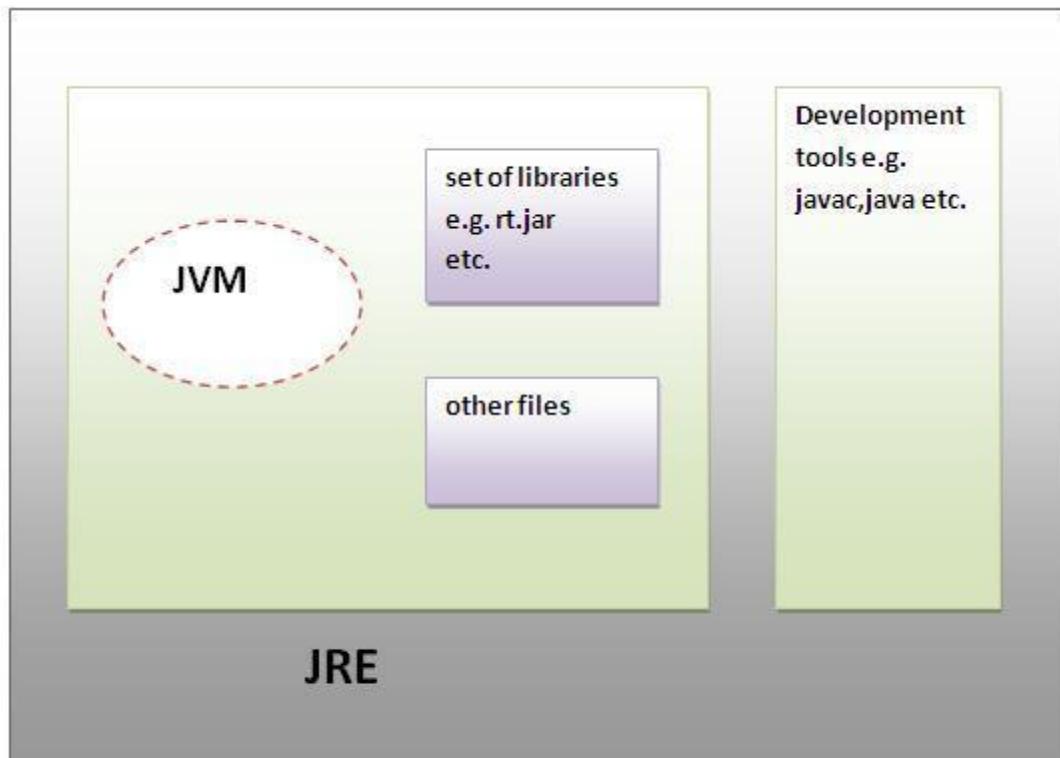
The specification provides a runtime environment in which Java bytecode can be executed.

JVMs are available for many hardware and software platforms.

JVM, JRE, and JDK are platform-dependent because the configuration of each OS differs. But Java is platform-independent.

The JVM performs the following main tasks:

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



### **Java Runtime Environment (JRE):**

JRE is an acronym for Java Runtime Environment. It is also written as **Java RTE**. The Java Runtime Environment is a software tool for developing Java applications. It is used to provide the runtime environment. It is the implementation of JVM. It physically exists. It contains a set of libraries + other files that JVM uses at runtime.

**JDK=JRE + Development Tools (JAVAC, JAVA)**

### **Java Development Kit (JDK):**

JDK is an acronym for Java Development Kit. The Java Development Kit (JDK) is a software development environment for developing Java applications and applets. It physically exists. It contains JRE + development tools.

JDK is an implementation of any one of the below given Java Platforms released by Oracle Corporation:

- Standard Edition Java Platform
- Enterprise Edition Java Platform
- Micro Edition Java Platform

The JDK contains a private Java Virtual Machine (JVM) and a few other resources, such as an interpreter/loader (java), a compiler (javac), an archiver (jar), a documentation generator (Javadoc), etc., to complete the development of a Java Application.

**Example:**

JDK 1.5,JDK 1.6 ,JDk 1.8.....23.

**Note:**

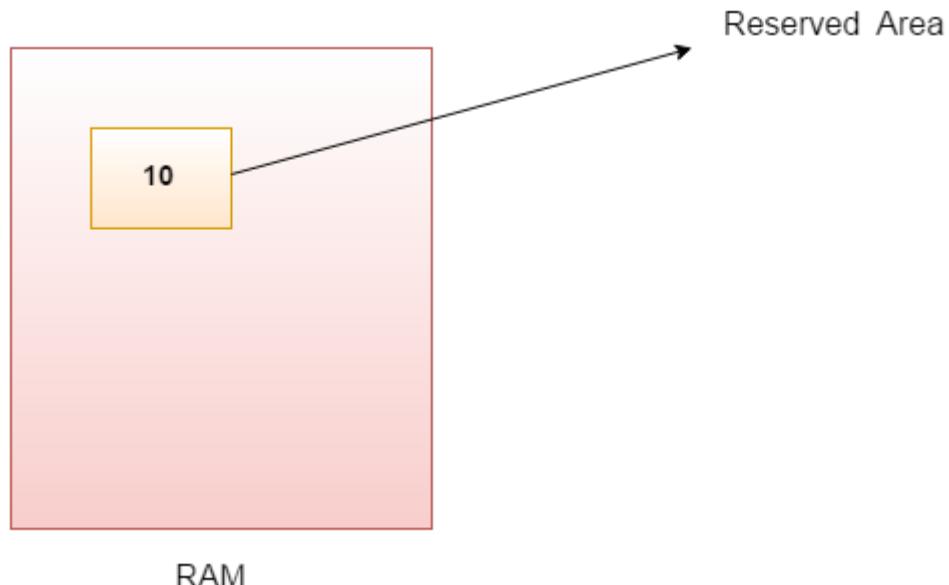
The JRE is contained within the JDK (i.e., if you download the JDK, you will be able to create and run Java programs.).

## Variables and Data Types in Java

- Java variable is a name given to a memory location. It is the basic unit of storage in a program.
- Variables are the data containers that save the data values during Java program execution.
- The value stored in a variable can be changed during program execution.
- In Java, all variables must be declared before use.

### Example:

```
int a=10;
```



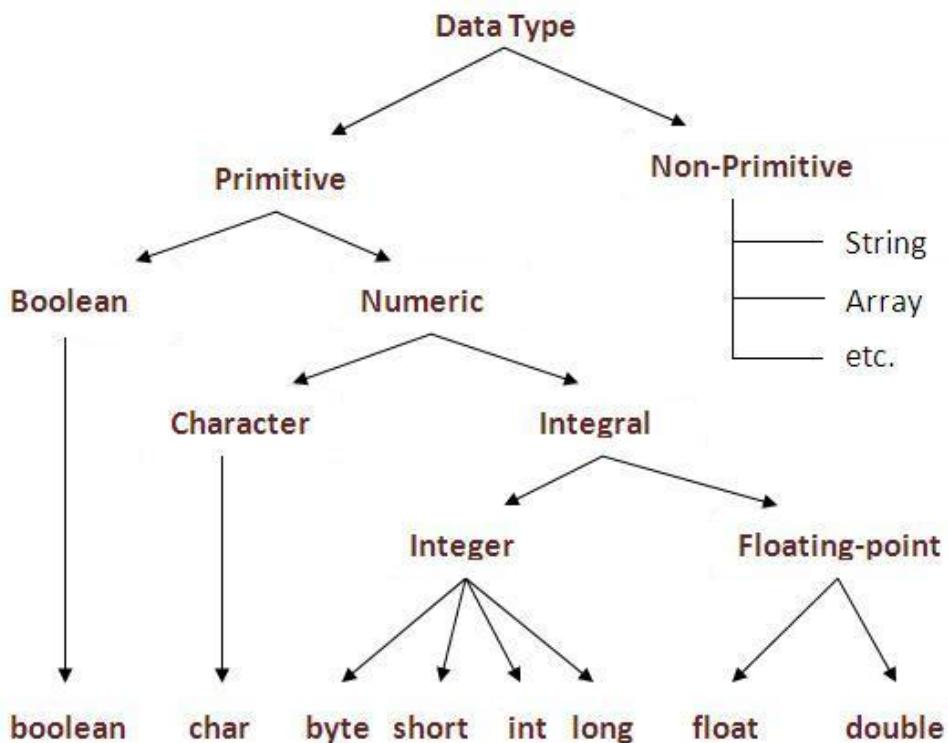
### Note:

In Java, every variable and expression has some type; the compiler should check every assignment for type compatibility; because of this reason, Java language is a strongly typed programming language.

## Data Types in Java

Data types represent the different values to be stored in the variable. In Java, there are two types of data types:

- **Primitive data types:** The primitive data types include boolean, char, byte, short, int, long, float, and double.
- **Non-primitive data types:** The non-primitive data types include Classes, Interfaces, and Arrays.



| Data Type | Default Value | Default size |
|-----------|---------------|--------------|
| boolean   | false         | 1 bit        |
| char      | '\u0000'      | 2 byte       |
| byte      | 0             | 1 byte       |
| short     | 0             | 2 byte       |
| int       | 0             | 4 byte       |
| long      | 0L            | 8 byte       |
| float     | 0.0f          | 4 byte       |
| double    | 0.0d          | 8 byte       |

## Data Type Range

The range of values is calculated as  $-(2^{n-1}) \text{ to } (2^{n-1}) - 1$

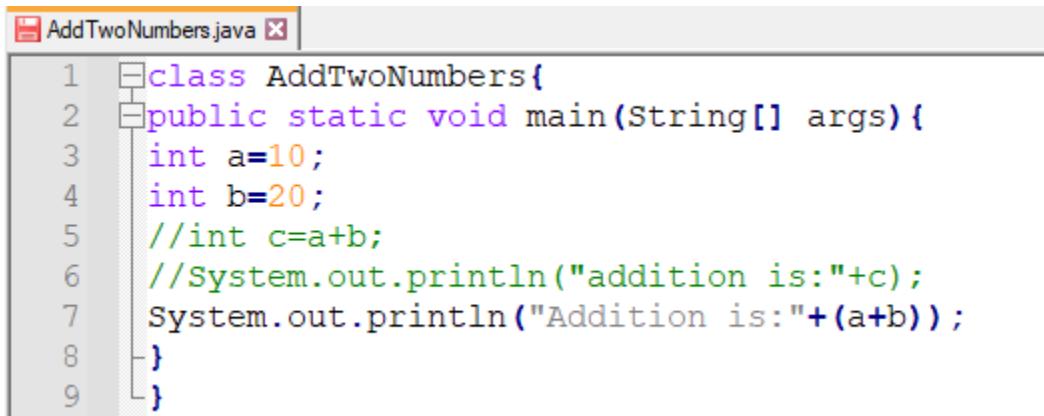
Where n is the number of bits required.

For example, the byte data type requires 1 byte = 8 bits. Therefore, the range of values that can be stored in the byte data type is  $-(2^{8-1}) \text{ to } (2^{8-1}) - 1$ .  $= -2^7 \text{ to } (2^7) - 1$ .  $= -128 \text{ to } 127$ .

| Data Type | Size    | Description                                 |
|-----------|---------|---|
| byte      | 1 byte  | Stores whole numbers from -128 to 127       |
| short     | 2 bytes | Stores whole numbers from -32,768 to 32,767 |

|         |         |   |
|---------|---------|---|
| int     | 4 bytes | Stores whole numbers from -2,147,483,648 to 2,147,483,647                         |
| long    | 8 bytes | Stores whole numbers from -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807 |
| float   | 4 bytes | Stores fractional numbers. Sufficient for storing 6 to 7 decimal digits           |
| double  | 8 bytes | Stores fractional numbers. Sufficient for storing 15 decimal digits               |
| boolean | 1 bit   | Stores true or false values   |
| char    | 2 bytes | Stores a single character/letter or ASCII values                                  |

## Example: Add Two Numbers



```
1 class AddTwoNumbers{
2     public static void main(String[] args){
3         int a=10;
4         int b=20;
5         //int c=a+b;
6         //System.out.println("addition is:"+c);
7         System.out.println("Addition is:"+ (a+b));
8     }
9 }
```

### Output:

```
D:\Java Code 2k23>javac AddTwoNumbers.java

D:\Java Code 2k23>java AddTwoNumbers
Addition is:30
```

## Example: Arithmetic Operations

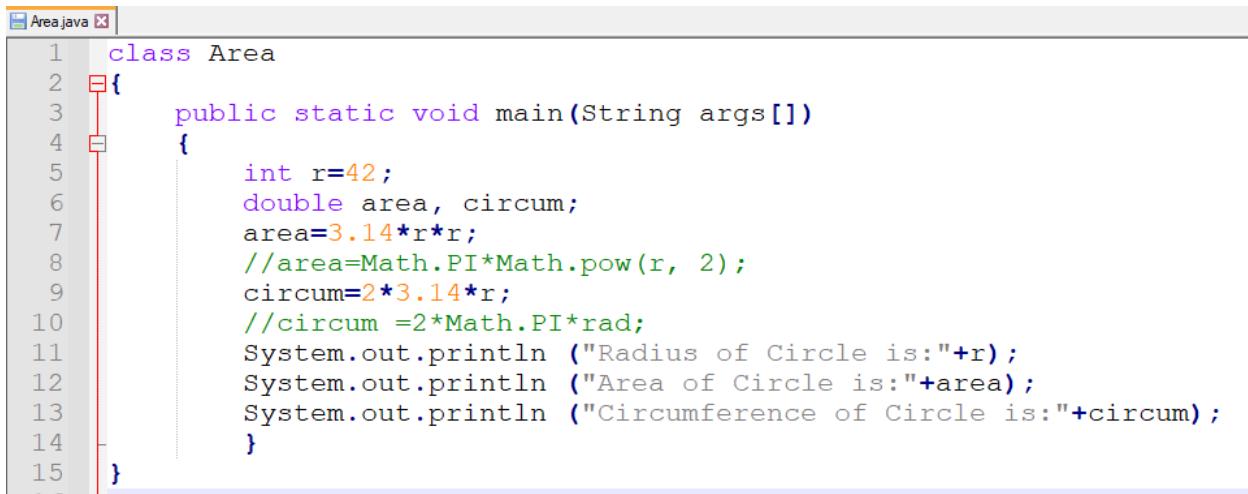
```
ArithOperation.java
1  class ArithOperation
2  {
3      public static void main(String args[])
4      {
5          int a=52,b=20;
6          System.out.println("Value of a:"+a);
7          System.out.println("Value of b:"+b);
8          //System.out.println("Value of a is:"+a +"\nValue of b is:"+b);
9          int c=a+b;
10         System.out.println("Sum is:"+ c);
11         //System.out.println("Sum is:"+ (a+b));
12         System.out.println("Difference is:"+ (a-b));
13         System.out.println("Multiplication is:"+ (a*b));
14         System.out.println("Division is:"+ (a/b));
15         System.out.println("Remainder is:"+ (a%b));
16     }
17 }
18
```

Output:

```
C:\Windows\system32\cmd.exe
D:\Java Code 2k23>javac ArithOperation.java

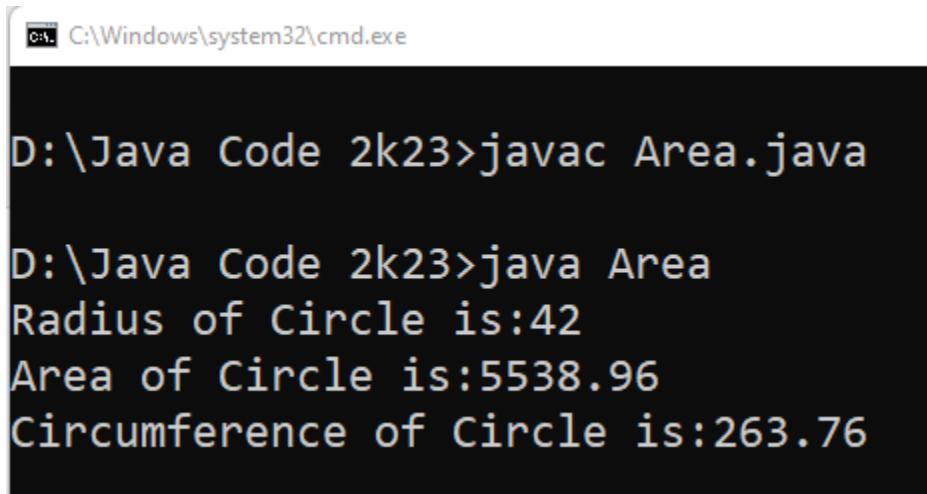
D:\Java Code 2k23>java ArithOperation
Value of a:52
Value of b:20
Sum is:72
Difference is:32
Multiplication is:1040
Division is:2
Remainder is:12
```

## Example: Calculate the area and circumference of a circle.



```
1 class Area
2 {
3     public static void main(String args[])
4     {
5         int r=42;
6         double area, circum;
7         area=3.14*r*r;
8         //area=Math.PI*Math.pow(r, 2);
9         circum=2*3.14*r;
10        //circum =2*Math.PI*rad;
11        System.out.println ("Radius of Circle is:"+r);
12        System.out.println ("Area of Circle is:"+area);
13        System.out.println ("Circumference of Circle is:"+circum);
14    }
15 }
```

Output:



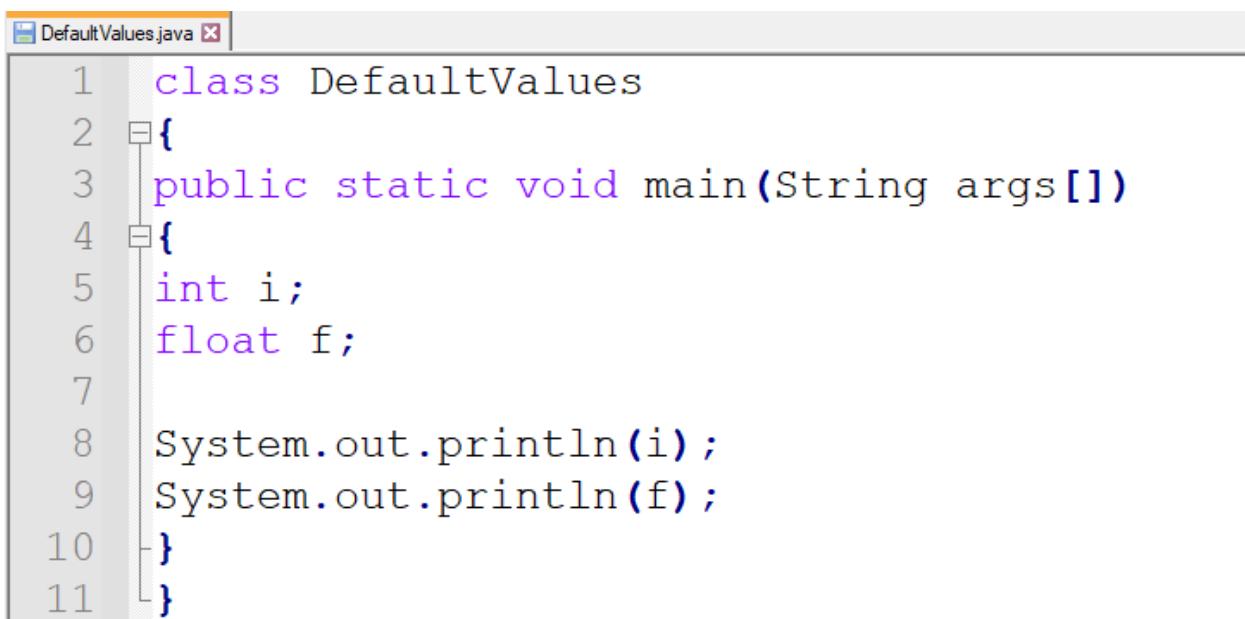
```
C:\Windows\system32\cmd.exe

D:\Java Code 2k23>javac Area.java

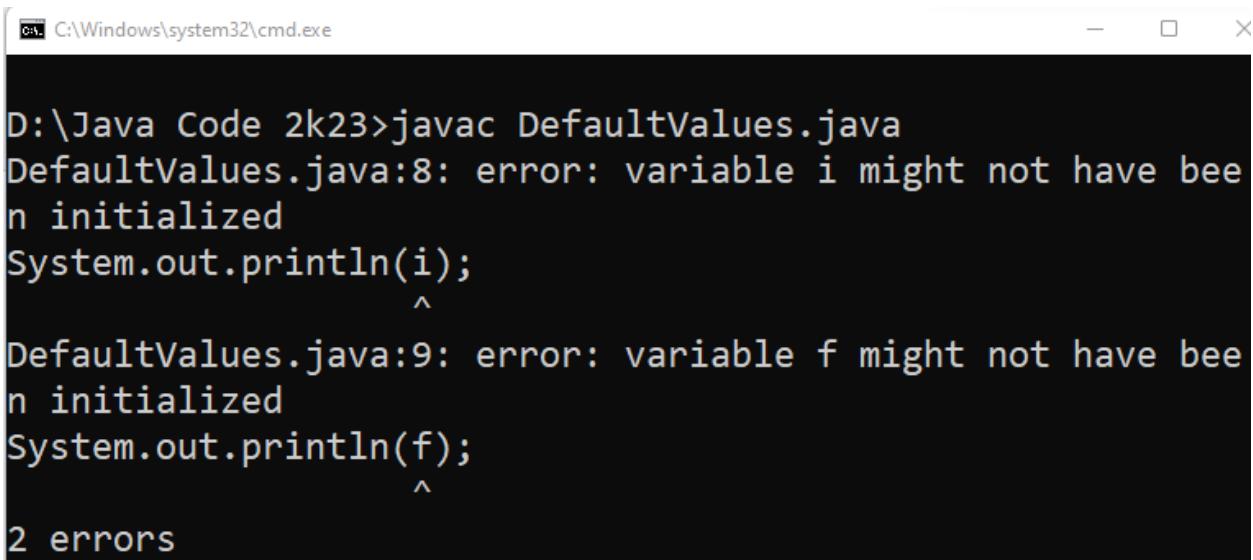
D:\Java Code 2k23>java Area
Radius of Circle is:42
Area of Circle is:5538.96
Circumference of Circle is:263.76
```

## Printing default values of primitive data types.

### Case 1:



```
1  class DefaultValues
2  {
3      public static void main(String args[])
4      {
5          int i;
6          float f;
7
8          System.out.println(i);
9          System.out.println(f);
10     }
11 }
```



```
c:\> C:\Windows\system32\cmd.exe
D:\Java Code 2k23>javac DefaultValues.java
DefaultValues.java:8: error: variable i might not have been initialized
System.out.println(i);
                  ^
DefaultValues.java:9: error: variable f might not have been initialized
System.out.println(f);
                  ^
2 errors
```

### Solution:

```
1 class DefaultValues
2 {
3     public static void main(String args[])
4     {
5         int i=0;
6         float f=0.0f;
7
8         System.out.println(i);
9         System.out.println(f);
10    }
11 }
```

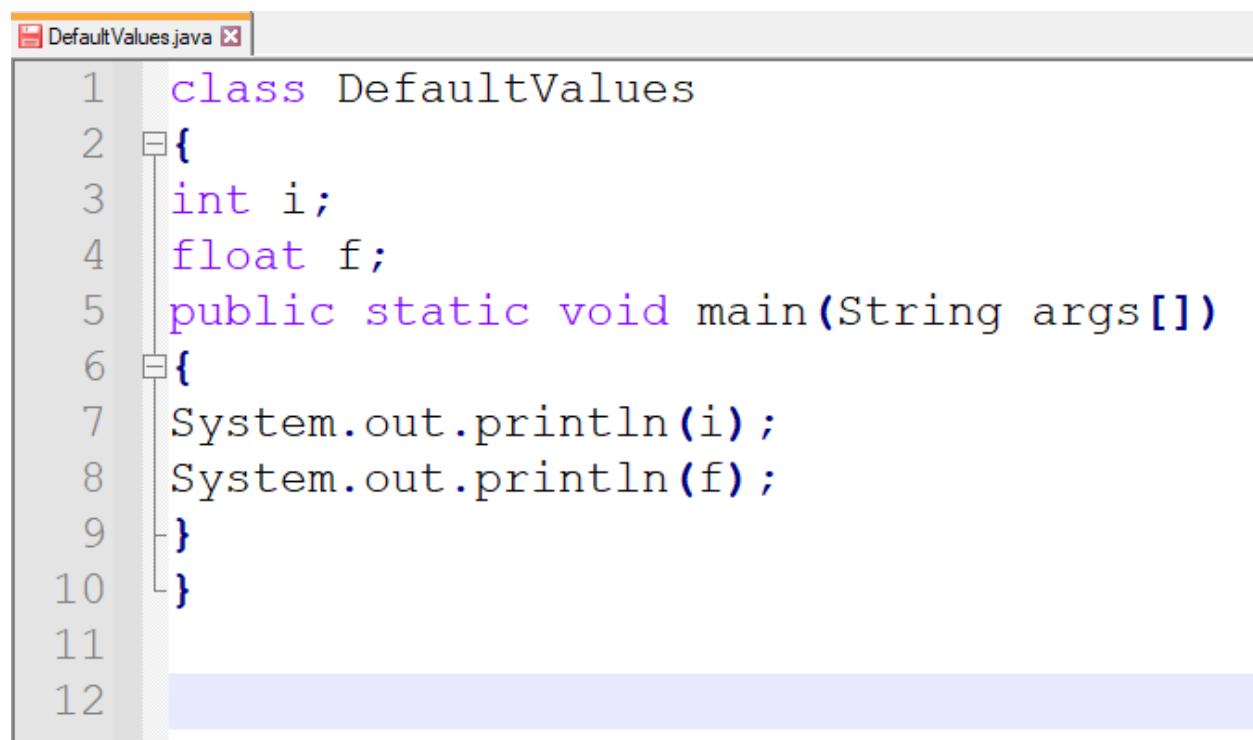
### Output:

```
D:\Java Code 2k23>javac DefaultValues.java

D:\Java Code 2k23>java DefaultValues
0
0.0
```

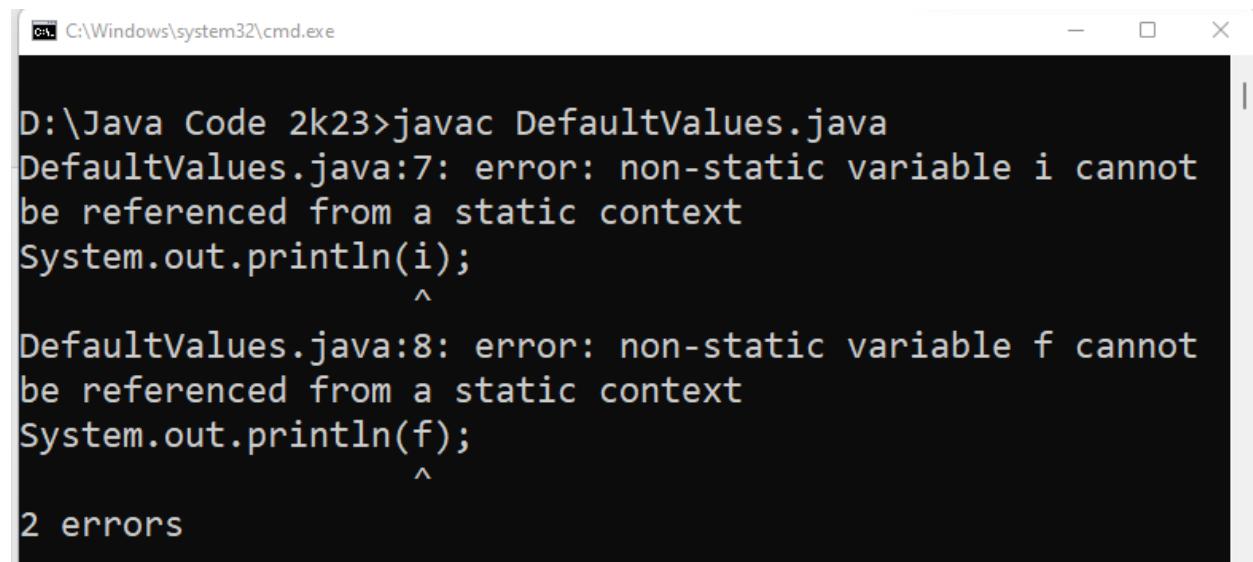
## Case 2

Declaring variables outside the main method.



The screenshot shows a Java code editor window titled "DefaultValues.java". The code is as follows:

```
1 class DefaultValues
2 {
3     int i;
4     float f;
5     public static void main(String args[])
6     {
7         System.out.println(i);
8         System.out.println(f);
9     }
10}
11
12
```



The screenshot shows a command-line interface window titled "C:\Windows\system32\cmd.exe". The user has run the command "javac DefaultValues.java". The output shows two errors:

```
D:\Java Code 2k23>javac DefaultValues.java
DefaultValues.java:7: error: non-static variable i cannot
be referenced from a static context
System.out.println(i);
           ^
DefaultValues.java:8: error: non-static variable f cannot
be referenced from a static context
System.out.println(f);
           ^
2 errors
```

**Solution:**

```
1 class DefaultValues
2 {
3     static int i;
4     static float f;
5     public static void main(String args[])
6     {
7         System.out.println(i);
8         System.out.println(f);
9     }
10 }
```

```
D:\Java Code 2k23>javac DefaultValues.java
```

```
D:\Java Code 2k23>java DefaultValues
```

```
0
```

```
0.0
```

## Java program that prints the range of primitive data types

```
(DataTypeRangeExample.java)
1  public class DataTypeRangeExample {
2      public static void main(String[] args) {
3          System.out.println("Range of primitive data types:");
4
5          System.out.println("byte: " + Byte.MIN_VALUE + " to " + Byte.MAX_VALUE);
6          System.out.println("short: " + Short.MIN_VALUE + " to " + Short.MAX_VALUE);
7          System.out.println("int: " + Integer.MIN_VALUE + " to " + Integer.MAX_VALUE);
8          System.out.println("long: " + Long.MIN_VALUE + " to " + Long.MAX_VALUE);
9
10         System.out.println("float: " + Float.MIN_VALUE + " to " + Float.MAX_VALUE);
11         System.out.println("double: " + Double.MIN_VALUE + " to " + Double.MAX_VALUE);
12
13         System.out.println("char: " + (int) Character.MIN_VALUE + " to " + (int) Character.MAX_VALUE);
14         System.out.println("boolean: " + Boolean.FALSE + " to " + Boolean.TRUE);
15
16     }
17 }
```

## Output:

```
D:\Java Code 2k23>javac DataTypeRangeExample.java

D:\Java Code 2k23>java DataTypeRangeExample
Range of primitive data types:
byte: -128 to 127
short: -32768 to 32767
int: -2147483648 to 2147483647
long: -9223372036854775808 to 9223372036854775807
float: 1.4E-45 to 3.4028235E38
double: 4.9E-324 to 1.7976931348623157E308
char: 0 to 65535
boolean: false to true
```

## **Identifiers in Java**

An identifier is any name in a Java program that can be used for identification purposes. It can be a method name, class name, variable name, or label name.

### **How many identifiers are present in the following program?**

```
class Student
{
    public static void main(String args[])
    {
        int x=10;
    }
}
```

Answer: 5

```
class Student{
    public static void main(String args[])
    {
        int x=10;
    }
}
```

Student-user-defined class name

main- method name

String-predefined class name

args and x are the variable names

### **How many identifiers are present in the following program?**

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

### **Answer: 7**

**First**-user defined class name

**main**-method name

**String**-predefined class name

**args**-variable name

**System**-Predefined class name

**out**-out is effectively an object name because it refers to a specific object of the PrintStream class.

**println()**-method name

### **Rules for Defining Java Identifiers:**

Following are the allowed characters for identifiers in Java:

A to Z

a to z

0 to 9

\$

\_ (underscore)

By mistake, if you are using any other character, you will get a compile-time error.

### **Identify the valid identifiers.**

even\_number

even#

even2

2even

### **Answer:**

even\_number      valid

even#            invalid

even2            valid

2even            invalid- identifiers cannot start with digits.

**Note:** Java identifiers are case-sensitive. Java language itself is treated as a case-sensitive programming language.

```
class Demo
{
public static void main(String args[])
{
int number =10; //Valid/Invalid
int Number =10; // Valid/Invalid
int NUMBER =20; //Valid/Invalid
}
}
```

**Note: We can differentiate with respect to the case.**

**How many characters are allowed in Java for identifiers?**

- 32
- 64
- 128
- 256
- 512
- No limit

```
class Demo
{
public static void main(String args[])
{
    int _x=5;
    int x_=6;
    int $=3;
}
```

```
int 123x=30;
int #=5;
int num=20;
int Num=30;
int NUM=40;
int x=10;
int xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx=20;
System.out.println(xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx);
}}
```

```
class Demo
{
public static void main(String args[])
{
    int _x=5;
    int x_=6;
    int $=3;
    int 123x=30; //illegal character
    int #=5;//illegal character
    int num=20;
    int Num=30;
    int NUM=40;
    //int num=50;//already defined

    int x=10;
    int xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx=20;
    System.out.println(xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx);
}
}
```

**Note :**

There is no length limit for Java identifiers, even you can take any length, but it is not recommended to take too lengthy identifiers.

Can we use the reserved word for Java identifiers?

```
class ReserveW
{
    public static void main(String args[])
    {
        int for =10;invalid
        int if =20; invalid
        int class =50;invalid
        int cat =70; valid
    }
}
```

**Note: We cannot use reserve words as identifiers.**

**CE: not a statement**

Can you tell what is the output we are going to get???

```
class Demo1
{
    public static void main(String args[])
    {
        int String =100;
        int Runnable=200;
        System.out.println(String);
        System.out.println(Runnable);
    }
}
```

}

output: 100

200

Question : Valid or not????

Answer : 100% valid

These are the predefined class or Interfaces of Java.

**Note:**

All predefined Java class names and interface names, we can use as Java identifiers. Even though it is valid, it is not good programming practice. It reduces readability and creates confusion.

**Which are the following valid/invalid Java Identifiers;TODO**

total\_num

total#

123total

total123

ca\$h

Integer

int

Int

\$\_\$\$\_\$

all@

Upes2020csf

**Answer:**

total\_num valid

total# inv

123total inv

total123 valid

ca\$h valid

Integer valid  
int inv  
Int valid  
\_\$\_\$\_ valid  
all@ inv  
Upes2020iot valid

### **Word:**

In any language-normal language or programming language, some words are reserved words.

Eg. In English

apple✓

students✓

run✓

sleep ✓

god ✓

dog ✓

odg ✗

Similarly, In java, some words are reserved words to present some meaning or functionality, such types of words are called reserve words.

### **Reserve Words in Java:**

These words can't be used for anything else because they're predefined. They can't be used as a variable name, object name, or any other identifier. There are 53 reserved terms or keywords in Java.

## How many reserved words are there in Java?

40

45

50

53

55

63

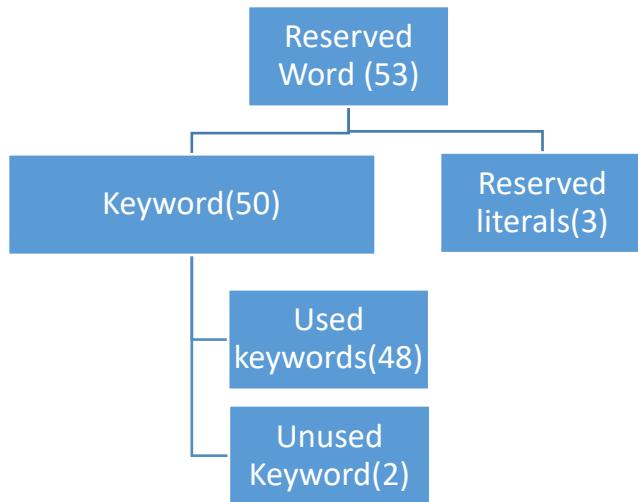
Answer: 53

|            |              |                |          |           |           |
|------------|--------------|----------------|----------|-----------|-----------|
| abstract   | assert       | <b>boolean</b> | break    | byte      | case      |
| catch      | char         | class          | const    | continue  | default   |
| double     | do           | else           | enum     | extends   | false     |
| final      | finally      | float          | for      | goto      | if        |
| implements | import       | instanceof     | int      | interface | long      |
| native     | new          | null           | package  | private   | protected |
| public     | return       | short          | static   | strictfp  | super     |
| switch     | synchronized | this           | throw    | throws    | transient |
| true       | try          | void           | volatile | while     |           |

50 (keyword)+ 3(reserve literals) **true, false and null**

50 keyword= used keyword 48(if else.....)+unused keyword 2(goto,const)

|                             |  |
|-----------------------------|--|
| <b>const</b>                | The 'const' keyword is no more supported in Java   |
| <b>goto</b>                 | The 'goto' keyword is no more supported in Java  |
| <b>true, false and null</b> | The words "true, false, null" are literals. Still, we cannot use them as identifiers in the program. |



### **Keywords for data types: 8**

1. byte
2. short
3. int
4. long
5. float
6. double
7. boolean
8. char

### **Keywords for flow control: 11**

9. if
10. else
11. switch
12. case
13. default
14. while
15. do

- 16. break
- 17. continue
- 18. return
- 19. for

**Keywords for modifiers:** **11** (Actually they are 12, default already included in flow control category)

- 20. public
- 21. private
- 22. protected
- 23. static
- 24. final
- 25. abstract
- 26. synchronized
- 27. native
- 28. strictfp- **1.2 version**
- 29. transient
- 30. volatile

**Keywords for exception handling** **6**

- 31. try
- 32. catch
- 33. finally
- 34. throw
- 35. throws
- 36. assert **1.4 version**

**Class-related keywords:** **6**

- 37. class
- 38. interface

- 39. extends
- 40. implements
- 41. package
- 42. import

### **Object related keywords: 4**

- 43. new
- 44. instanceof
- 45. super
- 46. this

### **Return type keyword:1**

- 47. void (default return type in java- void, default return type in c language-int)

### **Group of named const:1**

- 48. enum

### **Unused keyword:2**

- 49. goto- uses of goto created several problems in old languages, hence java people banned this keyword in java.
- 50. const- use final instead of const

**Note:** goto and const are unused keywords and if you are trying to use we will get compile time error.

### **Reserved word -literals:3**

- 51. true: value for Boolean data type
- 52. false: value for Boolean data type
- 53. null default value of object reference

**Note:** All 53-reserved words in Java contain only lowercase alphabet symbols.

**Note:** In Java, we have only a new keyword, and there is no deleting keyword because the destruction of useless objects is the responsibility of the garbage collector.

**Which are the following list contains Java reserve words:**

- new, delete
- goto , constant
- break,continue, return,exit
- final,finally,finalize-method
- throw, throws,thrown
- notify, notifyall
- implements, extends, imports
- sizeof, instanceof-not in java
- instanceif,strictFp
- byte,short,Int
- none of these

**Answer:**

new, delete  
goto , constant  
break,continue, return,exit  
final,finally,finalize-method  
throw, throws,thrown  
notify, notifyall  
implements, extends, imports  
sizeof, instanceof-not in java  
instanceif,strictFp  
byte,short,Int  
none of these-true

Note: Automatically they come in blue color in Java IDEs/Notepad+.

## ACCEPTING INPUT FROM USER

- Using Command Line //before java 1.5
- Using Scanner Class// popular but not very efficient
- Using GUI (Applets,servlets,jsp...):

### Using Command line arguments:

The Java command-line argument is an argument i.e. passed at the time of running the Java program.

The arguments passed from the console can be received in the Java program and it can be used as an input.

So, it provides a convenient way to check the behavior of the program for the different values. You can pass N (1,2,3 and so on) numbers of arguments from the command prompt.

```
class CommandLine1{  
    public static void main(String args[]){  
        System.out.println("Your first argument is: "+args[0]);  
    }  
}
```

| args[]  |      |      |       |   |  |  |  |  |     |
|---------|------|------|-------|---|--|--|--|--|-----|
| 0       | 1    | 2    | 3     | 4 |  |  |  |  | n-1 |
| saurabh | jain | 1234 | @@@ @ |   |  |  |  |  |     |

Save it: CommandLine1.java

Compile it: javac CommandLine1.java

**Execute it: java CommandLine1**

```
F:\Java Code 2020>java CommandLine1  
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 0  
        at CommandLine1.main(CommandLine1.java:3)
```

```
C:\> C:\Windows\System32\cmd.exe

F:\Java Code 2020>javac CommandLine1.java

F:\Java Code 2020>java CommandLine1 saurabh
Your first argument is: saurabh

F:\Java Code 2020>java CommandLine1 jain
Your first argument is: jain

F:\Java Code 2020>java CommandLine1 1234
Your first argument is: 1234

F:\Java Code 2020>java CommandLine1 @@@@@
Your first argument is: @@@@@
```

```
class CommandLine2{
public static void main(String args[]){

for(int i=0;i<args.length;i++) //you can pass any number of values
System.out.println(args[i]);

}
}
```

```
F:\Java Code 2020>javac CommandLine2.java

F:\Java Code 2020>java CommandLine2

F:\Java Code 2020>java CommandLine2 saurabh
saurabh

F:\Java Code 2020>java CommandLine2 saurabh jain
saurabh
jain

F:\Java Code 2020>java CommandLine2 saurabh jain 123
saurabh
jain
123
```

```
class AddCline
{
    public static void main(String args[])
    {
        double a =Double.parseDouble(args[0]); //convert the String Value in double
        double b =Double.parseDouble(args[1]);
        double c = a + b;
        System.out.println("Sum of a and b is :" + c);
    }
}
```

args

|      |      |
|------|------|
| 23.5 | 44.6 |
| 0    | 1    |

```
F:\Java Code 2020>java AddCline
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 0
        at AddCline.main(AddCline.java:5)

F:\Java Code 2020>java AddCline 23.5 44.6
Sum of a and b is :68.1

F:\Java Code 2020>java AddCline 23.5
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 1
        at AddCline.main(AddCline.java:6)

F:\Java Code 2020>java AddCline 23.5 23.5 45.7
Sum of a and b is :47.0
```

## **Using Scanner class:**

Using the Scanner class, we Accept input from the user in an interactive manner. A separate Scanner class is defined in the ‘java.util.Scanner’ package, which defines various methods for taking different data values from the user. Using the Scanner class in Java is the easiest way to read input in a Java program.

### **Java Scanner Input Types**

The Scanner class helps take the standard input stream in Java. So, we need some methods to extract data from the stream. Methods used for extracting data are mentioned below:

| Sr no | Method             | Description                    |
|-------|--------------------|--------------------------------|
| 1     | nextBoolean()      | Used for reading Boolean value |
| 2     | nextByte()         | Used for reading Byte value    |
| 3     | nextDouble()       | Used for reading Double value  |
| 4     | nextFloat()        | Used for reading Float value   |
| 5     | nextInt()          | Used for reading Int value     |
| 6     | nextLine(), next() | Used for reading Line value    |
| 7     | nextLong()         | Used for reading Long value    |
| 8     | nextShort()        | Used for reading Short value   |

#### **Note:**

In order to use the nextXYZ() method, we also need to create an instance of the Scanner class.

### **WAP in JAVA that takes two integer values from the user using Scanner class and print their sum.**

```
import java.util.*;
class UsingScanner
{
public static void main(String args[])
{
Scanner s=new Scanner(System.in);
int a,b;
System.out.println("Enter first number");
a=s.nextInt();
System.out.println("Enter Second number");
b=s.nextInt();
int c=a+b;
System.out.println("Addition is:"+c);
}
}
```

```
F:\Java Code 2020>java UsingScanner  
Enter first number  
23  
Enter Second number  
34  
Addition is:57
```

```
F:\Java Code 2020>java UsingScanner  
Enter first number  
aaa  
Exception in thread "main" java.util.InputMismatchException
```

```
F:\Java Code 2020>java UsingScanner  
Enter first number  
30.5  
Exception in thread "main" java.util.InputMismatchException
```

WAP in JAVA takes student information from the user and prints their record.

```
1 import java.util.*;
2 import java.lang.*;//default package
3 class UsingScanner1
4 {
5     public static void main(String args[])
6     {
7         Scanner s=new Scanner(System.in);
8         int roll;
9         double per;
10        String name;
11        System.out.println("Enter roll number");
12        roll=s.nextInt();
13        System.out.println("Enter percentage");
14        per=s.nextDouble();
15        System.out.println("Enter name");
16        name=s.next();
17        System.out.println("Roll number is:"+roll);
18        System.out.println("Percentage is:"+per);
19        System.out.println("Name is:"+name);
20    }
21 }
```

```
C:\Windows\System32\cmd.exe - □ ×

F:\1 Jan-june 2019\JAVA 2019\Java Code>javac UsingScanner1.java

F:\1 Jan-june 2019\JAVA 2019\Java Code>java UsingScanner1
Enter roll number
101
Enter percentage
91.7
Enter name
Saurabh
Roll number is:101
Percentage is:91.7
Name is:Saurabh
```

**Class Work:**

- WAP in JAVA shows arithmetic operations. (Take inputs from the user using the Scanner class).
- WAP in JAVA takes the radius as input from the user and prints the area and circumference.
- WAP to find the largest of three numbers.

**WAP in JAVA shows arithmetic operations. (Take inputs from the user using the Scanner class).**

```
1 import java.util.Scanner;
2 class MiniCalculator
3 {
4     public static void main(String args[])
5     {
6         Scanner s=new Scanner(System.in);
7         double a,b;
8         System.out.println("Enter first number");
9         a=s.nextDouble();
10        System.out.println("Enter Second number");
11        b=s.nextDouble();
12        double c=a+b;
13        double d=a-b;
14        double e=a*b;
15        double f=a/b;
16        double g=a%b;
17        System.out.println("Addition is:"+c);
18        System.out.println("Subtraction is:"+d);
19        System.out.println("Multiplication is:"+e);
20        System.out.println("Division is :" +f);
21        System.out.println("Modulo division is:" +g);
22    }
23 }
```

**Output:**

```
ca| C:\Windows\System32\cmd.exe
F:\1 Jan-june 2019\JAVA 2019\Java Code>javac MiniCalculator.java

F:\1 Jan-june 2019\JAVA 2019\Java Code>java MiniCalculator
Enter first number
45
Enter Second number
6
Addition is:51.0
Subtraction is:39.0
Multiplication is:270.0
Division is :7.5
Modulo division is:3.0
```

WAP in JAVA takes the radius as input from the user and prints the area and circumference.

```
1 import java.util.Scanner;
2 class Circle
3 {
4     public static void main(String args[])
5     {
6         int radius;
7         double area,circum;
8         Scanner s=new Scanner(System.in);
9         System.out.println("Enter Radius of circle:");
10        radius=s.nextInt();
11        area=Math.PI*Math.pow(radius,2);
12        circum=2*Math.PI*radius;
13        System.out.println("Radius of Circle is :" +radius);
14        System.out.println("Area of Circle is :" +area);
15        System.out.println("Circumference of Circle is :" +circum);
16    }
17 }
```

```
C:\ C:\Windows\System32\cmd.exe
F:\1 Jan-june 2019\JAVA 2019\Java Code>javac Circle.java

F:\1 Jan-june 2019\JAVA 2019\Java Code>java Circle
Enter Radius of circle:
7
Radius of Circle is :7
Area of Circle is :153.93804002589985
Circumference of Circle is :43.982297150257104
```

**WAP to find the largest of three numbers.**

```
import java.util.*;
class Largest
{
public static void main(String args[])
{
Scanner s=new Scanner(System.in);
System.out.println("Enter Three Integer values:");
int a=s.nextInt();
int b=s.nextInt();
int c=s.nextInt();
if(a>b && a>c)
{
System.out.println("greater number is:"+a);
}
else if (b>a && b>c)
{
System.out.println("greater number is:"+b);
}
else
{
System.out.println("greater number is:"+c);
}}}
```

```
C:\Windows\System32\cmd.exe
```

```
F:\Java Code>java Largest
Enter Three Integer values:
40
50
60
greater number is:60
```

```
F:\Java Code>java Largest
Enter Three Integer values:
60
50
40
greater number is:60
```

**3. Using GUI:** We will discuss this later. (Topics Included-Applets/Swing/ JSP/ Servlets)

## OOPs (Object-Oriented Programming System)

- **Object** means a real-world entity such as a pen, chair, table, computer, watch, etc.
- **Object-oriented programming is a methodology or paradigm used** to design a program using classes and objects.
- The main aim of object-oriented programming is to implement real-world entities, for example, objects, classes, abstraction, inheritance, polymorphism, etc.
- It simplifies the software development and maintenance.

### Note:

- **Simula** is considered the first object-oriented programming language. The programming paradigm where everything is represented as an object is known as a truly object-oriented programming language.
- **Smalltalk** is considered the first truly object-oriented programming language.
- The popular object-oriented languages are Java, C#, PHP, Python, C++, etc.

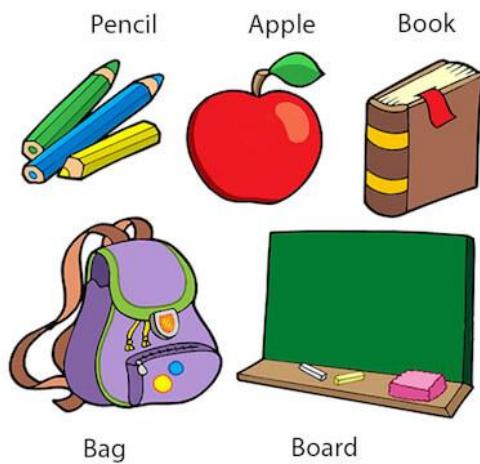
### Object:

**An object is an instance of a class.** A class is a template or blueprint from which objects are created. So, an object is the instance(result) of a class.

### Object Definitions:

- An object is a *real-world entity*.
- An object is a *runtime entity*.
- The object is an *instance of a class*.
- Object is the basic unit of object-oriented programming.
- An object is a variable of type class.
- It stores data and class methods.
- Example: Alto, Zen Estilo, Swift etc are the objects of class Car.

### Objects: Real World Examples



### **Creation of Objects:**

- Once the class is created, one or more objects can be created from the class as objects are instances of the class.
- Objects are declared as:
  - Syntax: class-name object-name= new class-name();
  - Example: Student S1=new Student();

The **new** keyword is used to allocate memory at runtime. All objects get memory in the Heap memory area.

### **Class:**

- Class is a collection of data members and member functions.
  - Member data specifies the type of data to be stored.
  - Member function acts as a mediator between user and data. They manipulate data.
- Class is a user-defined data type.
- The class implements encapsulation and abstraction.
- 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-data members
- Methods/functions
- Constructors/special member function
- Blocks
- Nested class and interface

### **Class Declaration/ Class Prototyping:**

- A class is specified with a name after the keyword class.
- The body of the class begins with the curly bracket '{' and is followed by the code containing the data members and functions. Then the class is closed with curly bracket '}' and concluded with a semi-colon (;)- Optional.
- There are different access specifiers for defining the data and functions present inside a class.
- In Java, the class can be declared as follows:
- Syntax:      class Classname
  - {
  - list of data members;
  - constructors
  - list of member function;
  - etc..
  - }
  - (optional)

## **Methods or Functions**

- A Method/function is a group of statements that performs any coherent task.
- A function contains a set of statements that are bundled within a set of curly brackets and is identified by a name called function name.
- It can be classified into two categories:
  - Predefined or In-Built functions
  - User-defined function.
- The following three things have to be done to use a function
  - **Function Prototyping**  
Syntax:  
`Return type function name (list of ordered formal arguments);  
void main(String args[])`  
int addition(int x,int y) // x and y are formal arguments
  - **Function Definition/Body**  
Syntax:  
`Return type function name (list of formal arguments)  
{  
Set of statements;//function body  
}`
  - **Function Calling**  
Syntax:  
`Object name. function name (list of ordered actual arguments);  
s.addition(3,5);//3 and 5 are actual arguments`

## **Declaration styles of method:**

1. int addition(int x, int y) –takes something, returns something
2. void addition(int x,int y)- takes something, returns nothing
3. int addition( )- takes nothing returns something
4. void addition() - takes noting, returns nothing

## **Access modifiers:**

There are two types of modifiers in Java: access modifiers and non-access modifiers.

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

There are four types of access modifiers/specifiers in the Java programming language:

- Private (Access within class)
- protected (Access within class and derived class) A and B child class
- Package (Access within the directory/folder)- It is by default in Java
- Public (Access from anywhere)

There are many non-access modifiers, such as static, abstract, synchronized, native, volatile, transient, etc. **TODO**

Java Program to illustrate how to define a class and fields(data members).

```
1  class Demo
2  {
3      int roll;
4      String name;
5      double marks;
6      public static void main(String args[])
7      {
8          Demo d1 =new Demo();
9          System.out.println(d1.roll);
10         System.out.println(d1.name);
11         System.out.println(d1.marks);
12     }
13 }
14
15
```

```
C:\Windows\System32\cmd.exe
F:\Java Code>javac Demo.java

F:\Java Code>java Demo
0
null
0.0
```

**Note:** In real-time development, we create classes and use them from another class. It is a better approach than the previous one. Let's see a simple example where we have a main() method in another class.

## main outside the class

```
1  class Demo1
2  {
3      int roll;
4      String name;
5      double marks;
6  }
7
8  class TestDemo1
9  {
10     public static void main(String args[])
11     {
12         Demo1 d1 =new Demo1();
13         System.out.println(d1.roll);
14         System.out.println(d1.name);
15         System.out.println(d1.marks);
16     }
17 }
```

If we save this program as **Demo1.java**, it will show a compile-time error.

```
C:\Windows\System32\cmd.exe
F:\Java Code>javac Demo1.java

F:\Java Code>java Demo1
Error: Main method not found in class Demo1, please define the main method as:
    public static void main(String[] args)
or a JavaFX application class must extend javafx.application.Application
```

We can have multiple classes in different Java files or a single Java file. If you define multiple classes in a single Java source file, it is a good idea to save the file name with the class name which has the **main()** method.

```
C:\Windows\System32\cmd.exe
F:\Java Code>javac TestDemo1.java

F:\Java Code>java TestDemo1
0
null
0.0
```

There are three ways to initialise the object in Java.

- By reference variable
- By method
- By constructor

### **Initialize object by reference variable**

Initialising an object means storing data in the object. Let's see a simple example where we are going to initialise the object through a reference variable.

```
1  class Demo2
2  {
3      int roll;
4      String name;
5      double marks;
6  }
7
8  class TestDemo2
9  {
10     public static void main(String args[])
11     {
12         Demo1 d1 =new Demo1();
13         //roll=20; will show compile time error
14         d1.roll=20;
15         d1.name="Saurabh";
16         d1.marks=97.6754;
17
18         //System.out.println(roll); will show error
19         System.out.println(d1.roll);
20         System.out.println(d1.name);
21         System.out.println(d1.marks);
22     }
23 }
```

#### Output:

```
C:\Windows\System32\cmd.exe
F:\Java Code>javac TestDemo2.java

F:\Java Code>java TestDemo2
20
Saurabh
97.6754
```

## Initialization through method

In this example, we are creating the two objects of the Student class and initializing the value of these objects by invoking the method.

```
1  class Student
2  {
3      int roll;
4      String name;
5      double marks;
6      public void setData(int r, String s, double m) // mutator
7      {
8          roll=r;
9          name=s;
10         marks=m;
11     }
12     public void getData() // accessor
13     {
14         System.out.println("Roll number of student:"+roll);
15         System.out.println("Name of student:"+name);
16         System.out.println("Marks of student:"+marks);
17     }
18 }
19 class TestStudent
20 {
21     public static void main(String args[])
22     {
23         Student S1=new Student(); //object create
24         S1.setData(49,"Rahul",840.67);
25         S1.getData();
26         System.out.println(); //Inserting new line
27         Student S2=new Student();
28         S2.setData(54,"Shruti",785.89);
29         S2.getData();
30     }
31 }
```

```
C:\Windows\System32\cmd.exe
```

```
F:\Java Code>javac TestStudent.java
```

```
F:\Java Code>java TestStudent
```

```
Roll number of student:49
```

```
Name of student:Rahul
```

```
Marks of student:840.67
```

```
Roll number of student:54
```

```
Name of student:Shruti
```

```
Marks of student:785.89
```

**Initialisation through a constructor:** We will discuss this later.

### Class Work

**Write a program using a function that prints the area of a rectangle(Take input from the user through the Scanner class).**

```
import java.util.Scanner;
class Rectangle
{
    int l,b,area;

    public void insertData()
    {
        Scanner s=new Scanner(System.in);
        System.out.println("Enter length");
        l=s.nextInt();
        System.out.println("Enter Width");
        b=s.nextInt();
    }
    public void printArea()
    {
        area=l*b;
        System.out.println("Area is :" +area);
    }
}

class TestRectangle
{
    public static void main(String args[])
    {
        Rectangle r=new Rectangle();
        r.insertData();
        r.printArea();
    }
}
```

```
C:\Windows\System32\cmd.exe

F:\Java Code>javac TestRectangle.java

F:\Java Code>java TestRectangle
Enter length
34
Enter Width
45
Area is :1530
```

**Java Program will demonstrate how to use a banking system, where we deposit and withdraw amounts from our account. Display the transaction details of the customer account.**

```
class Account1{
int acc_no;
String name;
double amount;
//Method to initialize the object
void insert(int a,String n,double amt){
acc_no=a;
name=n;
amount=amt;
}
//deposit method
void deposit(double amt){
amount=amount+amt;
System.out.println(amt+" deposited");
}
//withdraw method
void withdraw(double amt){
if(amount<amt){
System.out.println("Insufficient Balance");
}else{
amount=amount-amt;
System.out.println(amt+" withdrawn");
```

```
}

}

//method to check the balance of the account
void checkBalance()
{
System.out.println("Balance is: "+amount);
//method to display the values of an object
void display()
{System.out.println("Account No is:"+ acc_no);
System.out.println("Customer Name is:"+name);
}
}

//Creating a test class to deposit and withdraw amount

class TestAccount1{
public static void main(String[] args){
Account1 a1=new Account1();
a1.insert(123456,"Saurabh",50000);
a1.display();
a1.checkBalance();
a1.deposit(5000);
a1.checkBalance();
a1.withdraw(10000);
a1.checkBalance();
}}
```

C:\Windows\System32\cmd.exe

```
F:\Java Code 2020>javac TestAccount1.java

F:\Java Code 2020>java TestAccount1
Account No is:123456
Customer Name is:Saurabh
Balance is: 50000.0
5000.0 deposited
Balance is: 55000.0
10000.0 withdrawn
Balance is: 45000.0
```

Using Scanner class:

```
import java.util.*;
class Account{
long a;
String n;
double amt;
Scanner s=new Scanner(System.in);
//Method to initialize object
void insert(){
System.out.println("Enter acc_no");
long a=s.nextLong();
System.out.println("Enter name");
String n =s.next();
System.out.println("Enter amount");
double amt=s.nextDouble();
this.a=a;
this.n=n;
this.amt=amt;
}
//deposit method
void deposit(){
System.out.println("Enter amount to be deposited");
double damount=s.nextDouble();
amt=damount+amt;
System.out.println(damount+" deposited");
}
//withdraw method
void withdraw(){
System.out.println("Enter amount to be withdrawn");
double wamount=s.nextDouble();
if(amt<wamount){
System.out.println("Insufficient Balance");
}else{
amt=amt-wamount;
System.out.println(wamount+" withdrawn");
}
}
//method to check the balance of the account
void checkBalance(){System.out.println("Balance is: "+amt);}
//method to display the values of an object
void display()
{System.out.println("Account No is:"+ a);
System.out.println("Customer Name is:"+n);
}
```

```
//Creating a test class to deposit and withdraw amount

class TestAccount{
public static void main(String[] args){
Account a1=new Account();
a1.insert();
a1.display();
a1.checkBalance();
a1.deposit();
a1.checkBalance();
a1.withdraw();
a1.checkBalance();
}}
```

```
C:\Windows\System32\cmd.exe
F:\Java Code 2020>java TestAccount
Enter acc_no
12345
Enter name
Saurabh
Enter amount
50000
Account No is:12345
Customer Name is:Saurabh
Balance is: 50000.0
Enter amount to be deposited
5000
5000.0 deposited
Balance is: 55000.0
Enter amount to be withdrawn
10000
10000.0 withdrawn
Balance is: 45000.0
```

## Method Signature

A method signature consists only of the name of the method, the parameter types, and their order. The modifiers, return type, and throws clause are not part of the signature.

public static int add(int i, int j) → called method declaration

int add(int i,int j) // method prototype

add(int,int) → Method Signature

| Method                                  | Method Signature               |
|---|--------------------------------|
| • void setData(String n,double b,int c) | --setData(String, double, int) |
| • void addNum(double d,float f)         | —addNum(double,float)          |
| • public void addNum(float f, double d) | —addNum(float,double)          |

```
class MSignature  
{
```

```
    public void meth1(int i)  
    {  
        System.out.println("method 1");  
    }  
  
    public void meth2(String s)  
    {  
        System.out.println("method 2");  
    }  
    public static void main(String args[])  
    {  
        MSignature m=new MSignature(); // created one object  
    }
```

m.meth1(4); // calling method ?

m.meth2("Saurabh"); // calling method ?

```
}
```

```
C:\Windows\System32\cmd.exe
F:\Java Code 2020>javac MSignature.java
F:\Java Code 2020>java MSignature
method 1
method 2
```

Compiler is going to be maintained in the method signature table.

- **meth1(int)**
- **meth2(String)**

Is there any method **meth1**, which can take int argument, present in MSignature's method table?  
Answer:

Yes

Is there any method **meth2** that can take the String argument, present in MSignature's method table?

Answer:

**Yes**

```
class MSignature
{
    public void meth1(int i)
    {
        System.out.println("method 1");
    }

    public void meth2(String s)
    {
        System.out.println("method 2");
    }

    public static void main(String args[])
    {
        MSignature m=new MSignature();
        m.meth1(4);
        m.meth2("Saurabh");
        m.meth3(10.5); //meth3(double)
```

```
}
```

m of type MSignature

Is there any meth3 method with double type argument, present in MSignature's method table?  
Answer :

no

So following the compiler time error, you are going to get:

```
F:\Java Code 2020>javac MSignature.java
MSignature.java:18: error: cannot find symbol
    m.meth3(10.5);
           ^
      symbol:   method meth3(double)
      location: variable m of type MSignature
1 error
```

Question: Who is going to use the method signature?

Answer: compiler

Question: When the compiler will use the method signature?

Answer: while calling/invoking methods, for resolving method calls.

```
class Msign
{
    public void meth1(int i)// meth1(int)
    {
        SOP("hello");
    }
    public int meth1(int j)//meth1(int)
    {
        return 10;
    }
}
```

is it valid or not???

Answer: Not valid.

```
F:\Java Code 2020>javac Msign.java
Msign.java:7: error: method meth1(int)
is already defined in class Msign
    public int meth1(int j)
                           ^
1 error
```

Within a class, two methods with the same signature are not allowed.

## **Method Overloading:**

Two methods are said to be overloaded if and only if both methods have the same name but different argument types.

OR

Defining a function again with the same name, in the same class, and with different arguments is known as method overloading.

If we have to perform only one operation, having same name of the methods increases the **readability of the program**.

Suppose you have to perform addition of the given numbers but there can be any number of arguments, if you write the method such as a(int,int) for two parameters, and b(int,int,int) for three parameters then it may be difficult for you as well as other programmers to understand the behavior of the method because its name differs. So, we perform method overloading to figure out the program quickly.

Advantage: Method overloading **increases the readability of the program**.

```
class MOverload
{
    public void m1()
    {
        System.out.println("no args method");
    }

    public void m1(int i)
    {
        System.out.println("int args method");
    }

    public void m1( double d)
    {
        System.out.println("double args method");
    }

    public static void main(String args[])
    {
        MOverload m=new MOverload();
        m.m1(); //m1()
        m.m1(5); //m1(int)
        m.m1(7.5); //m1(double)
    }
}
```

The screenshot shows a Windows Command Prompt window titled 'C:\Windows\System32\cmd.exe'. The command 'javac MOverload.java' is run, followed by 'java MOverload'. The output shows three method definitions: 'no args method', 'int args method', and 'double args method'.

```
C:\Windows\System32\cmd.exe
F:\Java Code 2020>javac MOverload.java
F:\Java Code 2020>java MOverload
no args method
int args method
double args method
```

In overloading, the compiler always takes care of method resolution based on the reference type. Hence, overloading is also considered to compile time polymorphism, static polymorphism, and early binding.

### Different ways to overload the method

There are two ways to overload the method in java

1. By changing the number of arguments
2. By changing the data type

### BY CHANGING THE NUMBER OF ARGUMENTS

In this example, we have created two overloaded methods, the first sum method performs the addition of two numbers and the second sum method performs the addition of three numbers.

```
1. class Calculation{
2.     void sum(int a, int b) //sum(int,int)
3.     {
4.         System.out.println(a+b);
5.     }
6.     void sum(int a,int b,int c)//sum(int,int,int)
7.     {
8.         System.out.println(a+b+c);
9.     }
10.
11.    public static void main(String args[]){
12.        Calculation obj=new Calculation();
13.        obj.sum(10,10,10); // sum(int,int,int)→30
14.        obj.sum(20,20); //sum(int,int) →40
15.
16.    }
```

```
17. }
```

Output:30  
40

### BY CHANGING THE DATA TYPE OF THE ARGUMENT

In this example, we have created two overloaded methods that differs in data type. The first sum method receives two integer arguments and second sum method receives two double arguments.

```
1. class Calculation
2. {
3.     void sum(int a,int b)// sum(int,int)
4.     {
5.         System.out.println(a+b);
6.     }
7.     void sum(double a,double b)// sum(double.double)
8.     {
9.         System.out.println(a+b);
10.    }
11.    public static void main(String args[]){
12.        Calculation obj=new Calculation();
13.        obj.sum(10.5,10.5);//
14.        obj.sum(20,20);
15.
16.    }
17. }
```

Output:21.0  
40

```
1. class Calculation{
2.     int sum(int a,int b)//sum(int,int)----1
3.     {
4.         System.out.println(a+b);
5.     }
6.
7.     double sum(int a,int b)// sum(int,int)----2
8.     {
9.         System.out.println(a+b);
10.    }
11.
12.    public static void main(String args[]){
13.        Calculation Obj=new Calculation();
14.        Obj.sum(20,20); //
15.    }
16. }
```

int result=obj.sum(20,20); //Here how can java determine which sum() method should be called

In java, method overloading is not possible by changing the return type of the method because there may be ambiguity.

**Cases:**

```
class MOverCase1
{
public void m1(int i)
{
System.out.println("Int arg");
}
public void m1(float f)
{
System.out.println("Float arg");
}
public static void main(String args[])
{
MOverCase1 c=new MOverCase1();
c.m1(10);//
c.m1(20.5f);//
c.m1('a');// c.m1(97)
c.m1(10l);//long value   m1(long)
}
}
```

```
F:\Java Code 2020>javac MOverCase1.java
```

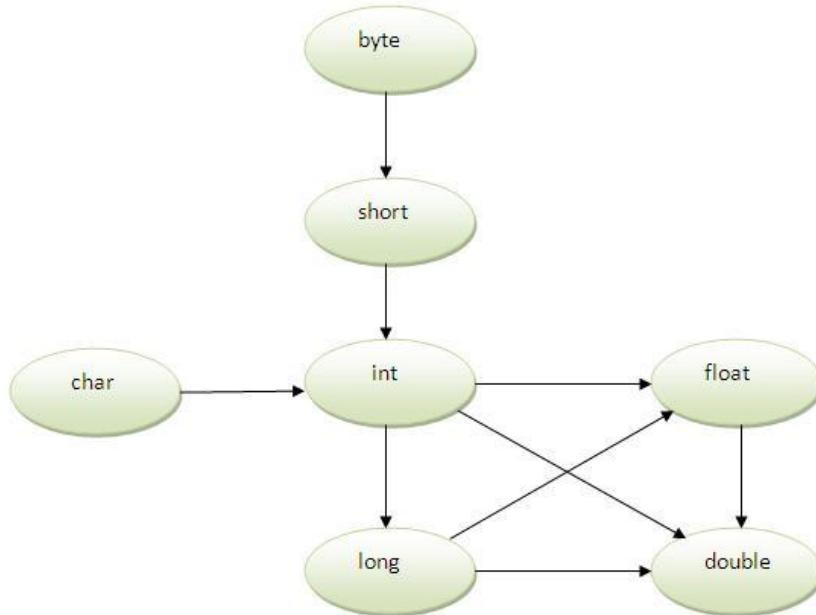
```
F:\Java Code 2020>java MOverCase1
Int arg
Float arg
Int arg
Float arg
```

```
c.m1(20.5); //Compile time error -- m1(double)
```

```
F:\Java Code 2020>javac MOverCase1.java
MOverCase1.java:18: error: no suitable method found for m1(double)
    c.m1(20.5);
               ^
        method MOverCase1.m1(int) is not applicable
            (argument mismatch; possible lossy conversion from double to int)
        method MOverCase1.m1(float) is not applicable
            (argument mismatch; possible lossy conversion from double to float)
1 error
```

## Method Overloading and Type Promotion

One type is promoted to another implicitly if no matching datatype is found. Let's understand the concept by the figure given below:



As displayed in the above diagram, byte can be promoted to short, int, long, float or double. The short datatype can be promoted to int, long, float or double. The char datatype can be promoted to int, long, float or double and so on.

## EXAMPLE OF METHOD OVERLOADING WITH TYPEPROMOTION

```
1. class Calculation{  
2.     void sum(int a,long b)      sum(int,long)  
3.     {System.out.println(a+b);  
4. }  
5.     void sum(int a,int b,int c)    sum(int,int,int)  
6.     {System.out.println(a+b+c);  
7. }  
8.  
9.     public static void main(String args[]){  
10.        Calculation obj=new Calculation();  
11.        obj.sum(20,20);//sum(int,int) -not present  
12.        obj.sum(20,20,20);  
13.  
14.    }  
15. }
```

Output:  
40  
60

If there are matching type arguments in the method, type promotion is not performed.

```
1. class Calculation{  
2.     void sum(int a,int b){System.out.println("int arg method invoked");}  
3.     void sum(long a,long b){System.out.println("long arg method invoked");}  
4.  
5.     public static void main(String args[]){  
6.        Calculation obj=new Calculation();  
7.        obj.sum(20,20);  
8.    }  
9. }
```

Output:int arg method invoked

If there are no matching type arguments in the method, and each method promotes a similar number of arguments, there will be ambiguity.

```
1. class Calculation{
2.     void sum(int a,long b){System.out.println("a method invoked");} // sum(int,long)
3.     void sum(long a,int b){System.out.println("b method invoked");} //sum(long,int)
4.
5.     public static void main(String args[]){
6.         Calculation obj=new Calculation();
7.         obj.sum(20,20);// sum(int,int)-not present
8.     }
9. }
```

a) a method invoked  
b) b method invoked  
c) both a and b  
d) compile time error

Output: Compile Time Error

**Case:**

```
class MOverCase2
{
    public void m1(String s)
    {
        System.out.println("String arg");
    }
    public void m1(Object o)
    {
        System.out.println("Object arg");
    }
    public static void main(String args[])
    {
        MOverCase2 c=new MOverCase2();
        c.m1(new Object());
        c.m1("Saurabh");
        c.m1(null);
    }
}
```

```
F:\Java Code 2020>javac MOverCase2.java
```

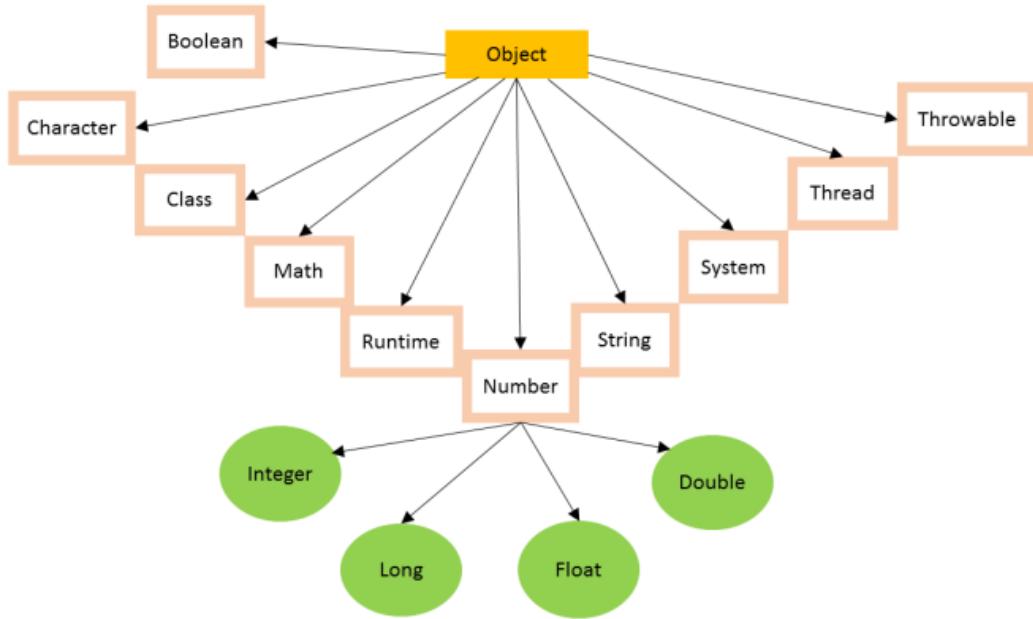
```
F:\Java Code 2020>java MOverCase2
Object arg
String arg
String arg
```

```
1 class MOverCase2
2 {
3     /*public void m1(String s)
4     {
5         System.out.println("String arg");
6     }
7 */
8     public void m1(Object o)
9     {
10        System.out.println("Object arg");
11    }
12    public static void main(String args[])
13    {
14        MOverCase2 c=new MOverCase2();
15        c.m1(new Object());
16        c.m1("Saurabh");
17        c.m1(null);
18    }
19 }
```

```
D:\Java Code 2k23>java MOverCase2
Object arg
Object arg
Object arg
```

## Object class in Java

The Object class is the parent class of all the classes in java by default. In other words, it is the topmost class of java.



## CONSTRUCTOR IN JAVA

### Constructors

- A constructor is a special member function of a class that initializes the data members of that class.
- Whenever we are creating an object, some piece of code will be executed automatically to perform initialization. This piece of code is nothing but a constructor. Hence, the main objective of the constructor is to perform initialization for the newly created object.
- A constructor is invoked automatically whenever an object of that class is created.
- In Java, a constructor is a block of codes similar to the method. It is called when an instance of the is created, and memory is allocated for the object.
- It is called a constructor because it constructs the values at the time of object creation.

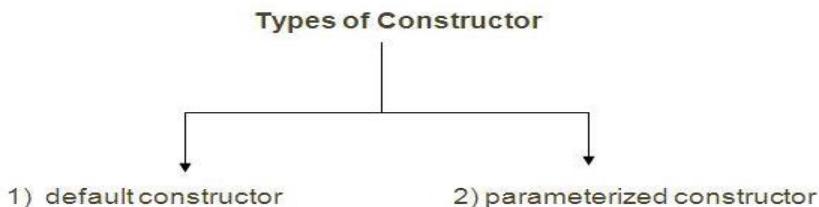
### Rules to define Constructor:

- The constructor's name must be the same as its class name.
- A constructor does not have any return type (even void also), By mistake if we declare a return type for the constructor, the compiler will treat it as a normal method.
- The only applicable modifiers for the constructor are private, protected, public, or default (PPPD), if we are trying any other modifier, we will get compile time error.-“Modifier xxxxx, is not allowed here”.
- A Java constructor cannot be abstract, static, final....., and synchronized.

### Types of constructors

There are two types of constructors:

1. Default constructor (no-args constructor)
2. Parameterized constructor



### DEFAULT CONSTRUCTOR

A constructor that has no parameter is known as a default constructor.

- Every time an object is created using a new () keyword, at least one constructor is called. It is called a default constructor.
- If we are not writing any constructor in class then the compiler will automatically generate the default constructor.
- If we are writing at least one constructor in class then the compiler will not generate the default constructor.
- Hence, a class can contain either programmer written constructor or compiler generated constructor but not both simultaneously.

**Syntax of default constructor:**

```
class_name()
{
Statement/s
}
```

## EXAMPLE OF DEFAULT CONSTRUCTOR

In this example, we are creating the no-args constructor in the ConsDemo class. It will be invoked at the time of object creation.

```
class ConsDemo
{
    ConsDemo() //default constructor
    {
        System.out.println("Default Constructor");
    }
    public static void main(String args[])
    {
        ConsDemo c=new ConsDemo();
    }
}
```

```
C:\Windows\System32\cmd.exe
```

```
F:\Java Code 2020>javac ConsDemo.java
```

```
F:\Java Code 2020>java ConsDemo
Default Constructor
```

```
1 class ConsDemo
2 {
3     ConsDemo()
4     {
5         System.out.println("Default Constructor");
6     }
7     public static void main(String args[])
8     {
9         ConsDemo c=new ConsDemo();
10        ConsDemo c1=new ConsDemo();
11    }
12 }
```

```
D:\Java Code 2k23>java ConsDemo
Default Constructor
Default Constructor
```

```
1 class ConsDemo
2 {
3     ConsDemo()
4     {
5         System.out.println("Default Constructor");
6     }
7     public static void main(String args[])
8     {
9         ConsDemo c=new ConsDemo();
10        c.ConsDemo();
11    }
12 }
```

```
D:\Java Code 2k23>javac ConsDemo.java
ConsDemo.java:11: error: cannot find symbol
c.ConsDemo();
^
      symbol:   method ConsDemo()
      location: variable c of type ConsDemo
1 error
```

### The behavior of the Constructor with access modifiers.

Example

```
class ConsDemo
{
    private ConsDemo()
    {
        System.out.println("Default Constructor");
    }
    public static void main(String args[])
    {
        ConsDemo c=new ConsDemo();
    }
}
```

```
F:\Java Code 2020>javac ConsDemo.java
```

```
F:\Java Code 2020>java ConsDemo
Default Constructor
```

```
class ConsDemo
{
    public ConsDemo()
    {
        System.out.println("Default Constructor");
    }
    public static void main(String args[])
    {
        ConsDemo c=new ConsDemo();
    }
}
```

```
F:\Java Code 2020>javac ConsDemo.java
```

```
F:\Java Code 2020>java ConsDemo
Default Constructor
```

```
class ConsDemo
{
    final ConsDemo()
    {
        System.out.println("Default Constructor");
    }
    public static void main(String args[])
    {
        ConsDemo c=new ConsDemo();
    }
}
```

```
F:\Java Code 2020>javac ConsDemo.java
ConsDemo.java:3: error: modifier final not allowed here
    final ConsDemo()
                  ^
1 error
```

```
class ConsDemo
{
    transient ConsDemo()
    {
        System.out.println("Default Constructor");
    }
    public static void main(String args[])
    {
        ConsDemo c=new ConsDemo();
    }
}
```

```
F:\Java Code 2020>javac ConsDemo.java
ConsDemo.java:3: error: modifier transient not allowed here
    transient ConsDemo()
                  ^
1 error
```

### The behavior of the Constructor with return type:

```
class ConsDemo
{
    void ConsDemo()//This is a method, not a constructor
    {
        System.out.println("Default Constructor");
    }
    public static void main(String args[])
    {
        ConsDemo c=new ConsDemo();
    }
}
```

```
C:\> C:\Windows\System32\cmd.exe  
F:\Java Code 2020>javac ConsDemo.java  
F:\Java Code 2020>java ConsDemo  
F:\Java Code 2020>
```

void ConsDemo()- Compiler treated as a normal method, not a constructor.

```
class ConsDemo  
{  
    void ConsDemo()  
    {  
        System.out.println("Default Constructor");  
    }  
    public static void main(String args[])  
    {  
        ConsDemo c=new ConsDemo();  
        c.ConsDemo();//explicit call  
    }  
}
```

```
F:\Java Code 2020>java ConsDemo  
Default Constructor
```

## EXAMPLE OF A DEFAULT CONSTRUCTOR THAT DISPLAYS THE DEFAULT VALUES

```
class ConsDemo1
{
    int id;
    String name;
    double marks;
    ConsDemo1()
    {
    }
    void display()
    {
        System.out.println(id+" "+name+" "+marks);
    }
    public static void main(String args[])
    {
        ConsDemo1 c1=new ConsDemo1();
        ConsDemo1 c2=new ConsDemo1();
        c1.display();
        c2.display();
    }
}
```

```
F:\Java Code 2020>javac ConsDemo1.java
```

```
F:\Java Code 2020>java ConsDemo1
0 null 0.0
0 null 0.0
```

Here 0 ,null and 0.0 values are provided by the default constructor.

## NO USER-DEFINED CONSTRUCTOR IS PRESENT IN THIS CLASS

```
1 public class ConsDemo1
2 {
3
4     int id;
5     String name;
6     double marks;
7
8     void display()
9     {
10        System.out.println(id+" "+name+" "+marks);
11    }
12    public static void main(String args[])
13    {
14        ConsDemo1 c1=new ConsDemo1();
15        ConsDemo1 c2=new ConsDemo1();
16        c1.display();
17        c2.display();
18    }
19 }
```

```
D:\Java Code 2k23>javac ConsDemo1.java
```

```
D:\Java Code 2k23>java ConsDemo1
0 null 0.0
0 null 0.0
```

Here **0 ,null and 0.0** values are provided by the compiler-generated default constructor.

In this class, we are not creating any constructor, so the compiler generates a default constructor.

### Note:

If we are not writing any constructor in class, then the compiler will automatically generate the default constructor.

If we are writing at least one constructor in class then the compiler will not generate the default constructor.

## **PARAMETERIZED CONSTRUCTOR**

A constructor that has parameters is known as a parameterized constructor. Parameterized constructor is used to provide different values to the distinct objects.

## **EXAMPLE OF PARAMETERIZED CONSTRUCTOR**

In this example, we have created the constructor of this class that has three parameters. We can have any number of parameters in the constructor.

```
class ConsDemo2
{
    int id;
    String name;
    double marks;
    ConsDemo2(int i, String n, double m)
    {
        id = i;
        name = n;
        marks = m;
    }
    void display()
    {
        System.out.println(id + " " + name + " " + marks);
    }
    public static void main(String args[])
    {
        ConsDemo2 c1 = new ConsDemo2(11, "saurabh", 89.89);
        c1.display();
        ConsDemo2 c2 = new ConsDemo2(22, "rohit", 87.99);
        c2.display();
    }
}
```

```
C:\Windows\System32\cmd.exe
```

```
F:\Java Code 2020>javac ConsDemo2.java  
F:\Java Code 2020>java ConsDemo2  
11 saurabh 89.89  
22 rohit 87.99
```

## CONSTRUCTOR OVERLOADING

Constructor overloading in Java is a concept that allows a class to have more than one constructor method with the same name but with different parameters (in terms of their number, type, or both). When you overload constructors, you provide multiple ways to initialize objects of a class, allowing for greater flexibility in how objects are created.

### EXAMPLE OF CONSTRUCTOR OVERLOADING

```
public class ConsOverload  
{  
    int id;  
    String name;  
    int age;  
    ConsOverload(int i,String n)  
        { //constructor with 2 parameters  
        id = i;  
        name = n;  
        }  
    ConsOverload(int i,String n,int a)  
        { //constructor with 3 parameters  
        id = i;  
        name = n;  
        age=a;  
        }  
    void display(){System.out.println(id+" "+name+" "+age);}
```

```
public static void main(String args[]){
    ConsOverload s1 = new ConsOverload(111,"Karan");
    ConsOverload s2 = new ConsOverload(222,"Aryan",25);
    s1.display();
    s2.display();
}
```

```
F:\Java Code 2020>javac ConsOverload.java

F:\Java Code 2020>java ConsOverload
111 Karan 0
222 Aryan 25
```

```
class ConsOverload
{
    int id;
    String name;
    int age;
    ConsOverload(int i,String n)
        { //constructor with 2 parameters
        id = i;
        name = n;
        }
    ConsOverload(int i,String n,int a)
        { //constructor with 3 parameters
        id = i;
        name = n;
        age=a;
        }
    void display(){System.out.println(id+" "+name+" "+age);}

    public static void main(String args[]){
        ConsOverload s = new ConsOverload(); // output??
        ConsOverload s1 = new ConsOverload(111,"Karan");
        ConsOverload s2 = new ConsOverload(222,"Aryan",25);
        s1.display();
        s2.display();
    }
}
```

```
C:\> C:\Windows\system32\cmd.exe
^
constructor ConsOverload.ConsOverloa
d(int,String) is not applicable
(actual and formal argument lists
differ in length)
constructor ConsOverload.ConsOverloa
d(int,String,int) is not applicable
(actual and formal argument lists
differ in length)
1 error
```

### Difference between constructor and method

There are many differences between constructors and methods. They are given below.

| Constructor   | Method  |
|---|---|
| A constructor is used to initialize the state of an object.                         | The method is used to expose the behavior of an object.       |
| The constructor must not have a return type.  | The method must have a return type.                           |
| The constructor is invoked implicitly.  | The method is invoked explicitly.                             |
| The java compiler provides a default constructor if you don't have any constructor. | The method is not provided by the compiler in any case.       |
| The constructor's name must be the same as the class name.                          | The method name may or may not be the same as the class name. |

## INHERITANCE IN JAVA

Inheritance can be defined as the process where one class acquires the properties (methods and fields) of another.

The class that inherits the properties of others is known as subclass (derived class, child class) and the class whose properties are inherited is known as superclass (base class, parent class).

Inheritance represents the **IS-A relationship**.

### USE OF INHERITANCE

- For Method Overriding.
- For Code Reusability.

### SYNTAX OF INHERITANCE

```
class Subclass-name extends Superclass-name
{   //methods and fields
}
```

Example:

```
class Dog extends Animal
```

Ex:

```
class Child extends Parent
{
Statement/s
}
```

**extends** is the keyword used to inherit the properties of a class.

Example:(Without extends keyword)

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

class Programmer {
    int bonus=10000;

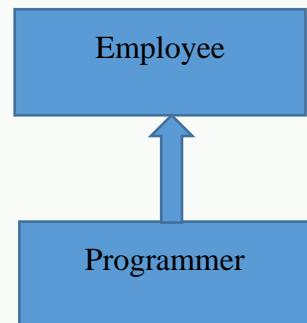
    public static void main(String args[]){
        Programmer p=new Programmer();
        System.out.println("Programmer salary is:"+p.salary);
        System.out.println("Bonus of Programmer is:"+p.bonus);
    }
}
```

```
C:\Windows\System32\cmd.exe
F:\Java Code 2020>javac Programmer.java
Programmer.java:10: error: cannot find symbol
    System.out.println("Programmer salary is:"+p.salary);
                                         ^
symbol:   variable salary
location: variable p of type Programmer
1 error
```

After using the keyword **extends**

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

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);
}
```

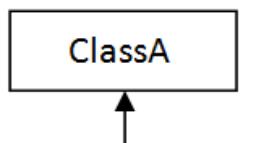
```
F:\Java Code 2020>javac Programmer.java

F:\Java Code 2020>java Programmer
Programmer salary is:40000.0
Bonus of Programmer is:10000
```

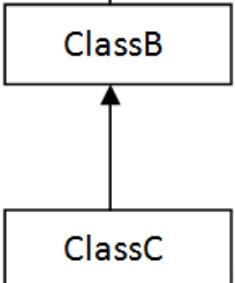
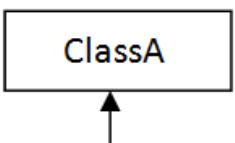
In the above example, Programmer object p can access the field of own class as well as of Employee class i.e. **code reusability**.

## TYPES OF INHERITANCE

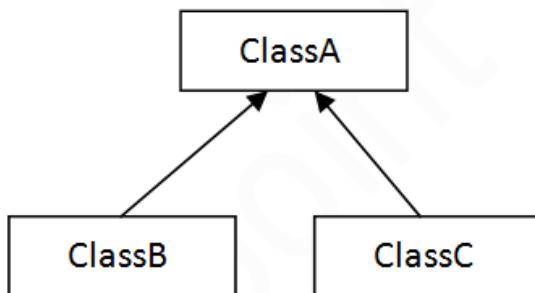
There are various types of inheritance as demonstrated below.



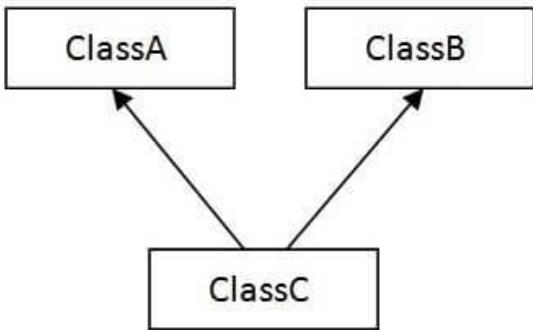
1) Single



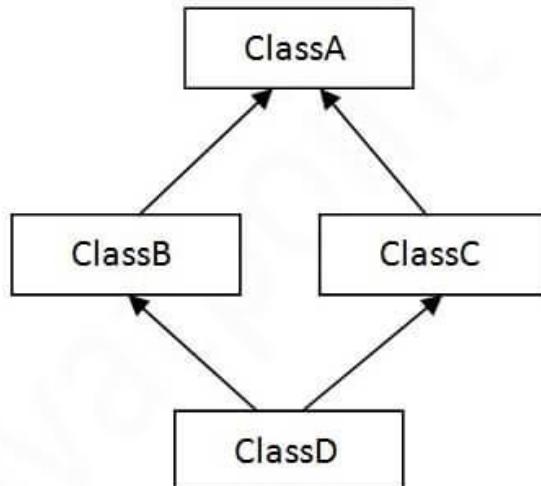
2) Multilevel



3) Hierarchical



4) Multiple



5) Hybrid

When one class inherits multiple classes, it is known as multiple inheritances.

Multiple inheritance is not supported in Java through the class.

## WHY MULTIPLE INHERITANCE IS NOT SUPPORTED IN JAVA?

To reduce the complexity and simplify the language, multiple inheritances are not supported in Java.

example:

```
class A{
void message(){System.out.println("Hello");}
}

class B{
void message(){System.out.println("Welcome");}
}

class C extends A,B{
public static void main(String args[]){
C obj=new C();
obj.message();//Now which message() method would be invoked?
}
}
```

Output: **Compile Time Error**

**Example:**

```
class Parent
{
public void m1()
{
System.out.println("Parent Method");
}
}
```

```
class Child extends Parent
{
public void m2()
{
System.out.println("Child Method");
}
}
```

```
class TestPC
{
public static void main(String args[])
{
    Parent p=new Parent();
    p.m1();
    p.m2();

    Child c = new Child();
    c.m1();
    c.m2();
}
```

```
    Parent p1=new Child();
    p1.m1();
    p1.m2();
```

```
    Child c1 = new Parent();
    c1.m1();
    c1.m2();
}
```

**Case 0:**

```
Parent p=new Parent();
p.m1();
```

```
F:\Java Code 2020>javac TestPC.java
```

```
F:\Java Code 2020>java TestPC
Parent Method
```

**Case 1:**

```
Parent p=new Parent();
p.m2();
```

```
F:\Java Code 2020>javac TestPC.java
TestPC.java:22: error: cannot find symbol
  p.m2();
  ^
symbol:   method m2()
location: variable p of type Parent
1 error
```

**Case 3:**

```
Child c = new Child();
c.m1();
c.m2();
```

Whatever the parent class has by default available to the child, means child class reference can call both parent and child class methods.

```
F:\Java Code 2020>javac TestPC.java
```

```
F:\Java Code 2020>java TestPC  
Parent Method  
Child Method
```

**Case 4:**

```
Parent p1=new Child();-----1  
p1.m1();
```

```
F:\Java Code 2020>javac TestPC.java
```

```
F:\Java Code 2020>java TestPC  
Parent Method
```

**Case 5:**

```
Parent p1=new Child(); ----1  
p1.m2();
```

Whatever the child class has by default is not available to the parent class (**Case 5, Case 1**), hence parent class reference can call only parent class methods(**case 0,case 4**), and cannot call child class methods(**Case 5,Case 1**).

Parent class reference can be used to hold child class objects(**case 4 -----1,case 5---1**).By using that reference we can call only parent class methods(**case 4**) but we cannot call child class methods(**case 5**).

```
F:\Java Code 2020>javac TestPC.java
TestPC.java:22: error: cannot find symbol
    p1.m2();
           ^
      symbol:   method m2()
     location: variable p1 of type Parent
1 error
```

**Case 6:**

```
Child c1 = new Parent();
```

We can't use child class references to hold parent class objects(**Case 6**).

```
F:\Java Code 2020>javac TestPC.java
TestPC.java:21: error: incompatible types: Parent
cannot be converted to Child
    Child c1 = new Parent();
                           ^
                           ^
```

**Object class in Java:**

- Object class is present in `java.lang` package. Every class in Java is directly or indirectly derived from the Object class.
- If a Class does not extend any other class, then it is a direct child class of Object and if extends another class then it is indirectly derived. Therefore, the Object class methods are available to all Java classes. Hence Object class acts as a root of the inheritance hierarchy in any Java Program.
- The common functionality, which is required for any java class, is defined in the Object class and by keeping that class as a superclass, its functionality by default is available to every java class.

## **Method Overriding in Java**

Defining the method again with the same name and parameters but in different classes is called method Overriding. If subclass (child class) has the same method as declared in the parent class, it is known as **method overriding**.

In other words, if a subclass provides the specific implementation of the method that has been provided by one of its parent classes, it is known as Method Overriding. Also known as Runtime Polymorphism/Dynamic polymorphism/Late binding.

Overriding method resolution is also known as “Dynamic Method Dispatch”

## **ADVANTAGE OF JAVA METHOD OVERRIDING**

- Method Overriding is used to provide a specific implementation of a method that is already provided by its super class.
- Method Overriding is used for Runtime Polymorphism.

## **RULES FOR 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).

## **UNDERSTANDING THE PROBLEM WITHOUT METHOD OVERRIDING**

Let's understand the problem that we may face in the program if we don't use method overriding.

```
1. class Vehicle{  
2.     void run()  
3.     {  
4.         System.out.println("Vehicle is running");}  
5.     }  
6. class Bike extends Vehicle  
7. {  
8.     public static void main(String args[]){  
9.         Bike obj = new Bike();  
10.        obj.run();  
11.    }  
12. }
```

**Output:** The vehicle is running

The problem is that I have to provide a specific implementation of the run() method in a subclass which is why we use method overriding.

In this example, we have defined the run method in the subclass as defined in the parent class but it has some specific implementation. The name and parameter of the method are the same and there is an IS-A relationship between the classes, so there is method overriding.

```
1. class Vehicle{  
2.     void run(){System.out.println("Vehicle is running");}  
3. }  
  
4. class Bike extends Vehicle{  
5.     void run(){System.out.println("Bike is running safely");}  
6. }  
7.  
8. public static void main(String args[]){  
9.     Bike obj = new Bike();  
10.    obj.run();  
11. }
```

**Output:** Bike is running safely

### A REAL EXAMPLE OF A JAVA METHOD OVERRIDING

Consider a scenario, Bank is a class that provides functionality to get a rate of interest. But, the rate of interest varies according to banks. For example, SBI, ICICI, and AXIS banks could provide 8%, 7% and 9% rate of interest.

```
1. class Bank{  
2.     int getRateOfInterest(){return 0;}  
3. }  
4.  
5. class SBI extends Bank{  
6.     int getRateOfInterest(){return 8;}  
7. }  
8.
```

```
9. class ICICI extends Bank{  
10. int getRateOfInterest(){return 7;}  
11. }  
12. class AXIS extends Bank{  
13. int getRateOfInterest(){return 9;}  
14. }  
15.  
16. class Test{  
17. public static void main(String args[]){  
18. SBI s=new SBI();  
19. ICICI i=new ICICI();  
20. AXIS a=new AXIS();  
21. System.out.println("SBI Rate of Interest: "+s.getRateOfInterest());  
22. System.out.println("ICICI Rate of Interest: "+i.getRateOfInterest());  
23. System.out.println("AXIS Rate of Interest: "+a.getRateOfInterest());  
24. }  
25. }
```

**Output:**

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

## instanceof operator

The **java instanceof operator** is used to test whether the object is an instance of the specified type (class or subclass or interface).

The instanceof in Java is also known as type *comparison operator* because it compares the instance with type. It returns either true or false. If we apply the instanceof operator with any variable that has null value, it returns false.

---

```
1  class Animal {  
2  }  
3  
4  class Mammal extends Animal {  
5  }  
6  
7  class Reptile extends Animal {  
8  }  
9  
10 public class Dog extends Mammal {  
11  
12     public static void main(String args[]) {  
13         Animal a = new Animal();  
14         Mammal m = new Mammal();  
15         Dog d = new Dog();  
16  
17         System.out.println(m instanceof Animal);  
18         System.out.println(d instanceof Mammal);  
19         System.out.println(d instanceof Animal);  
20     }  
21 }
```

---

## OUTPUT:

```
C:\Windows\System32\cmd.exe  
F:\Java Code>javac Dog.java  
  
F:\Java Code>java Dog  
true  
true  
true
```

## **SUPER KEYWORD**

The **super** is a reference variable that is used to refer to an immediate parent class object. Whenever you create the instance of a subclass, an instance of the parent class is created implicitly i.e., referred to by the super reference variable.

### **USAGE OF SUPER KEYWORD**

1. super is used to refer immediate parent class instance variable.
2. super() is used to invoke the immediate parent class constructor.
3. super is used to invoke the immediate parent class method.

#### **1) SUPER IS USED TO REFER IMMEDIATE PARENT CLASS INSTANCE VARIABLE.**

##### **The problem without super keyword**

```
1. class Vehicle{  
2.     int speed=50; // parent instance variable  
3. }  
4.  
5. class Bike extends Vehicle{  
6.     int speed=100; // child instance variable  
7.  
8.     void display(){  
9.         System.out.println(speed);  
10.    }  
11.    public static void main(String args[]){  
12.        Bike b=new Bike();  
13.        b.display();  
14.  
15.    }  
16. }
```

Output:100

In the above example Vehicle and Bike, both classes have a common property speed. An instance variable of the current class is referred to by instance by default, but I must refer parent class instance variable which is why we use the super keyword to differentiate between the parent class instance variable and the current class instance variable.

### Solution by super keyword

```
1. //example of super keyword
2. class Vehicle{
3.     int speed=50;
4. }
5.
6. class Bike extends Vehicle{
7.     int speed=100;
8.
9.     void display(){
10.     System.out.println(super.speed);//will print speed of Vehicle now //50
11. }
12. public static void main(String args[]){
13.     Bike b=new Bike();
14.     b.display();
15. }
16. }
```

Output:50

### 2) SUPER IS USED TO INVOKE THE PARENT CLASS CONSTRUCTOR.

The super keyword can also be used to invoke the parent class constructor as given below:

```
1. class Vehicle{
2.     Vehicle(){System.out.println("Vehicle is created");}
3. }
4.
5. class Bike extends Vehicle{
6.     Bike(){
7.         super();//will invoke parent class constructor
8.         System.out.println("Bike is created");
9.     }
10.    public static void main(String args[]){
11.        Bike b=new Bike();
12.
13.    }
14. }
```

Output: Vehicle is created  
Bike is created

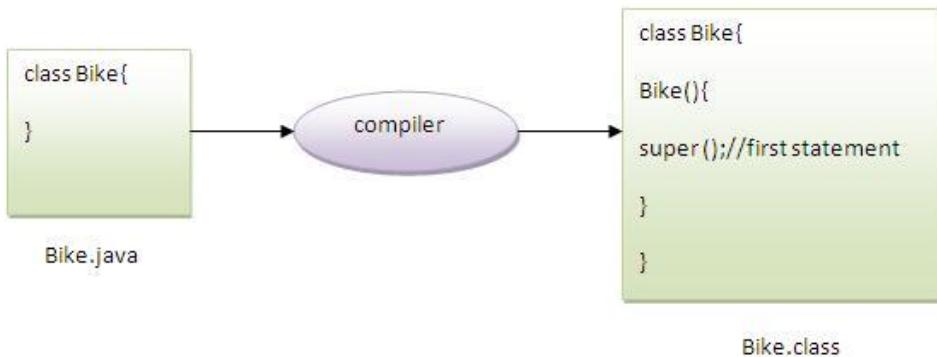
## EXAMPLE OF SUPER KEYWORD WHERE SUPER() IS PROVIDED BY THE COMPILER IMPLICITLY.

```
1  class Vehicle{
2      Vehicle(){System.out.println("Vehicle is created");}
3  }
4
5  class Bike extends Vehicle{
6      Bike(){
7          System.out.println("Bike is created");
8      }
9  public static void main(String args[]){
10     Bike b=new Bike();
11
12 }
13 }
```

```
C:\Windows\System32\cmd.exe
F:\Java Code>javac Bike.java
F:\Java Code>java Bike
Vehicle is created
Bike is created
```

### Reason:

As we know well the default constructor is provided by the compiler automatically, but it also adds super () for the first statement. If you are creating your own constructor and you don't have either this () or super() as the first statement, the compiler will provide super() as the first statement of the constructor.



```

1  class Vehicle{
2      Vehicle(){System.out.println("Vehicle is created");}
3  }
4
5      class Bike extends Vehicle{
6          Bike(){
7              System.out.println("Bike is created");
8              super();
9          }
10     public static void main(String args[]){
11         Bike b=new Bike();
12     }
13 }
14 }
15

```

```

F:\Java Code>javac Bike.java
Bike.java:8: error: call to super must be first statement in constructor
        super();
               ^
1 error

```

### 3) SUPER CAN BE USED TO INVOKE THE PARENT CLASS METHOD.

The super keyword can also be used to invoke the parent class method. It should be used in case the subclass contains the same method as the parent class as in the example given below:

```
1  class Person {
2      void message() {
3          System.out.println("Welcome");
4      }
5  }
6
7  class Student_Super extends Person {
8      void message() { // Overridden method
9          System.out.println("Welcome to Java");
10     }
11
12     void display() {
13         message(); // Calls the overridden method in Student class
14         super.message(); // Calls the method from the Person class
15     }
16
17     public static void main(String args[]) {
18         Student_Super s = new Student_Super();
19         s.display();
20     }
21 }
```

#### Output:

```
D:\1 UPES Data\Java Code>java Student_Super
Welcome to Java
Welcome
```

In the above example, Student and Person, both classes have the message () method. If we call the message() method from the Student class, it will call the message() method of the Student class, not of the Person class, because priority is given to local.

#### Note:Why does message(); work inside display() without an object?

In the display() method of Student, message(); is called **directly**, without an object. This works because display() and message() both belong to the same instance of Student. When calling a non-static method inside another non-static method of the same class, the method is implicitly invoked using this.message();.

If there is no method in the subclass as a parent, there is no need to use super. In the example given below, the message() method is invoked from the Student class, but the Student class does not have a message() method, so you can directly call the message() method.

### **Program in case super is not required.**

```
class Person {  
    void message() {  
        System.out.println("welcome");  
    }  
}  
  
class Student extends Person {  
    void display() {  
        message(); // Calls the inherited message() method from Person class  
    }  
  
    public static void main(String args[]) {  
        Student s = new Student();  
        s.display();  
    }  
}
```

### **Output:**

welcome

# STATIC KEYWORD

The **static keyword** in Java is used for memory management mainly. We can apply Java static keywords with variables, methods, blocks, and nested classes. The static keyword belongs to the class rather than an instance of the class.

The static can be:

1. variable (also known as class variable)
2. method (also known as class method)
3. block
4. nested class (we will discuss later)

## JAVA STATIC VARIABLE

If you declare any variable as static, it is known static variable.

- The static variable can be used to refer to the common property of all objects (that is not unique for each object) e.g., company name of employees, college name of students, etc.
- The static variable gets memory only once in the class area at the time of class loading.

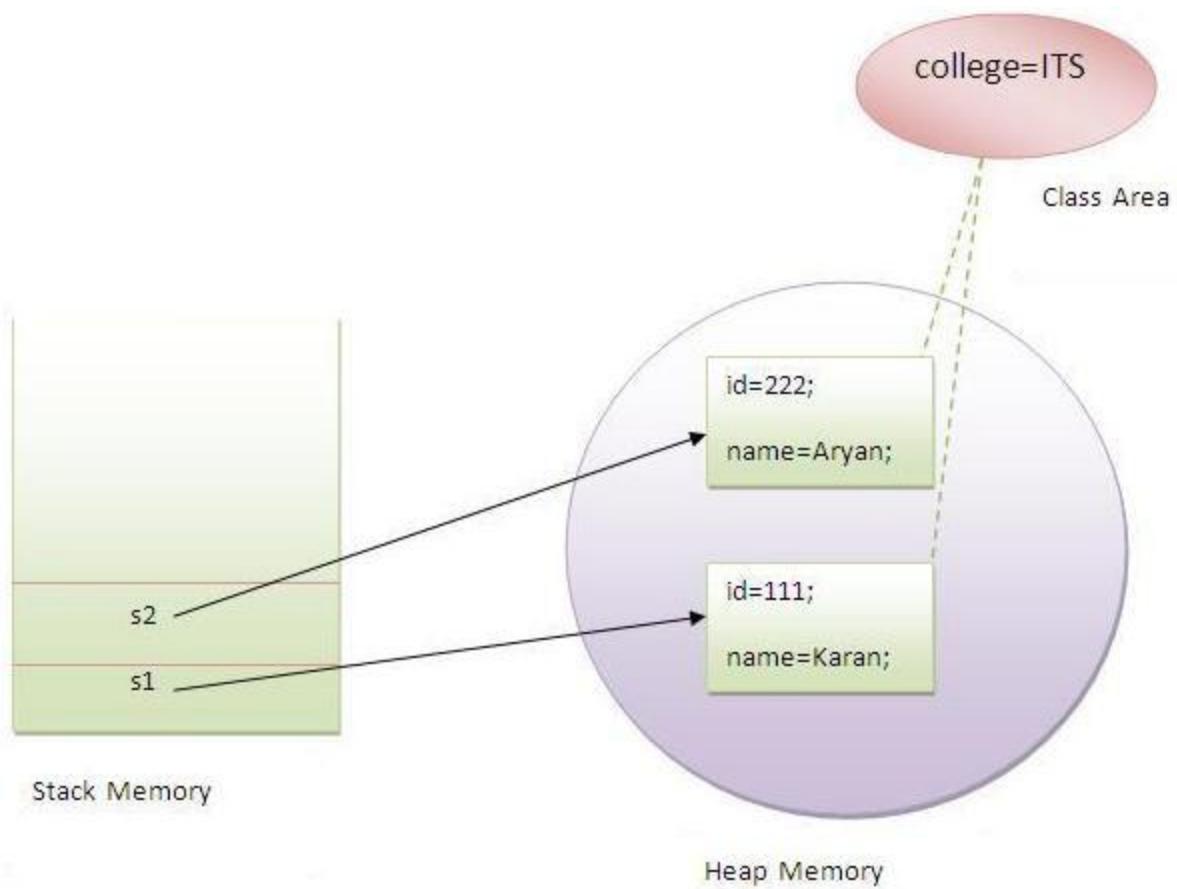
### Advantage of static variable

It makes your program **memory efficient** (i.e. it saves memory).

### The understanding problem without static variable

```
class Student{  
    int rollno;  
    String name;  
    static String college="ITS";  
}
```

Suppose there are 500 students in my college, now all instance data members will get memory each time when object is created. All student have its unique rollno and name so instance data member is good. Here, college refers to the common property of all objects. If we make it static, this field will get memory only once.



### **Example of static variable**

```
1. //Program of static variable
2.
3. class Student{
4.     int rollno;
5.     String name;
6.     static String college ="UPES"; //in class memory
7.
8.     Student(int r,String n){
9.         rollno = r;
10.        name = n;
11.    }
12.    void display (){System.out.println(rollno+" "+name+" "+college);}
13.    public static void main(String args[]){
14.        Student s1 = new Student(111,"Karan");
15.        Student s2 = new Student(222,"Aryan");
16.        s1.display();
17.        s2.display();
18.    }
19. }
```

Output:111 Karan UPES  
222 Aryan UPES

### **JAVA STATIC METHOD**

If you apply a static keyword with any method, it is known as a static method.

- A static method belongs to the class rather than the object of a class.
- A static method can be invoked without the need for creating an instance of a class.
- static method can access static data member and can change their value of it.

//Program of changing the common property of all objects (static field).

```
class Student
{
int rollno;
String name;
static String college = "ITS";
static void change(){
college = "UPES";

}
Student(int r, String n){
rollno = r;
name = n;
}
```

```
void display (){System.out.println(rollno+" "+name+" "+college);}
public static void main(String args[]){
    Student.change();

    Student s1 = new Student (111,"Karan");
    Student s2 = new Student (222,"Aryan");
    Student s3 = new Student (333,"Arjun");
    s1.display();
    s2.display();
    s3.display();
}
```

Output:

```
111 Karan UPES
222 Aryan UPES
333 Arjun UPES
```

## JAVA STATIC BLOCK

- Is used to initialize the static data member.
- It is executed before main method at the time of classloading.

Syntax:

```
static{
    statements.....
}
```

1. class A2{
2. static
3. {
4. System.out.println("static block is invoked");
5. }
6.
7. public static void main(String args[]){
8. System.out.println("Hello main");
9. }
10. }

Output: static block is invoked  
Hello main

## **Can we execute a program without main() method?**

Ans) Yes, one of the way is static block but in previous version of JDK not in JDK 1.7.

```
1. class A3{  
2.     static  
3.     {  
4.         System.out.println("static block is invoked");  
5.         System.exit(0);  
6.     }  
7. }
```

Output:static block is invoked (if not JDK7)

In JDK7 and above, output will be:

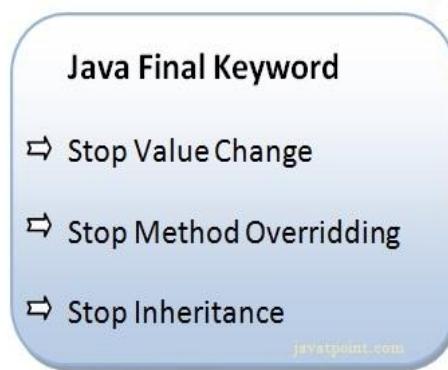
Output:Error: Main method not found in class A3, please define the main method as:  
public static void main(String[] args)

## FINAL KEYWORD IN JAVA

The **final keyword** in Java is used to restrict the user. The final keyword can be used in many contexts. The final can be:

1. variable
2. method
3. class

The final keyword can be applied to the variables, a final variable that has no value is called a blank final variable or uninitialized final variable. It can be initialized in the constructor only. The blank final variable can be static also which will be initialized in the static block only.



### 1) FINAL VARIABLE

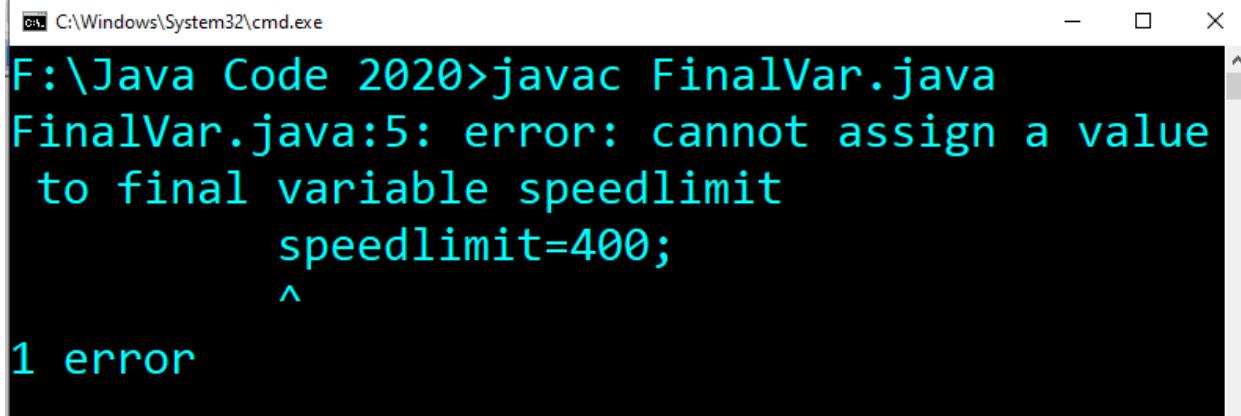
If you make any variable as final, you cannot change the value of final variable (It will be constant).

### EXAMPLE OF FINAL VARIABLE

There is a final variable speed limit, we are going to change the value of this variable, but It can't be changed because final variable once assigned a value can never be changed.

```
class FinalVar{  
  
    final int speedlimit=90;//final variable  
    void run(){  
        speedlimit=400;  
    }  
    public static void main(String args[]){  
        FinalVar obj=new FinalVar();  
        obj.run();  
    }  
}
```

```
}
```



```
C:\Windows\System32\cmd.exe
F:\Java Code 2020>javac FinalVar.java
FinalVar.java:5: error: cannot assign a value
to final variable speedlimit
    speedlimit=400;
               ^
1 error
```

## 2) FINAL METHOD

If you make any method as final, you cannot override it.

### EXAMPLE OF THE FINAL METHOD

```
class FinalMethod
{
    final void run()
    {
        System.out.println("running");
    }
}
class FinalTest extends FinalMethod {
    void run()
    {
        System.out.println("running safely with 100kmph");
    }

    public static void main(String args[]){
        FinalTest f= new FinalTest();
        f.run();
    }
}
```

```
F:\Java Code 2020>javac FinalTest.java
FinalTest.java:10: error: run() in FinalTest
cannot override run() in FinalMethod
        void run()
                  ^
  overridden method is final
1 error
```

### 3) FINAL CLASS

If you make any class as final, you cannot extend it.

#### EXAMPLE OF FINAL CLASS

```
1. final class Bike{
2.
3. } /base
4.
5. class Honda extends Bike{
6.     void run(){System.out.println("running safely with 100kmph");}
7.
8. public static void main(String args[]){
9.     Honda honda= new Honda();
10.    honda.run();
11. }
12. }
```

Output:Compile Time Error

Ques: Can we inherit the final method???

Ans) Yes, final method is inherited but you cannot override it. For Example:

```
1. class Bike{
2.     final void run(){System.out.println("running...");}
3. }
4. class Honda extends Bike{
5.     public static void main(String args[]){
6.         new Honda().run();
7.     }
8. }
```

Output:running...

## **Abstract:**

The abstract is the modifier applicable for **classes and methods** but not for variables.

### **Abstract method**

- A method declared as abstract and did not have an implementation (body) is known as an abstract method. i.e. abstract method can have only method declaration but not implementation. Hence every abstract method declaration should compulsorily end with ;(semicolon).
- The child class is responsible for providing the implementation of parent class abstract methods.
- By declaring abstract methods in parent class, we define guidelines for the child classes, which describe the methods to be compulsorily implemented by the child class.

```
abstract void addData(int x,int y);
```

#### Example abstract method

1. abstract void display(); //valid-only declaration, nobody defined.
2. abstract void display() { } //invalid

## **Abstract class in Java**

- A class that is declared with an abstract keyword is known as an abstract class in java.
- It can have an abstract method (method without body) and non-abstract methods (method with the body).
- It needs to be extended and its method should be implemented.
- It cannot be instantiated.

#### Example abstract class

1. abstract class A{ }

### Case 1:

---

```
1
2  abstract class Upes {
3      public static void main(String args[])
4      {
5          // Creating object of class inside main() method
6          Upes u =new Upes();
7      }
8 }
```

### Output:

```
C:\Windows\system32\cmd.exe
Microsoft Windows [Version 10.0.22000.778]
(c) Microsoft Corporation. All rights reserved.

D:\1 UPES Data aug 2022\Java CSF\abstract>javac Upes.java
Upes.java:6: error: Upes is abstract; cannot be instantiated
        Upes u =new Upes();
                           ^
1 error
```

### Case 2:

---

```
1
2  class Upes
3  {
4      // Method of this class
5      public void methodOne();
6 }
```

### Output:

```
D:\1 UPES Data aug 2022\Java CSF\abstract>javac Upes.java
Upes.java:4: error: missing method body, or declare abstract
        public void methodOne();
                           ^
1 error
```

**Case 3:**

```
class Upes
{ // Method of this class
    public abstract void method1() {}
```

**Output:**

```
D:\1 UPES Data aug 2022\Java CSF\abstract>javac Upes.java
Upes.java:2: error: Upes is not abstract and does not override abstract method method1() in Upes
    class Upes
    ^
Upes.java:4: error: abstract methods cannot have a body
    public abstract void method1() {}
                           ^
2 errors
```

**Case 4:**

```
class Upes
{ // Method of this class
    public abstract void method1();
```

**Output:**

```
D:\1 UPES Data aug 2022\Java CSF\abstract>javac Upes.java
Upes.java:2: error: Upes is not abstract and does not override abstract method method1() in Upes
    class Upes
    ^
1 error
```

**Example:**

```
abstract class Parent
{
    public abstract void m1();
    public abstract void m2();
}
class TestChild extends Parent
{
    public void m1(){ //body of m1}
}
```

**Output:**

```
F:\Java Code 2020>javac TestChild.java
TestChild.java:6: error: TestChild is not abstract and does
not override abstract method m2() in Parent
class TestChild extends Parent
^
1 error
```

We can handle this error either by declaring the child class as abstract (case 1) or by providing an implementation for m2() (case 2).

**case 1:**

```
abstract class Parent
{
    public abstract void m1();
    public abstract void m2();
}
abstract class TestChild extends Parent
{
    public void m1(){}
}
```

**case 2**

```
abstract class Parent
{
    public abstract void m1();
    public abstract void m2();
}
class TestChild extends Parent
{
```

```
public void m1(){}
public void m2(){}
}
```

C:\Windows\System32\cmd.exe

```
F:\Java Code 2020>javac TestChild.java
```

```
F:\Java Code 2020>
```

### Example of an abstract class that has an abstract method

```
// Abstract class
abstract class Animal {
    // Abstract method
    abstract void makeSound();

    // Non-abstract method
    void sleep() {
        System.out.println("Sleeping...");
    }
}

// Subclass extending the abstract class
class Dog extends Animal {
    // Implementing the abstract method
    void makeSound() {
        System.out.println("Bark! Bark!");
    }

    public static void main(String args[]) {
        // Animal obj = new Animal(); // Compile-time error (Animal is an abstract class)
        Dog obj = new Dog();
        obj.makeSound();
        obj.sleep();
    }
}
```

### Output:

Bark! Bark!  
Sleeping...

## Key Observations About Abstract Classes

- An instance of an abstract class cannot be created.
- Constructors are allowed in abstract classes.
- An abstract class can exist without abstract methods.
- An abstract method cannot be final because a final method cannot be overridden.
- Static methods are allowed in an abstract class.
- The abstract keyword can be used for both outer and inner classes.
- If a class contains at least one abstract method, the class must be declared as abstract.
- If a child class does not implement all abstract methods, it must also be declared as abstract.

### Example 1: Abstract Class with an Abstract Method

```
// Abstract class
abstract class Appliance {
    // Abstract method
    abstract void operate();
}

// Subclass providing implementation
class WashingMachine extends Appliance {
    void operate() {
        System.out.println("Washing clothes...");
    }
}

public static void main(String[] args) {
    Appliance obj = new WashingMachine();
    obj.operate();
}
```

Output:  
Washing clothes...

### Example 2: Abstract Class with Constructor and Methods

Abstract classes can have constructors and non-abstract methods. The following example demonstrates this:

```
// Abstract class with a constructor and methods
abstract class Vehicle {
    Vehicle() {
        System.out.println("Vehicle is being created");
    }

    // Abstract method
    abstract void start();

    // Non-abstract method
    void stop() {
```

```
        System.out.println("Vehicle stopped");
    }
}
```

```
// Subclass implementing abstract method
class Car extends Vehicle {
    void start() {
        System.out.println("Car started");
    }
}
```

```
public class Main {
    public static void main(String[] args) {
        Vehicle myCar = new Car();
        myCar.start();
        myCar.stop();
    }
}
```

**Output:**

```
Vehicle is being created
Car started
Vehicle stopped
```

### Example 3: Abstract Class Without Abstract Methods

An abstract class can exist without any abstract methods. It is useful when we want to restrict instantiation but allow inheritance.

```
abstract class Account {
    void showAccountType() {
        System.out.println("This is a bank account");
    }
}
```

```
// Subclass inheriting Account class
class SavingsAccount extends Account {
    void deposit() {
        System.out.println("Money deposited in savings account");
    }
}
```

```
public class Main {
    public static void main(String[] args) {
        SavingsAccount acc = new SavingsAccount();
        acc.showAccountType();
        acc.deposit();
    }
}
```

**Output:**

```
This is a bank account
Money deposited in savings account
```

#### **Example 4: Abstract Classes Can Have Final Methods**

An abstract class can have final methods, but abstract methods cannot be final.

```
abstract class Computer {  
    // Final method cannot be overridden  
    final void displayInfo() {  
        System.out.println("This is a computer");  
    }  
}  
  
// Subclass  
class Laptop extends Computer {  
    // Cannot override displayInfo() because it is final  
}  
  
public class Main {  
    public static void main(String[] args) {  
        Laptop myLaptop = new Laptop();  
        myLaptop.displayInfo();  
    }  
}  
Output:  
This is a computer
```

#### **Example 5: Abstract Class with Static Methods**

Static methods can be defined in an abstract class, and they can be called independently without an object.

```
abstract class MathOperations {  
    static void add(int a, int b) {  
        System.out.println("Sum: " + (a + b));  
    }  
}  
public class Main {  
    public static void main(String[] args) {  
        // Calling static method without creating an object  
        MathOperations.add(10, 20);  
    }  
}  
Output:  
Sum: 30
```

#### **Example 6: Abstract Class for Outer and Inner Classes**

Abstract classes can be applied to both outer and inner classes.

```
abstract class OuterClass {  
    // Abstract inner class  
    abstract class InnerClass {  
        abstract void message();  
    }  
}
```

```

}

// Subclass implementing inner abstract class
class SubClass extends OuterClass {
    class ConcreteInner extends InnerClass {
        void message() {
            System.out.println("Inner abstract class implementation");
        }
    }
}

public class Main {
    public static void main(String[] args) {
        SubClass outer = new SubClass();
        SubClass.ConcreteInner inner = outer.new ConcreteInner();
        inner.message();
    }
}

```

**Output:**  
Inner abstract class implementation

### Example 7: Child Class Declared as Abstract if Methods Are Not Implemented

If a child class does not implement all abstract methods of the parent class, it must be declared as abstract.

```

abstract class Animal {
    abstract void sound();
    abstract void eat();
}

// First child class - does not implement all methods, so it must be abstract
abstract class Mammal extends Animal {
    void sound() {
        System.out.println("Mammals make different sounds");
    }
}

// Second child class - implements all remaining methods
class Dog extends Mammal {
    void eat() {
        System.out.println("Dog eats bones");
    }
}

public class Main {
    public static void main(String[] args) {
        Dog myDog = new Dog();
        myDog.sound();
        myDog.eat();
    }
}

```

**Output:**

Mammals make different sounds

Dog eats bones

## **Abstraction in Java**

**Abstraction** is a process of hiding the implementation details and showing only functionality to the user. Another way is that it shows only important things to the user and hides the internal details. For example, when sending an email, you type the text and send the message. You don't know how to process the message delivery internally.

### **Ways to achieve Abstraction**

There are two ways to achieve abstraction in java

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

### **Examples:**

```
abstract class A—0%
{
    m1(){stmts.....}—0%
    m2(){stmts.....}—0%
    m3() {stmts.....}--0%
    m4() {stmts.....}--0%
    m5(){stmts.....}--0%
}
```

```
abstract class A—60%
{
    m1(){stmts.....}—0%
    m2(){stmts.....}—0%
    abstract m3();--20%
    abstract m4();--20%
    abstract m5();--20%
}
```

```
abstract class A—100%
{
    abstract m1();—20%
    abstract m2();—20%
    abstract m3();--20%
    abstract m4();--20%
    abstract m5();--20%
}
```

**Note:** The use of abstract methods, abstract class and interfaces are recommended, and it is always good programming practice.

## INTERFACE IN JAVA

- An **interface** is a blueprint of a class.
- The interface is a **mechanism to achieve full abstraction** in Java.
- There can be only abstract methods in the interface.
- It is used to achieve full abstraction and multiple inheritances in Java.
- It cannot be instantiated just like an abstract class.

## WHY USE INTERFACE?

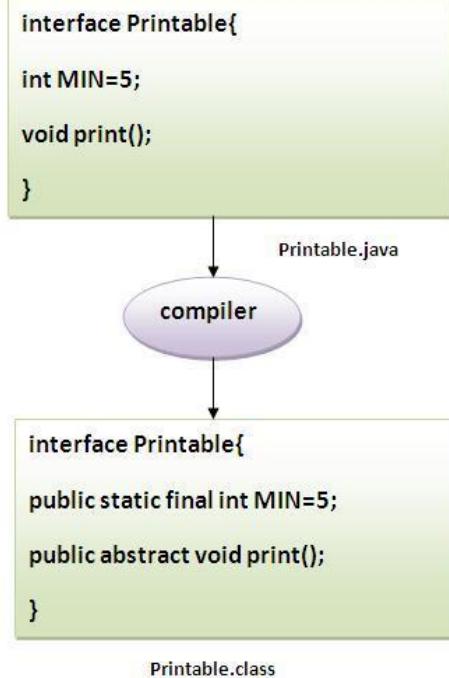
There are the following reasons to use an interface. They are given below.

- It is used to achieve full (100%) abstraction.
- By interface, we can support the functionality of multiple inheritances.

### Interface fields(data members) and Methods( member functions):

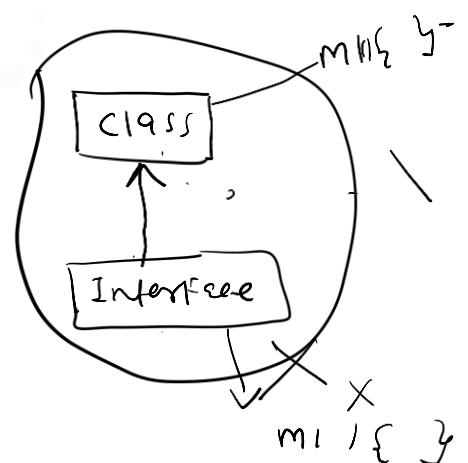
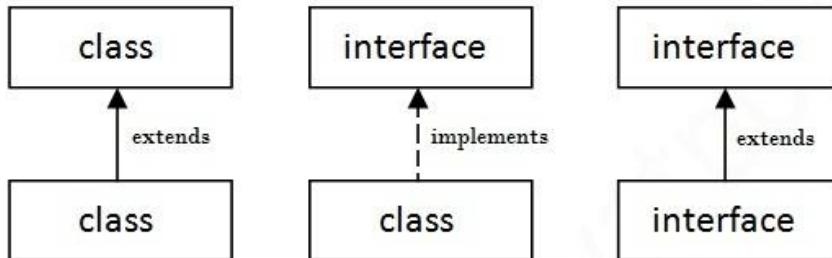
In other words, Interface fields are **public, static, and final** by default, and methods are **public and abstract**.

The java compiler adds public, static, and final keywords before data members and adds public and abstract keywords before the interface method.



## Understanding relationship between classes and interfaces

- A class extends another class.
- An interface extends another interface
- but a **class implements an interface**.



## **extends Vs implements**

A class can extend only one class at a time.

class A extends B ✓

class A extends B, C ✗

A class can implement any number of interfaces at a time.

class A implements B, C ✓

A class can extend a class and implement any number of interfaces simultaneously.

class A extends B implements C,D ✓

class A implements B,C extends D ✗

An interface can extend any number of interfaces at a time.

interface A extends B,C ✓

Which of the following is true??

1. A class can extend any number of classes simultaneously.
2. A class implement only one interface at a time.
3. A class extend a class OR implement an interface but not both simultaneously.
4. An interface can implement any number of classes.
5. An interface can extend only one interface at a time.
6. None of the above.

## **Consider the expressions:**

**X extends Y**

Which of the following properties is true?

1. Both should be classes
2. Both should be interfaces
3. No restriction

4. Both can be either classes or interfaces.

**X extends Y,Z**

1. X,Y,Z should be classes.
2. X should be class and Y,Z should be interfaces.
3. X,Y,Z should be interfaces.

**X extends Y implements Z**

X,Y should be classes, Z should be interface ✓

**X implements Y extends Z** -Compile Time Error

### **Example: A class implements one interface**

```
1. interface printable{  
2.   void print(); //by default public and abstract  
3. }  
4.  
5. class A implements printable{  
6.   public void print()//overridden method  
7. {  
8.   System.out.println("Hello");  
9. }  
10.  
11. public static void main(String args[]){  
12.   //printable obj = new printable ()//  
13.   A obj=new A();  
14.   //printable obj = new A ()//  
15.   obj.print();  
16. }  
17. }
```

✗  
✓  
✓

### **OUTPUT:HELLO**

### **EXAMPLE: TWO CLASSES IMPLEMENT ONE INTERFACE**

```
1. interface Drawable {  
2.   void draw();  
3. }  
4. class Rectangle implements Drawable{  
5.   public void draw()//overridden  
6. {  
7.   System.out.println("drawing rectangle");  
8. }  
9. }  
10. class Circle implements Drawable  
11. {  
12.   public void draw()//overridden  
13. {  
14.   System.out.println("drawing circle");  
15. }  
16. }  
17. class TestInterface  
18. {  
19.   public static void main(String args[])  
20. {  
21.   Drawable d=new Circle();  
22.   Drawable e =new Rectangle();  
23.   d.draw();
```

```
24. e.draw();
25. }
26. }
```

**Output:**

drawing circle  
drawing rectangle

**EXAMPLE: A CLASS EXTENDS ONE CLASS AND IMPLEMENTS ONE INTERFACE (MULTIPLE INHERITANCE)**

```
class Teacher
{
int marks;
void setMark(int m)
{
marks=m;
}
void getMark()
{
System.out.println("marks are:"+marks);
}
}
```

```
interface Hod
{
int total=200;
void putSign();
}
```

```
class Results extends Teacher implements Hod
{
public void putSign()
{
System.out.println("marks verified and put sign and forward");
}
void display()
{
System.out.println("Out of =" +total);
}
public static void main(String args[])
{
Results r=new Results();
r.setMark(175);
r.getMark();
r.display();
```

```
r.putSign();
}
}
```

C:\WINDOWS\system32\cmd.exe

```
D:\Java Lab\13>javac Results.java

D:\Java Lab\13>java Results
marks are:175
Out of =200
marks verified and put sign and forward

D:\Java Lab\13>
```

//A Class implements multiple interfaces (Multiple inheritance)

```
1. interface Printable{
2.     void print();
3. }
4.
5. interface Showable{
6.     void show();
7. }
8.
9. class A implements Printable,Showable{
10.
11. public void print(){System.out.println("Hello");}
12. public void show(){System.out.println("Welcome");}
13.
14. public static void main(String args[]){
15.     A obj = new A();
16.     obj.print();
17.     obj.show();
18. }
19. }
```

**Output:**

Hello  
Welcome

//No ambiguity in multiple inheritance.

```
1. interface Printable{
2.     void print();
3. }
4.
5. interface Showable{
6.     void print();
7. }
8.
9. class A implements Printable,Showable{
10.
11. public void print(){System.out.println("Hello");}
12. public static void main(String args[]){
13.     A obj = new A();
14.     obj.print();
15. }
```

16. }

Output:Hello

## MULTILEVEL INHERITANCE

```
1. interface Printable{      0
2. void print();
3. }
4. interface Showable extends Printable{    1
5. void show();
6. }
7. class A implements Showable{      2
8. public void print(){System.out.println("Hello");}
9. public void show(){System.out.println("Welcome");}
10.
11. public static void main(String args[]){
12. A obj = new A();
13. obj.print();
14. obj.show();
15. }
16. }
```

Output: Hello

Welcome

## Default Methods in Java 8

Before Java 8, interfaces could have only abstract methods. The implementation of these methods must be provided in a separate class. So, if a new method is to be added to an interface, then its implementation code must be provided in the class implementing the same interface. To overcome this issue, Java 8 introduced the concept of default methods, which allow the interfaces to have methods with implementation without affecting the classes that implement the interface.

The default methods were introduced to provide backward compatibility so that existing interfaces can use lambda expressions without implementing the methods in the implementation class. Default methods are also known as **defender methods or virtual extension methods**.

```
// A simple program to Test Interface default methods in Java
```

```
1  interface TestInterface
2  {
3      public void square(int a); // abstract method
4      default void show() // default method
5      {
6          System.out.println("Default Method Executed");
7      }
8  }
9  class TestClass implements TestInterface
10 {
11     // implementation of square abstract method
12     public void square(int a)
13     {
14         System.out.println(a*a);
15     }
16     public static void main(String args[])
17     {
18         TestClass d = new TestClass();
19         d.square(4);
20         d.show(); // default method executed
21     }
22 }
```

Output:

```
D:\Java Code 2k23>java TestClass
16
Default Method Executed
```

### **Static Methods:**

The interfaces can have static methods as well, which is like the static method of classes.

```
// A simple Java program to TestClassnstrate static methods in Java
```

```
interface TestInterface
{
    // abstract method
    public void square (int a);

    // static method
    static void show()
    {
        System.out.println("Static Method Executed");
    }
}

class TestClass implements TestInterface
{
    // Implementation of square abstract method
    public void square (int a)
    {
        System.out.println(a*a);
    }

    public static void main(String args[])
    {
        TestClass d = new TestClass();
        d.square(4);

        // Static method executed
        TestInterface.show();
    }
}
```

### **Output:**

```
16
```

```
Static Method Executed
```

## **Default Methods and Multiple Inheritance**

In case both the implemented interfaces contain default methods with the same method signature, the implementing class should explicitly specify which default method is to be used, or it should override the default method.

```
// A simple Java program to demonstrate multiple inheritance through default methods.
```

```
interface TestInterface1
{
    // default method
    default void show()
    {
        System.out.println("Default TestInterface1");
    }
}

interface TestInterface2
{
    // Default method
    default void show()
    {
        System.out.println("Default TestInterface2");
    }
}

// Implementation class code
class TestClass implements TestInterface1, TestInterface2
{
    // Overriding default show method
    public void show()
    {
        // use super keyword to call the show
        // method of TestInterface1 interface
        TestInterface1.super.show();

        // use super keyword to call the show
        // method of TestInterface2 interface
        TestInterface2.super.show();
    }

    public static void main(String args[])
    {
        TestClass d = new TestClass();
        d.show();
    }
}
```

### **Output:**

Default TestInterface1

Default TestInterface2

Important Points:

## Java Packages

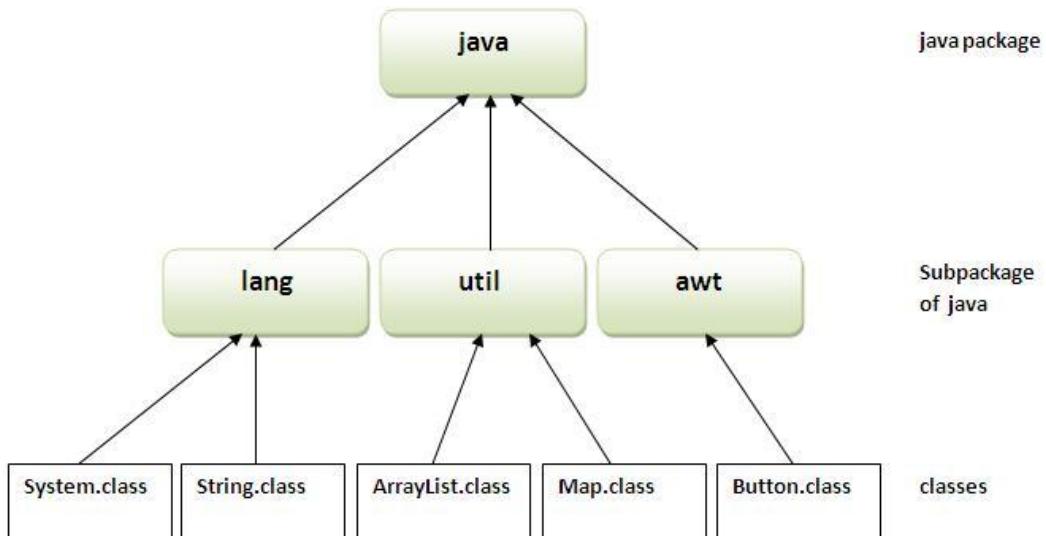
It is an encapsulation mechanism to group related classes and interfaces into a single module.  
A **java package** is a group of similar types of classes, interfaces, and sub-packages.

Packages in java can be categorized in two forms:

- built-in packages like lang, util, awt, javax, swing, net, io, sql etc.-predefined packages
- user-defined package

### Advantages of Java Package

- Java package is used to categorize the classes and interfaces so that they can be easily maintained.
- Java package provides access protection/security to the classes and interfaces. So that an outside person cannot access it directly.
- It improves the modularity of the application.
- Java package removes naming collision.



C:\Program Files\Java\jdk-11.0.15.1\lib\src.zip\java.base\java

| Name     | Type        | Compressed size | Password ... | Size | Ratio |
|----------|-------------|-----------------|--------------|------|-------|
| io       | File folder |                 |              |      |       |
| lang     | File folder |                 |              |      |       |
| math     | File folder |                 |              |      |       |
| net      | File folder |                 |              |      |       |
| nio      | File folder |                 |              |      |       |
| security | File folder |                 |              |      |       |
| text     | File folder |                 |              |      |       |
| time     | File folder |                 |              |      |       |
| util     | File folder |                 |              |      |       |

## Location of Scanner class:

| Name                      | Type      | Compressed size | Password ... | Size   | Ratio | Date modified     |
|---------------------------|-----------|-----------------|--------------|--------|-------|-------------------|
| Scanner                   | JAVA File | 20 KB           | No           | 116 KB | 84%   | 4/25/2022 8:43 PM |
| ServiceConfigurationError | JAVA File | 1 KB            | No           | 2 KB   | 56%   | 4/25/2022 8:43 PM |
| ServiceLoader             | JAVA File | 16 KB           | No           | 73 KB  | 79%   | 4/25/2022 8:43 PM |
| Set                       | JAVA File | 6 KB            | No           | 30 KB  | 81%   | 4/25/2022 8:43 PM |
| SimpleTimeZone            | JAVA File | 14 KB           | No           | 70 KB  | 81%   | 4/25/2022 8:43 PM |
| SortedMap                 | JAVA File | 4 KB            | No           | 13 KB  | 76%   | 4/25/2022 8:43 PM |
| SortedSet                 | JAVA File | 4 KB            | No           | 12 KB  | 74%   | 4/25/2022 8:43 PM |
| Spliterator               | JAVA File | 9 KB            | No           | 37 KB  | 77%   | 4/25/2022 8:43 PM |

## In old versions:

If you want to see the .class files, they're in **lib\rt.jar** in the JRE directory (.jar is the same as .zip, so you can open it with anything that can open zip files).

| This PC > Local Disk (C:) > Program Files > Java > jre1.8.0_131 > lib |                           |                  |                       |           |
|---|---------------------------|------------------|-----------------------|-----------|
|   | Name                      | Date modified    | Type                  | Size      |
| ss  | calendar                  | 8/1/2018 9:50 AM | Properties Source ... | 2 KB      |
| ce Aug ⌘  | charsets                  | 8/1/2018 9:50 AM | Executable Jar File   | 2,966 KB  |
| 2019  | classlist                 | 8/1/2018 9:50 AM | File                  | 83 KB     |
| e 2020  | content-types             | 8/1/2018 9:50 AM | Properties Source ... | 6 KB      |
| es _2020  | currency.data             | 8/1/2018 9:50 AM | DATA File             | 5 KB      |
|   | deploy                    | 8/1/2018 9:50 AM | Executable Jar File   | 4,921 KB  |
|   | flavormap                 | 8/1/2018 9:50 AM | Properties Source ... | 4 KB      |
|   | fontconfig.bfc            | 8/1/2018 9:50 AM | BFC File              | 4 KB      |
|   | fontconfig.properties.src | 8/1/2018 9:50 AM | SRC File              | 11 KB     |
|   | hijrah-config-umalqura    | 8/1/2018 9:50 AM | Properties Source ... | 14 KB     |
|   | javafx                    | 8/1/2018 9:50 AM | Properties Source ... | 1 KB      |
|   | javaws                    | 8/1/2018 9:50 AM | Executable Jar File   | 919 KB    |
|   | jce                       | 8/1/2018 9:50 AM | Executable Jar File   | 114 KB    |
|   | jfr                       | 8/1/2018 9:50 AM | Executable Jar File   | 548 KB    |
|   | jfxswt                    | 8/1/2018 9:50 AM | Executable Jar File   | 34 KB     |
|   | jsse                      | 8/1/2018 9:50 AM | Executable Jar File   | 570 KB    |
|   | jvm.hprof                 | 8/1/2018 9:50 AM | Text Document         | 5 KB      |
|   | logging                   | 8/1/2018 9:50 AM | Properties Source ... | 3 KB      |
|   | management-agent          | 8/1/2018 9:50 AM | Executable Jar File   | 1 KB      |
|   | meta-index                | 8/1/2018 9:50 AM | File                  | 3 KB      |
|   | net                       | 8/1/2018 9:50 AM | Properties Source ... | 5 KB      |
|   | plugin                    | 8/1/2018 9:50 AM | Executable Jar File   | 1,878 KB  |
|   | psfont.properties.ja      | 8/1/2018 9:50 AM | JA File               | 3 KB      |
|   | psfontj2d                 | 8/1/2018 9:50 AM | Properties Source ... | 11 KB     |
|   | resources                 | 8/1/2018 9:50 AM | Executable Jar File   | 3,411 KB  |
|   | rt                        | 8/1/2018 9:50 AM | Executable Jar File   | 53,231 KB |
|   | sound                     | 8/1/2018 9:50 AM | Properties Source ... | 2 KB      |
|   | tzdb.dat                  | 8/1/2018 9:50 AM | DAT File              | 104 KB    |

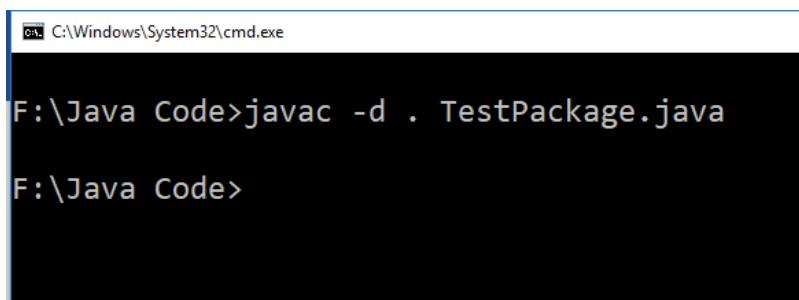
## My first Java package

```
package mypackage;
public class TestPackage{
    public static void main(String args[]){
        System.out.println("This is my package");
    }
}
```

### Save it:

TestPackage.java

### Compilation command:

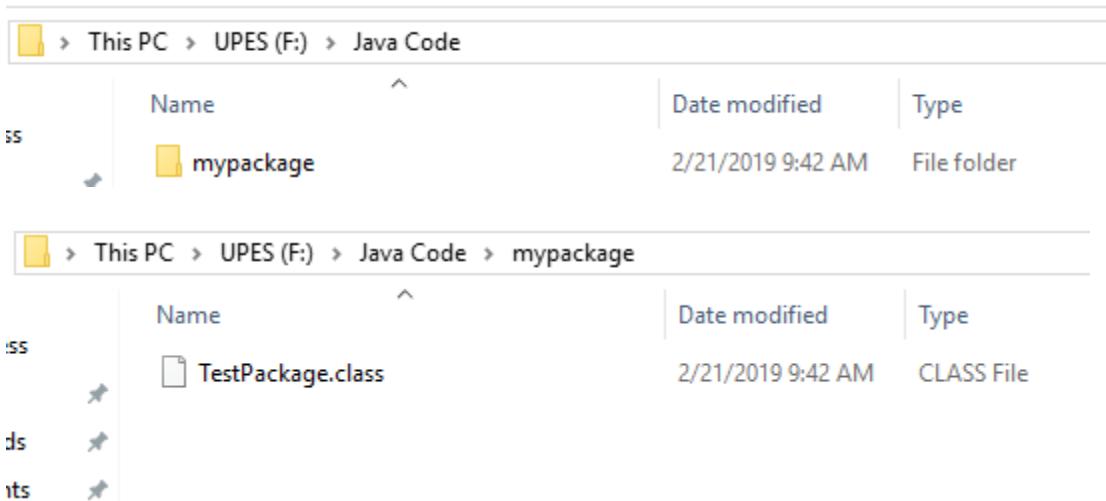


```
C:\Windows\System32\cmd.exe
F:\Java Code>javac -d . TestPackage.java
F:\Java Code>
```

The `-d` is a switch that tells the compiler where to put the class file i.e. it represents the destination. And `.` (dot) represents the current working directory/folder.

After the compilation of this program, the compiler will generate two things:

1. `mypackage` folder at **F:\Java Code** location
2. `TestPackage.class` file at `mypackage` folder.



### **Some Wrong Commands:**

```
F:\Java Code>javac-d.TestPackage.java
'javac-d.TestPackage.java' is not recognized as an in-
operable program or batch file.

F:\Java Code>javac -d.TestPackage.java
javac: file not found: -d.TestPackage.java
Usage: javac <options> <source files>
use -help for a list of possible options

F:\Java Code>javac -d .TestPackage.java
javac: directory not found: .TestPackage.java
Usage: javac <options> <source files>
use -help for a list of possible options
```

### **Execution command:**

```
F:\Java Code>javac -d . TestPackage.java

F:\Java Code>java mypackage.TestPackage
This is my package
```

### **Creation of package at user-defined location:**

```
package mypackage1;
public class TestPackage1{
    public static void main(String args[]){
        System.out.println("This is my package1");
    }
}
```

**Valid:**

```
F:\Java Code>javac -d "F:\Java Code\checkpack" TestPackage1.java  
F:\Java Code>
```

**Invalid command :**

```
F:\Java Code\checkpack>javac -d F:\Java Code\checkpack TestPackage1.java  
javac: invalid flag: Code\checkpack  
Usage: javac <options> <source files>  
use -help for a list of possible options
```

After the compilation of this program, the compiler will generate two things:

1. Mypackage1 folder at **F:\Java Code\checkpack** location
2. TestPackage1.class file at mypackage1 folder.

| This PC > UPES (F:) > Java Code > checkpack |            |                    |             |      |
|---|------------|--------------------|-------------|------|
|   | Name       | Date modified      | Type        | Size |
|   | mypackage1 | 2/21/2019 10:28 AM | File folder |      |

| This PC > UPES (F:) > Java Code > checkpack > mypackage1 |                    |                    |            |  |
|--|--------------------|--------------------|------------|--|
|  | Name               | Date modified      | Type       |  |
|  | TestPackage1.class | 2/21/2019 10:28 AM | CLASS File |  |

```
F:\Java Code>cd checkpack  
F:\Java Code\checkpack>java mypackage1.TestPackage1  
This is my package1
```

There are three techniques to access the package from outside the package. (From one package to another package)

### 1. import package.\*;

The statement import package.\*; is used in Java to import all the classes and interfaces from a specific package into your current Java file.

**Creating a package named “upes”**

```
1 package upes;
2 public class Ifm{
3     public void ifmData()
4     {
5         System.out.println("Data of IFM Students");
6     }
7 }
```

```
F:\Java Code>javac -d . Ifm.java
```

Creating another package named “adminupes” and importing “upes” package.

```
package adminUpes;
import upes.*;

class Admin{
    public static void main(String args[]){
        Ifm i = new Ifm();
        i.ifmData();
    }
}
```

```
F:\Java Code>javac -d . Admin.java
```

```
F:\Java Code>java adminUpes.Admin
Data of IFM Students
```

## 2. import package.classname;

The statement import package.classname; is used in Java to specifically import a single class (classname) from a specific package (package).

### Creation of package named “upes”

```
1 package upes;
2 public class Ifm{
3     public void ifmData()
4     {
5         System.out.println("Data of IFM Students");
6     }
7 }
```

### Creation of package named “adminUpes” and importing “Ifm” class from “upes” package

```
1 package adminUpes;
2 import upes.Ifm;
3
4 class Admin{
5     public static void main(String args[]){
6         Ifm i = new Ifm();
7         i.ifmData();
8     }
9 }
```

### Output:

```
F:\Java Code>javac -d . Ifm.java

F:\Java Code>javac -d . Admin.java

F:\Java Code>java adminUpes.Admin
Data of IFM Students
```

### 3. fully qualified name.

Fully qualified name (FQN) refers to the complete name of a class or interface, including the package name to which it belongs. A fully qualified name uniquely identifies a specific class or interface within the entire Java ecosystem.

```
package adminUpes;
//import upes.Ifm;

class Admin{
    public static void main(String args[]){
        upes.Ifm i = new upes.Ifm();
        i.ifmData();
    }
}
```

```
F:\Java Code>javac -d . Ifm.java

F:\Java Code>javac -d . Admin.java

F:\Java Code>java adminUpes.Admin
Data of IFM Students
```

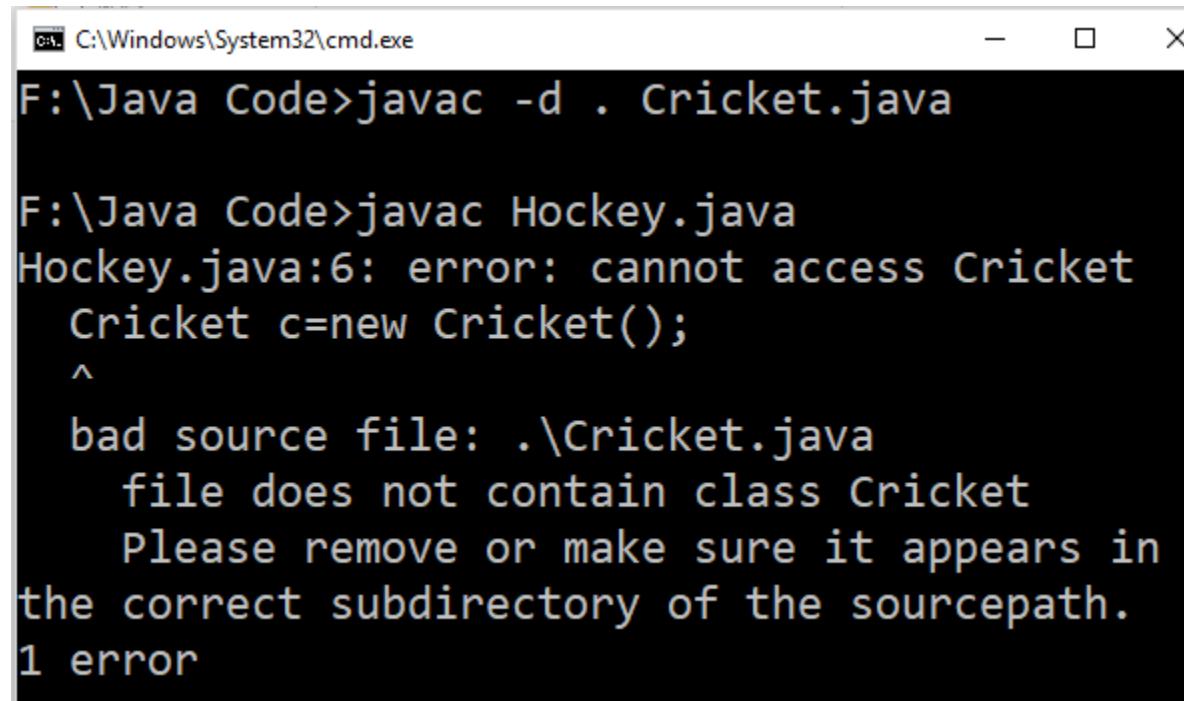
According to compiler version behavior, there are only two ways to access the package from outside the package. (class → package).

#### 1. import package.\*;

```
package Sport;
public class Cricket{
String name="Subham";
    public void run()
    {
        System.out.println(name+ " is a Cricket Player ");
    }
}
```

**import Sport.\*;** //will give a compile time error, this technique will work with predefined packages only. It may be a compiler issue/bug.

```
import Sport.*;
class Hockey{
    public static void main(String args[]){
        Cricket c=new Cricket();
        c.run();
    }
}
```



C:\Windows\System32\cmd.exe

```
F:\Java Code>javac -d . Cricket.java

F:\Java Code>javac Hockey.java
Hockey.java:6: error: cannot access Cricket
    Cricket c=new Cricket();
               ^
bad source file: .\Cricket.java
  file does not contain class Cricket
    Please remove or make sure it appears in
the correct subdirectory of the sourcepath.
1 error
```

## 2. import package.classname;

```
import Sport.Cricket;

class Hockey{
    public static void main(String args[]){
        Cricket c=new Cricket();
        c.run();
    }
}
```

```
F:\Java Code>javac -d . Cricket.java
```

```
F:\Java Code>javac Hockey.java
```

```
F:\Java Code>java Hockey  
Subham is a Cricket Player
```

### 3. fully qualified name.

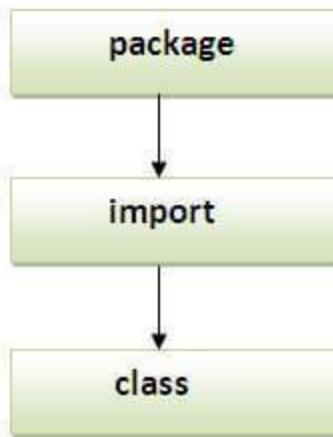
```
//import Sport.Cricket—no need to write here  
class Hockey{  
    public static void main(String args[]){  
        Sport.Cricket c=new Sport.Cricket();  
        c.run();  
    }  
}
```

```
F:\Java Code>javac -d . Cricket.java
```

```
F:\Java Code>javac Hockey.java
```

```
F:\Java Code>java Hockey  
Subham is a Cricket Player
```

If you import a package, all the classes and interfaces of that package will be imported excluding the classes and interfaces of the sub packages. Hence, you need to import the sub package as well.



### **Subpackage in java**

The package inside the package is called the **subpackage**. It should be created to categorize the package further.

Let's take an example, Sun Microsystem has defined a package named java that contains many classes like System, String, Reader, Writer, Socket, etc. These classes represent a particular group e.g. Reader and Writer classes are for Input/Output operation, Socket and ServerSocket classes are for networking, etc., and so on. So, Sun has subcategorized the java package into sub-packages such as lang, net, io, etc., and put the Input/Output related classes in the io package, Server and ServerSocket classes in net packages, and so on.

```

package upes.socs.ifm;
class Data{
    public static void main(String args[]){
        System.out.println("data of subpackage");
    }
}

```

```

F:\Java Code>javac -d . Data.java

F:\Java Code>java upes.socs.ifm.Data
data of subpackage

```

| File Explorer |            |                    |            |
|---------------|------------|--------------------|------------|
|               | Name       | Date modified      | Type       |
| ss            | Data.class | 2/22/2019 12:13 PM | CLASS File |

**Note:** In any java program there should be only at most 1 package statement. If we are taking more than one package statement we will get compile time error.

```
package pack1;  
package pack2;//error
```

```
class MyPackage1  
{  
//  
--statements—  
}
```

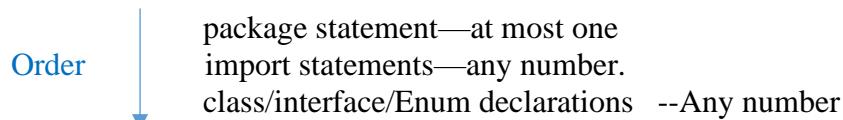
compile time error: class, interface, and enum expected.

**Note:** If any java program the first non-comment statement should be a package statement(if it is available).

```
import java.util.*;  
package pack1;//error  
class MyPackage2  
{  
}
```

Compile time error: class, interface, and enum expected.

Structure of Java source file:



Compiling multiple classes in single package:

---

```
1 // File: calculation/Addition.java
2 package calculation;
3
4 public class Addition {
5     public static int add(int a, int b) {
6         return a + b;
7     }
8 }
```

---

```
1 // File: calculation/Subtraction.java
2 package calculation;
3
4 public class Subtraction {
5     public static int subtract(int a, int b) {
6         return a - b;
7     }
8 }
```

---

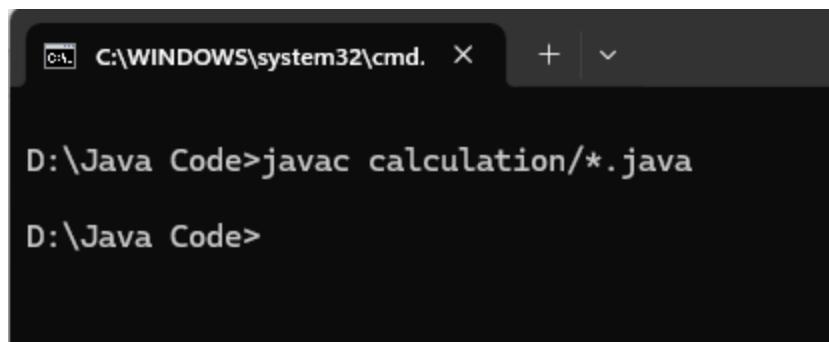
```
1 // File: calculation/Multiplication.java
2 package calculation;
3
4 public class Multiplication {
5     public static int multiply(int a, int b) {
6         return a * b;
7     }
8 }
```

---

```
1 // File: calculation/Division.java
2 package calculation;
3
4 public class Division {
5     public static double divide(int a, int b) {
6         if (b == 0) {
7             throw new IllegalArgumentException("Cannot divide by zero");
8         }
9         return (double) a / b;
10    }
11 }
```

```
1 // File: Test.java
2 import calculation.*;
3
4 public class Test {
5     public static void main(String[] args) {
6         int num1 = 10;
7         int num2 = 5;
8
9         System.out.println("Addition: " + num1 + " + " + num2 + " = " + Addition.add(num1, num2));
10        System.out.println("Subtraction: " + num1 + " - " + num2 + " = " + Subtraction.subtract(num1, num2));
11        System.out.println("Multiplication: " + num1 + " * " + num2 + " = " + Multiplication.multiply(num1, num2));
12        try {
13            System.out.println("Division: " + num1 + " / " + num2 + " = " + Division.divide(num1, num2));
14        } catch (IllegalArgumentException e) {
15            System.out.println(e.getMessage());
16        }
17    }
18}
```

| This PC > New Volume (D:) > Java Code > calculation |                      |                   |            |      |
|---|----------------------|-------------------|------------|------|
|   | Name                 | Date modified     | Type       | Size |
|   | Addition.class       | 3/26/2024 3:42 PM | CLASS File | 1 KB |
|   | Addition             | 3/26/2024 3:38 PM | JAVA File  | 1 KB |
|   | Division.class       | 3/26/2024 3:42 PM | CLASS File | 1 KB |
|   | Division             | 3/26/2024 3:39 PM | JAVA File  | 1 KB |
|   | Multiplication.class | 3/26/2024 3:42 PM | CLASS File | 1 KB |
|   | Multiplication       | 3/26/2024 3:39 PM | JAVA File  | 1 KB |
|   | Subtraction.class    | 3/26/2024 3:42 PM | CLASS File | 1 KB |
|   | Subtraction          | 3/26/2024 3:39 PM | JAVA File  | 1 KB |
|   | Test.class           | 3/26/2024 3:42 PM | CLASS File | 2 KB |
|   | Test                 | 3/26/2024 3:39 PM | JAVA File  | 1 KB |



A screenshot of a Windows Command Prompt window titled "C:\WINDOWS\system32\cmd.". The window shows the command "javac calculation/\*.java" being run from the directory "D:\Java Code". The output shows the command being executed and then the prompt returning to "D:\Java Code>".

```
C:\WINDOWS\system32\cmd. + | v  
D:\Java Code>javac calculation/*.java  
D:\Java Code>
```

Execute and see the output.

## Java String

In java, string is basically an object that represents sequence of char values. An array of characters works same as java string. For example:

1. `char[] ch={'s','a','u','r','a','b','h'};`
2. `String s=new String(ch);`

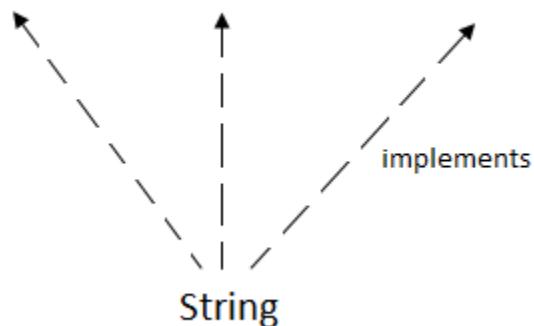
is same as:

1. `String s="saurabh";`

**Java String** class provides a lot of methods to perform operations on string such as `compare()`, `concat()`, `equals()`, `split()`, `length()`, `replace()`, `compareTo()`, `intern()`, `substring()` etc.

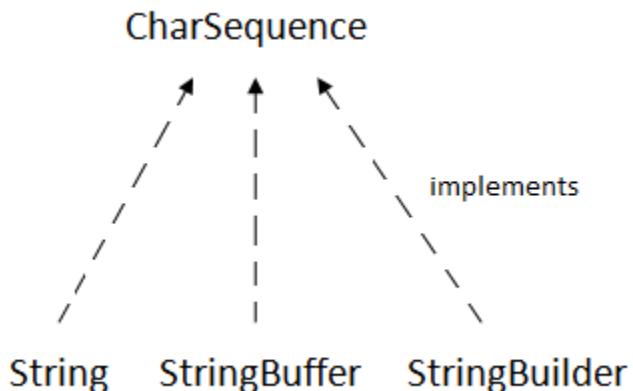
The `java.lang.String` class implements *Serializable*, *Comparable* and *CharSequence* interfaces.

**Serializable   Comparable   CharSequence**



## CharSequence Interface

The `CharSequence` interface is used to represent sequence of characters. It is implemented by `String`, `StringBuffer` and `StringBuilder` classes. It means, we can create string in java by using these 3 classes.



The java String is immutable i.e. it cannot be changed. Whenever we change any string, a new instance is created. For mutable string, you can use StringBuffer and StringBuilder classes.

We will discuss about immutable string later. Let's first understand what is string in java and how to create the string object.

### What is String in java

Generally, string is a sequence of characters. But in java, string is an object that represents a sequence of characters. The `java.lang.String` class is used to create string object.

### How to create String object?

There are two ways to create String object:

1. By string literal
2. By new keyword

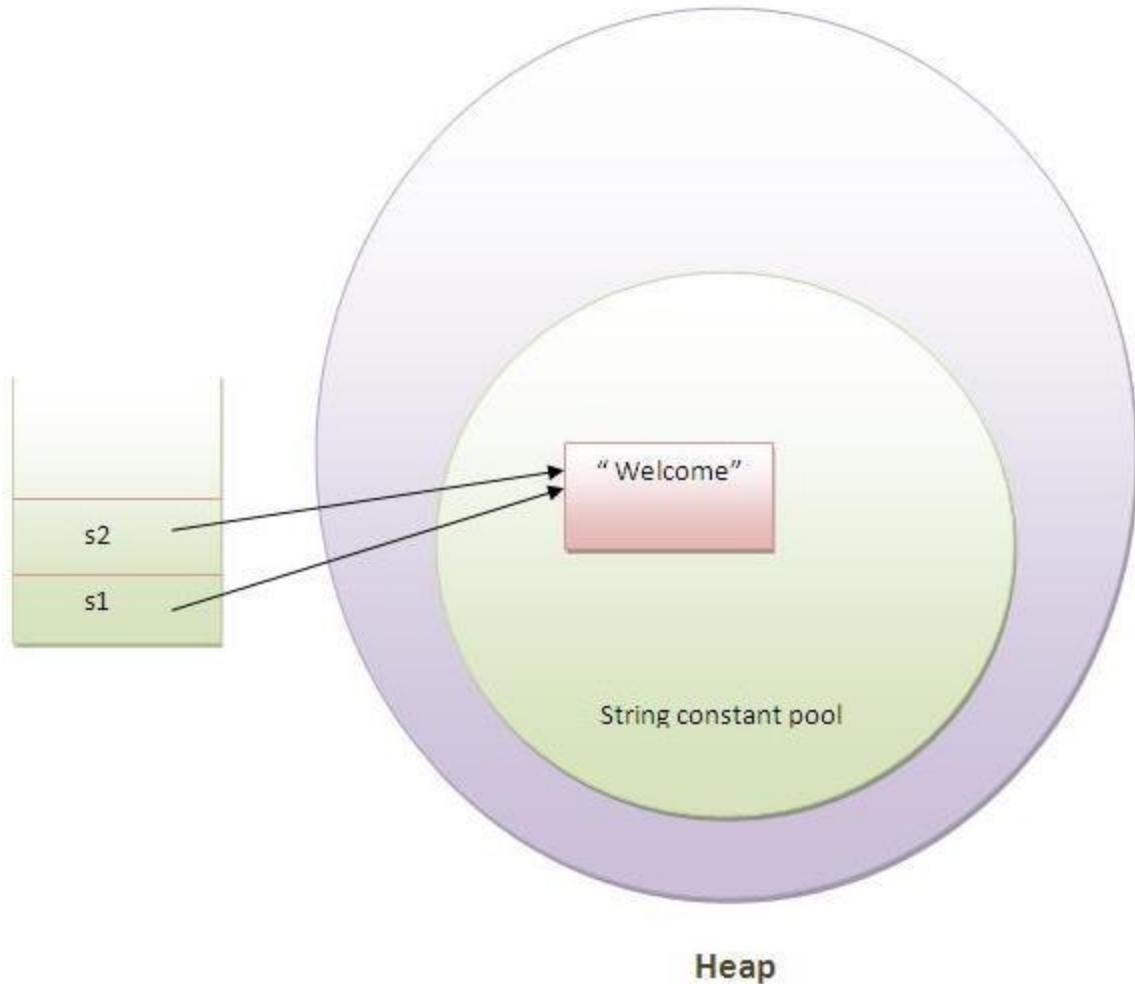
#### 1) String Literal

Java String literal is created by using double quotes. For Example:

1. `String s="welcome";`

Each time you create a string literal, the JVM checks the string constant pool(SCP) first. If the string already exists in the pool, a reference to the pooled instance is returned. If string doesn't exist in the pool, a new string instance is created and placed in the pool. For example:

1. String s1="Welcome";
2. String s2="Welcome";//will not create new instance



In the above example only one object will be created. Firstly JVM will not find any string object with the value "Welcome" in string constant pool, so it will create a new object. After that it will find the string with the value "Welcome" in the pool, it will not create new object but will return the reference to the same instance.

**Note: String objects are stored in a special memory area known as string constant pool.**

Why java uses concept of string literal?

To make Java more memory efficient (because no new objects are created if it exists already in string constant pool).

2) By new keyword

1. String s=**new** String("Welcome");//creates two objects and one reference variable

In such case, JVM will create a new string object in normal(non pool) heap memory and the literal "Welcome" will be placed in the string constant pool. The variable s will refer to the object in heap(non pool).

### Java String Example

1. **public class** StringExample{
2. **public static void** main(String args[]){
3. String s1="java";//creating string by java string literal
4. **char** ch[]={'s','t','r','i','n','g','s'};
5. String s2=**new** String(ch);//converting char array to string
6. String s3=**new** String("example");//creating java string by new keyword
7. System.out.println(s1);
8. System.out.println(s2);
9. System.out.println(s3);
10. }}

Output:

```
java
strings
example
```

### Java String class methods

The `java.lang.String` class provides many useful methods to perform operations on sequence of char values.

| No. | Method   | Description                                 |
|-----|--|---|
| 1   | <u><a href="#">char charAt(int index)</a></u>                              | returns char value for the particular index |
| 2   | <u><a href="#">int length()</a></u>  | returns string length                       |
| 3   | <u><a href="#">static String format(String format, Object... args)</a></u> | returns formatted string                    |

|    |  |   |
|----|--|---|
| 4  | <code>static String format(Locale l, String format, Object... args)</code>                               | returns formatted string with given locale                      |
| 5  | <code>String substring(int beginIndex)</code>  | returns substring for given begin index                         |
| 6  | <code>String substring(int beginIndex, int endIndex)</code>  | returns substring for given begin index and end index           |
| 7  | <code>boolean contains(CharSequence s)</code>  | returns true or false after matching the sequence of char value |
| 8  | <code>static String join(CharSequence delimiter, CharSequence... elements)</code>                        | returns a joined string   |
| 9  | <code>static String join(CharSequence delimiter, Iterable&lt;? extends CharSequence&gt; elements)</code> | returns a joined string   |
| 10 | <code>boolean equals(Object another)</code>  | checks the equality of string with object                       |
| 11 | <code>boolean isEmpty()</code>   | checks if string is empty                                       |
| 12 | <code>String concat(String str)</code>   | concatinates specified string                                   |
| 13 | <code>String replace(char old, char new)</code>  | replaces all occurrences of specified char value                |
| 14 | <code>String replace(CharSequence old, CharSequence new)</code>  | replaces all occurrences of specified CharSequence              |
| 15 | <code>static String equalsIgnoreCase(String another)</code>  | compares another string. It doesn't check case.                 |
| 16 | <code>String[] split(String regex)</code>  | returns splitted string matching regex                          |
| 17 | <code>String[] split(String regex, int limit)</code>   | returns splitted string matching regex and limit                |
| 18 | <code>String intern()</code>   | returns interned string   |
| 19 | <code>int indexOf(int ch)</code>   | returns specified char value index                              |
| 20 | <code>int indexOf(int ch, int fromIndex)</code>  | returns specified char value index starting with given index    |

|    |   |   |
|----|---|---|
| 21 | <u>int indexOf(String substring)</u>                | returns specified substring index                           |
| 22 | <u>int indexOf(String substring, int fromIndex)</u> | returns specified substring index starting with given index |
| 23 | <u>String toLowerCase()</u>                         | returns string in lowercase.                                |
| 24 | <u>String toLowerCase(Locale l)</u>                 | returns string in lowercase using specified locale.         |
| 25 | <u>String toUpperCase()</u>                         | returns string in uppercase.                                |
| 26 | <u>String toUpperCase(Locale l)</u>                 | returns string in uppercase using specified locale.         |
| 27 | <u>String trim()</u>                                | removes beginning and ending spaces of this string.         |
| 28 | <u>static String valueOf(int value)</u>             | converts given type into string. It is overloaded.          |

## Immutable String in Java

In java, **string objects are immutable**. Immutable simply means unmodifiable or unchangeable.

Once string object is created its data or state can't be changed but a new string object is created.

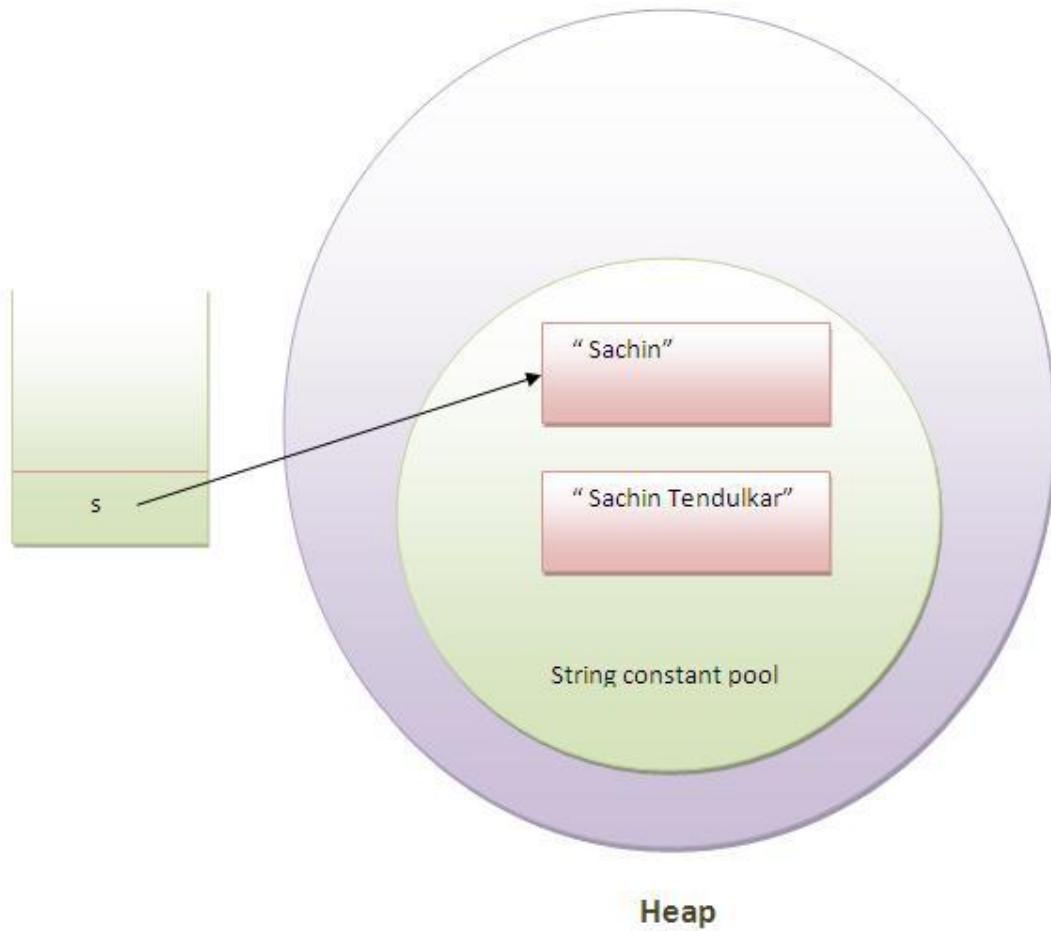
Let's try to understand the immutability concept by the example given below:

```

1. class Testimmutablestring{
2.     public static void main(String args[]){
3.         String s="Sachin";
4.         s.concat(" Tendulkar");//concat() method appends the string at the end
5.         System.out.println(s);//will print Sachin because strings are immutable objects
6.     }
7. }
```

Output:Sachin

Now it can be understood by the diagram given below. Here Sachin is not changed but a new object is created with sachintendulkar. That is why string is known as immutable.



As you can see in the above figure that two objects are created but `s` reference variable still refers to "Sachin" not to "Sachin Tendulkar".

But if we explicitly assign it to the reference variable, it will refer to "Sachin Tendulkar" object. For example:

```

1. class Testimmutablestring1{
2.     public static void main(String args[]){
3.         String s="Sachin";
4.         s=s.concat(" Tendulkar");
5.         System.out.println(s);
6.     }
7. }
```

Output:

Sachin Tendulkar

In such case, s points to the "Sachin Tendulkar". Please notice that still sachin object is not modified.

## Why string objects are immutable in java?

Because java uses the concept of string literal. Suppose there are 5 reference variables, all refers to one object "sachin". If one reference variable changes the value of the object, it will be affected to all the reference variables. That is why string objects are immutable in java.

## Java String compare

We can compare string in java on the basis of content and reference.

It is used in **authentication** (by equals() method), **sorting** (by compareTo() method), **reference matching** (by == operator) etc.

There are three ways to compare string in java:

1. By equals() method
2. By == operator
3. By compareTo() method

### 1) String compare by equals() method

The String equals() method compares the original content of the string. It compares values of string for equality. String class provides two methods:

- o **public boolean equals(Object another)** compares this string to the specified object.
- o **public boolean equalsIgnoreCase(String another)** compares this String to another string, ignoring case.

```
1. class Teststringcomparison1{  
2.     public static void main(String args[]){  
3.         String s1="Sachin";  
4.         String s2="Sachin";  
5.         String s3=new String("Sachin");  
6.         String s4="Saurav";  
7.         System.out.println(s1.equals(s2));//true
```

```
8. System.out.println(s1.equals(s3));//true
9. System.out.println(s1.equals(s4));//false
10. }
11. }
```

Output:

```
true
true
false
```

```
1. class Teststringcomparison2{
2. public static void main(String args[]){
3. String s1="Sachin";
4. String s2="SACHIN";
5.
6. System.out.println(s1.equals(s2));//false
7. System.out.println(s1.equalsIgnoreCase(s3));//true
8. }
9. }
```

Output:false

```
true
```

## 2) String compare by == operator

The == operator compares references not values.

```
1. class Teststringcomparison3{
2. public static void main(String args[]){
3. String s1="Sachin";
4. String s2="Sachin";
5. String s3=new String("Sachin");
6. System.out.println(s1==s2);//true (because both refer to same instance)
7. System.out.println(s1==s3);//false(because s3 refers to instance created in nonpool)
8. }
9. }
```

Output:

true  
false

---

### 3) String compare by compareTo() method

The String compareTo() method compares values lexicographically and returns an integer value that describes if first string is less than, equal to or greater than second string.

Suppose s1 and s2 are two string variables. If:

- o **s1 == s2** :0
- o **s1 > s2** :positive value
- o **s1 < s2** :negative value

1. **class** Teststringcomparison4{
2. **public static void** main(String args[]){
3.   String s1="Sachin";
4.   String s2="Sachin";
5.   String s3="Ratan";
6.   System.out.println(s1.compareTo(s2));//0
7.   System.out.println(s1.compareTo(s3));//1(because s1>s3)
8.   System.out.println(s3.compareTo(s1));//-1(because s3 < s1 )
9. }
10. }

#### Test it Now

Output:0

1  
-1

## String Concatenation in Java

In java, string concatenation forms a new string *that is* the combination of multiple strings. There are two ways to concat string in java:

1. By + (string concatenation) operator
2. By concat() method

### 1) String Concatenation by + (string concatenation) operator

Java string concatenation operator (+) is used to add strings. For Example:

```
1. class TestStringConcatenation1{  
2.     public static void main(String args[]){  
3.         String s="Sachin"+" Tendulkar";  
4.         System.out.println(s); //Sachin Tendulkar  
5.     }  
6. }
```

Output:Sachin Tendulkar

The **Java compiler transforms** above code to this:

```
1. String s=(new StringBuilder()).append("Sachin").append(" Tendulkar").toString();
```

In java, String concatenation is implemented through the `StringBuilder` (or `StringBuffer`) class and its `append` method. String concatenation operator produces a new string by appending the second operand onto the end of the first operand. The string concatenation operator can concat not only string but primitive values also. For Example:

```
1. class TestStringConcatenation2{  
2.     public static void main(String args[]){  
3.         String s=50+30+"Sachin"+40+40;  
4.         System.out.println(s); //80Sachin4040  
5.     }  
6. }
```

**Output:**

80Sachin4040

**Note: After a string literal, all the + will be treated as string concatenation operator.**

## 2) String Concatenation by concat() method

The String concat() method concatenates the specified string to the end of current string. Syntax:

1. **public** String concat(String another)

Let's see the example of String concat() method.

```
1. class TestStringConcatenation3{  
2.     public static void main(String args[]){  
3.         String s1="Sachin ";  
4.         String s2="Tendulkar";  
5.         String s3=s1.concat(s2);  
6.         System.out.println(s3);//Sachin Tendulkar  
7.     }  
8. }
```

### Output

```
Sachin Tendulkar
```

## Nested Classes in Java

A **nested class** is a class defined within another class. It is used to logically group classes that are only used in a specific context, improving code organization and encapsulation.

### Types of Nested Classes

Java has two main types of nested classes:

1. **Static Nested Classes**
2. **Non-static Nested Classes (Inner Classes)**
  - o Member Inner Class
  - o Local Inner Class
  - o Anonymous Inner Class

#### 1. Static Nested Class

A **static nested class** is a static class inside another class. Since it is static, it does not have access to the outer class's instance variables or methods.

##### Example: Static Nested Class

```
class Outer {  
    static class StaticNested {  
        void display() {  
            System.out.println("Inside Static Nested Class");  
        }  
    }  
}  
  
public class Test {  
    public static void main(String[] args) {  
        Outer.StaticNested obj = new Outer.StaticNested(); // No need to create an instance of Outer  
        obj.display();  
    }  
}
```

## **Output:**

Inside Static Nested Class

## **2. Non-static Nested Classes (Inner Classes)**

Inner classes are associated with an instance of the outer class and have access to its members.

### **2.1 Member Inner Class**

A **member inner class** is a non-static class inside another class. It can access all members of the outer class, including private members.

#### **Example: Member Inner Class**

```
class Outer {  
    private String message = "Hello from Outer Class";  
  
    class Inner {  
        void display() {  
            System.out.println(message); // Accessing outer class private variable  
        }  
    }  
}  
  
public class Test {  
    public static void main(String[] args) {  
        Outer outer = new Outer();  
        Outer.Inner inner = outer.new Inner(); // Creating instance of Inner class  
        inner.display();  
    }  
}
```

## **Output:**

Hello from Outer Class

## 2.2 Local Inner Class

A **local inner class** is defined inside a method or a block. It can only be used within that method.

### Example: Local Inner Class

```
class Outer {  
    void outerMethod() {  
        class LocalInner {  
            void display() {  
                System.out.println("Inside Local Inner Class");  
            }  
        }  
        LocalInner local = new LocalInner(); // Creating an instance inside the method  
        local.display();  
    }  
}
```

```
public class Test {  
    public static void main(String[] args) {  
        Outer obj = new Outer();  
        obj.outerMethod();  
    }  
}
```

### Output:

Inside Local Inner Class

## 2.3 Anonymous Inner Class

An **anonymous inner class** is a class without a name, created for immediate use, usually for implementing an interface or extending a class.

### Example: Anonymous Inner Class

```
abstract class Anonymous {
```

```

abstract void show();

}

public class Test {
    public static void main(String[] args) {
        Anonymous obj = new Anonymous() { // Anonymous inner class
            void show() {
                System.out.println("Inside Anonymous Inner Class");
            }
        };
        obj.show();
    }
}

```

**Output:**

Inside Anonymous Inner Class

**Comparison: Static Nested Class vs. Inner Class**

| Feature                       | Static Nested Class   | Inner Class  |
|-------------------------------|---|--|
| Instance Dependency           | Does NOT require an instance of the outer class.                          | Requires an instance of the outer class.                         |
| Access to Outer Class Members | Can access only static members.   | Can access all members (static and non-static).                  |
| Object Creation               | OuterClass.StaticNested obj = new OuterClass.StaticNested();              | OuterClass.InnerClass obj = outerInstance.new InnerClass();      |
| Usage                         | Used for helper classes that don't need access to outer instance members. | Used when an inner class needs to interact with the outer class. |

## Case Study: Banking System Using Member Inner Class

A BankAccount class has an inner class Transaction that handles deposits and withdrawals.

```
class BankAccount {
```

```
    private double balance = 1000;
```

```
    class Transaction {
```

```
        void withdraw(double amount) {
```

```
            if (amount <= balance) {
```

```
                balance -= amount;
```

```
                System.out.println("Withdrawal successful! New Balance: " + balance);
```

```
            } else {
```

```
                System.out.println("Insufficient funds!");
```

```
            }
```

```
        }
```

```
}
```

```
public class Test {
```

```
    public static void main(String[] args) {
```

```
        BankAccount account = new BankAccount();
```

```
        BankAccount.Transaction transaction = account.new Transaction();
```

```
        transaction.withdraw(500);
```

```
}
```

```
}
```

Output:

Withdrawal successful! New Balance: 500.0

## **EXCEPTION HANDLING IN JAVA**

### **Errors**

#### **1. Compile time errors or Syntax errors**

missing;, misspelling, using variables without declaration, assigning values, underflow, overflow, data type mismatch, etc.

#### **2. Runtime errors**

Division by zero, invalid conversion, accessing an invalid index of array etc.

### **Exception**

- Runtime error results are an Exception.
- The exception is a representation of an error condition.
- Exception leads to abnormal termination of the program, giving some garbage message on the screen.
- **In Java, an exception is an event that disrupts the normal flow of the program. It is an object, which is thrown at runtime.**
- Whenever exceptions occur, java creates an object for the exception class or its deriving class.

### **Exception Handling:**

Exception Handling is a mechanism to handle runtime errors such as ClassNotFoundException, IOException, SQLException, RemoteException etc.

The advantage of exception handling is **that it maintains the normal flow of the application/program**. Exception normally disrupts the normal flow of the application that is why we use exception handling.

Example:

//Program

1. statement 1;
2. statement 2;
3. statement 3;
4. statement 4;
5. statement 5;//exception occurs
6. statement 6;
7. statement 7;
8. statement 8;
9. statement 9;
10. statement 10;

Suppose there are 10 statements in your program and there occurs an exception at statement 5, rest of the code will not be executed i.e. statement 6 to 10 will not run. If we perform exception handling, rest of the statement will be executed. That is why we use exception handling in java.

### Common Example of Exceptions:

If we divide any number by zero, there occurs an **ArithmaticException**.

```
int a=50;
```

```
int b=0;
```

```
int c=a/b;
```

1. int a=50/0;//ArithmaticException

If we have null value in any variable, performing any operation by the variable occurs a **NullPointerException**.

1. String s=null;
2. System.out.println(s.length());//NullPointerException

The wrong formatting of any value may occur `NumberFormatException`. Suppose I have a string variable that have characters, converting this variable into digit will occur `NumberFormatException`.

1. `String s="abc";`
2. `int i=Integer.parseInt(s); //NumberFormatException`

If you are inserting any value in the wrong index, it would result `ArrayIndexOutOfBoundsException` as shown below:

1. `int a[] = new int[5]; index 0-4`
2. `a[10]=50; //ArrayIndexOutOfBoundsException`

There are 5 keywords used in Java exception handling.

1. `try`
2. `catch`
3. `finally`
4. `throw`
5. `throws`

### **Java try block { .....**

Java try block is used to enclose the code that might throw an exception. It must be used within the method.

Java try block must be followed by either catch or finally block.

### **Java catch block**

Java catch block is used to handle the Exception. It must be used after the try block only.

You can use multiple catch block with a single try.

Example:

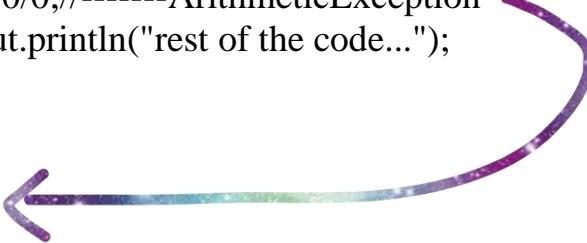
1. try{
2. //code that may throw an exception -----object of Exception class or it derive a class
3. }catch(Exception\_class\_Name ref){}

Example:

1. try{
2. //code that may throw exception
3. }finally{ }

### Program: Problem without Try Catch

```
1. public class Testtrycatch1{  
2.     public static void main(String args[]){  
3.         int data=50/0;-----ArithmaticException  
4.         System.out.println("rest of the code...");  
5.     }  
6. }
```



Output:

```
C:\Windows\System32\cmd.exe  
  
D:\1 Java\Programs>javac Testtrycatch1.java  
  
D:\1 Java\Programs>java Testtrycatch1  
Exception in thread "main" java.lang.ArithmaticException: / by zero  
at Testtrycatch1.main(Testtrycatch1.java:3)
```

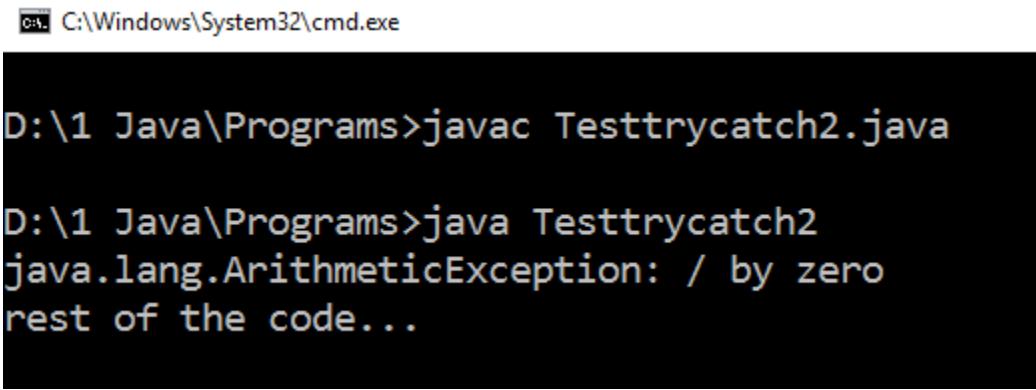
The code after the exception will not be executed.

```
//System.out.println("rest of the code...");
```

## Solution by exception handling

```
1. public class Testtrycatch2{  
2.   public static void main(String args[]){  
3.     try  
4.     {  
5.       int data=50/0; //it will throw an object of class ArithmeticException  
6.     }  
7.     catch(ArithmeticException e)// or catch(Exception e)  
8.     {  
9.       System.out.println(e);  
10.    }  
11.  System.out.println("rest of the code...");  
12. }
```

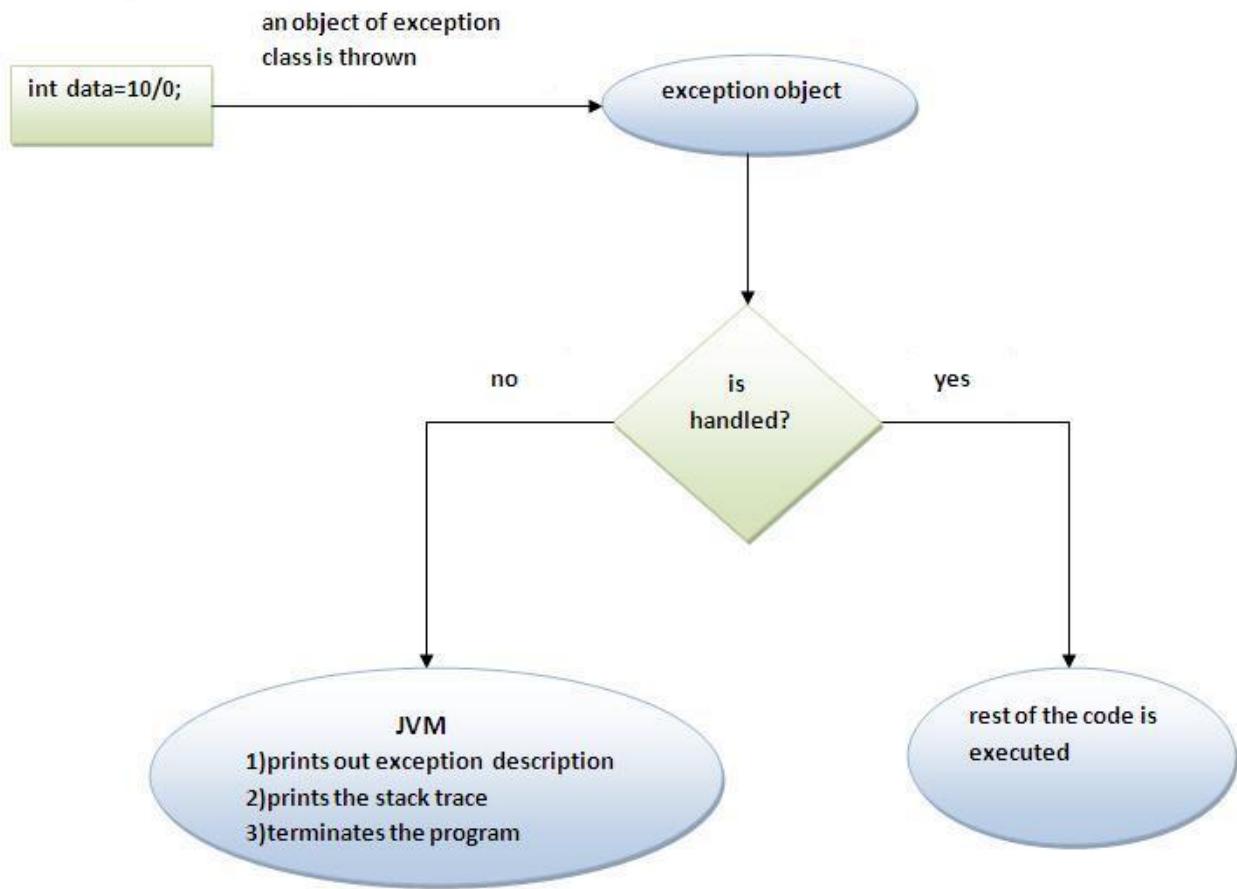
---



The screenshot shows a Windows Command Prompt window titled 'C:\Windows\System32\cmd.exe'. The command line shows the user navigating to 'D:\1 Java\Programs' and then running 'javac Testtrycatch2.java'. After compilation, the user runs 'java Testtrycatch2'. The output displays an 'ArithmeticException' message followed by the text 'rest of the code...'.

```
C:\Windows\System32\cmd.exe  
  
D:\1 Java\Programs>javac Testtrycatch2.java  
  
D:\1 Java\Programs>java Testtrycatch2  
java.lang.ArithmetricException: / by zero  
rest of the code...
```

## Working of java try-catch block



## Java Multi catch block

```
public class TestMultipleCatchBlock{
    public static void main(String args[]){
        try{
            int a[]={}; //index 0-4
            a[5]=30/0; //throw xxxxxxxxx Exception
        }
        catch(ArithmaticException e)
        {
            System.out.println("Catch ArithmaticException");
        }
        catch(ArrayIndexOutOfBoundsException e)
        {
            System.out.println("Catch ArrayIndexOutOfBoundsException");
        }
        catch(Exception e)
        {
            System.out.println("Catch All Exception");
        }

        System.out.println("continue with rest of the code...");
    }
}
```

```
C:\Windows\System32\cmd.exe

D:\1 Java\Programs>javac TestMultipleCatchBlock.java

D:\1 Java\Programs>java TestMultipleCatchBlock
Catch ArithmaticException
continue with rest of the code...
```

**Note:**

- At a time only one Exception occurs and at a time only one catch block is executed.

```
public class OrderofCatchBlock{
    public static void main(String args[]){
        try{
            int a[]=new int[5]; //index 0-4
            a[5]=30/0;
        }

        catch(Exception e)
        {
            System.out.println("Catch All Exception");
        }

        catch(ArithmaticException e)
        {
            System.out.println("Catch ArithmaticException");
        }

        catch(ArrayIndexOutOfBoundsException e)
        {
            System.out.println("Catch ArrayIndexOutOfBoundsException");
        }

        System.out.println("continue with rest of the code...");

    }
}
```

```
C:\Windows\System32\cmd.exe
OrderofCatchBlock.java:11: error: exception ArithmeticException
    has already been caught
        catch(ArithmeticException e)
        ^
OrderofCatchBlock.java:15: error: exception ArrayIndexOutOfBoundsException
    has already been caught
        catch(ArrayIndexOutOfBoundsException e)
        ^
2 errors
```

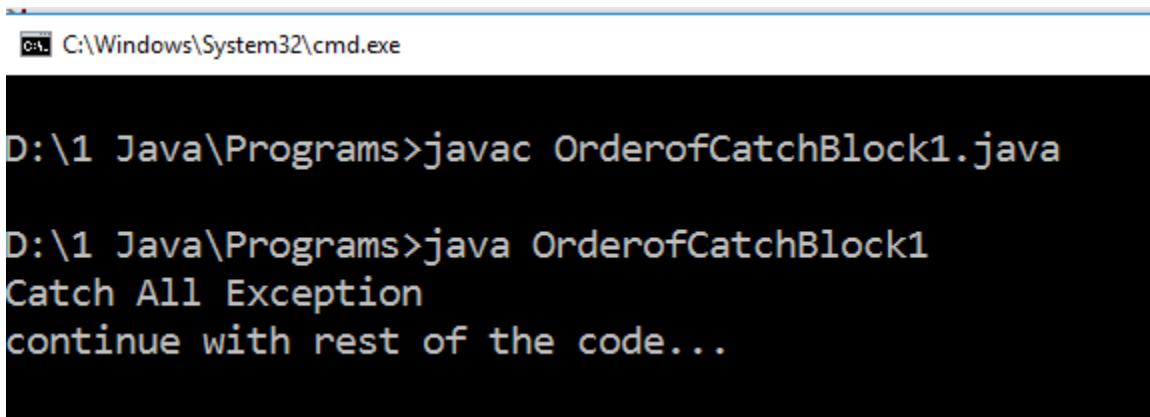
```
public class OrderofCatchBlock1{
    public static void main(String args[]){
        try{
            int a[]=new int[5];
            a[5]=30/0;
        }
        catch(Exception e)
        {
            System.out.println("Catch All Exception");
        }

        catch(ArrayIndexOutOfBoundsException e)
        {
            System.out.println("Catch ArrayIndexOutOfBoundsException");
        }
        System.out.println("continue with rest of the code... ");
    }
}
```

```
D:\1 Java\Programs>javac OrderofCatchBlock1.java
OrderofCatchBlock1.java:15: error: exception ArrayIndexOutOfBoundsException
    has already been caught
        catch(ArrayIndexOutOfBoundsException e)
        ^
1 error
```

```
class OrderofCatchBlock1{
    public static void main(String args[]){
        try{
            int a[]=new int[5]; ----→one exception object
            a[5]=30/0;
        }
        catch(Exception e)
        {
            System.out.println("Catch All Exception");
        }

        System.out.println("continue with rest of the code...");
    }
}
```



The screenshot shows a Windows Command Prompt window titled 'C:\Windows\System32\cmd.exe'. The command 'javac OrderofCatchBlock1.java' is run, followed by 'java OrderofCatchBlock1'. The output shows 'Catch All Exception' followed by 'continue with rest of the code...'. The window has a standard Windows title bar and a black background.

```
C:\Windows\System32\cmd.exe

D:\1 Java\Programs>javac OrderofCatchBlock1.java

D:\1 Java\Programs>java OrderofCatchBlock1
Catch All Exception
continue with rest of the code...
```

**Note:**

All catch blocks must be ordered from most specific (derive classes) to most general (base class) i.e. catch for ArithmeticException must come before catch for Exception.

```
public class TestMultipleCatchBlock{
    public static void main(String args[]){
        try{
            int a[]=new int[5]; //index 0-4
            a[5]=5; //throw xxxxxxxxx Exception
        }
        catch(ArithmeticException e)
        {
            System.out.println("Catch ArithmeticException");
        }
        catch(ArrayIndexOutOfBoundsException e)
        {
            System.out.println("Catch ArrayIndexOutOfBoundsException");
        }
        catch(Exception e)
        {
            System.out.println("Catch All Exception");
        }

        System.out.println("continue with rest of the code...");
    }
}
```

```
F:\Java Code>java TestMultipleCatchBlock
Catch ArrayIndexOutOfBoundsException
continue with rest of the code...
```

```
public class TestMultipleCatchBlock{
    public static void main(String args[]){
        try{
            int a[]=new int[5];
            a[5]=30/0; //throw xxxxxxxxx Exception

        }
        catch(ArithmeticException e)
        {
            System.out.println("Catch ArithmeticException");
        }
```

```
        }
    catch(ArrayIndexOutOfBoundsException e)
    {
        System.out.println("Catch ArrayIndexOutOfBoundsException");
    }
    catch(Exception e)
    {
        System.out.println("Catch All Exception");
    }

    System.out.println("continue with rest of the code...");
}
}
```

```
F:\Java Code>java TestMultipleCatchBlock
Catch ArithmeticException
continue with rest of the code...
```

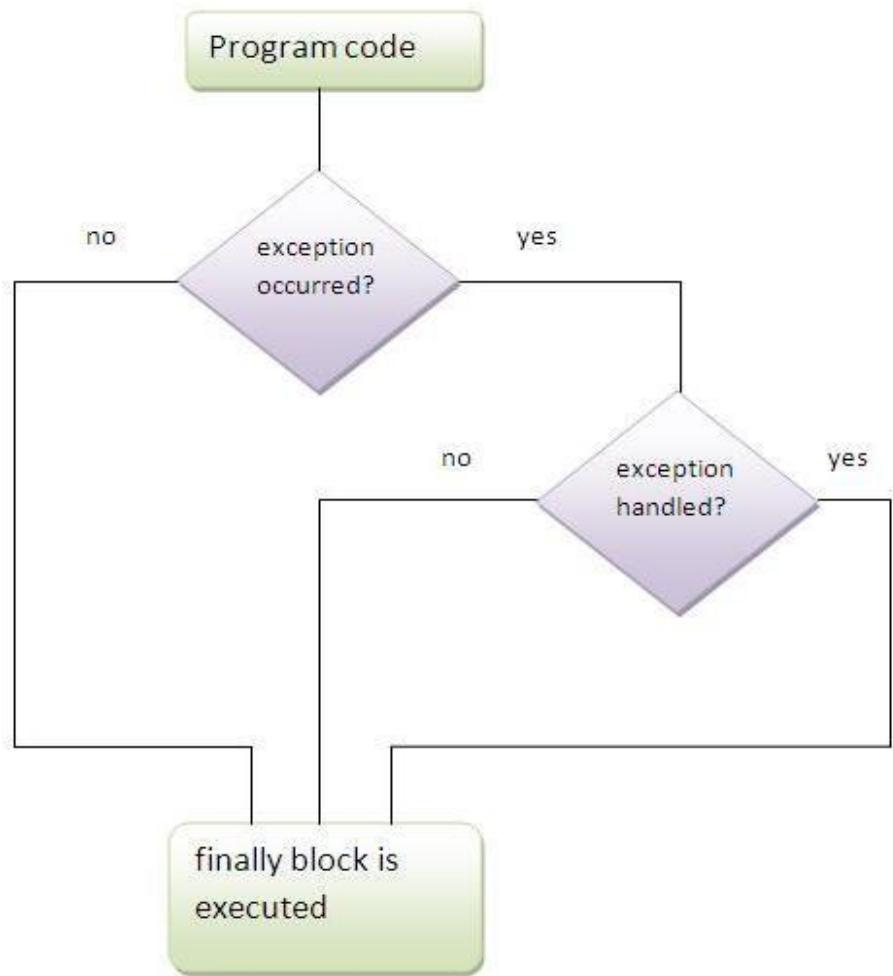
## Nested Try Blocks

```
class NestedTry{  
  
public static void main(String args[]){  
    try {  
        try{  
            System.out.println("Division of number occur");  
            int b =39/0;  
        }  
        catch(ArithmetricException e)  
        {  
            System.out.println(e);  
        }  
    try  
    {  
        int a[]={};  
        a[5]=4;  
    }  
    catch(ArrayIndexOutOfBoundsException e)  
    {  
        System.out.println(e);  
    }  
}  
catch(Exception e)  
{  
    System.out.println("It handle all type of exceptions");  
    System.out.println(e);  
}  
System.out.println("Program Continued..");  
}  
}
```

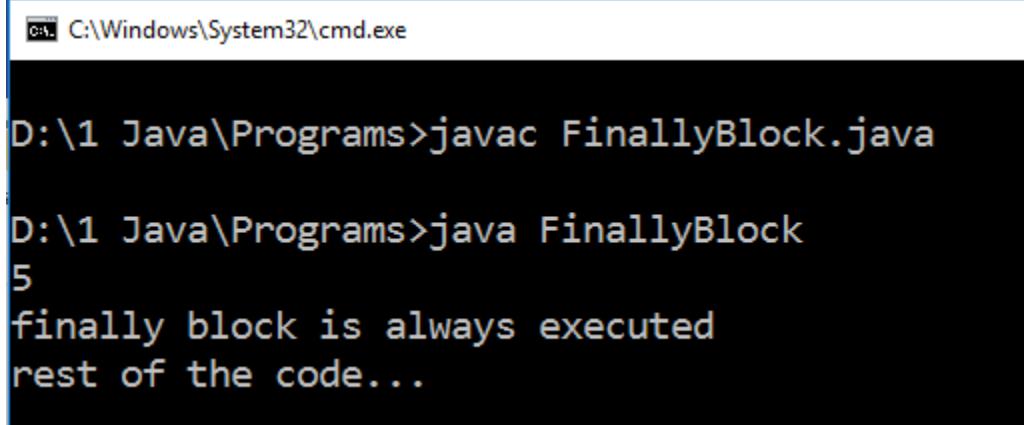
```
C:\Windows\System32\cmd.exe
D:\1 Java\Programs>javac NestedTry.java
D:\1 Java\Programs>java NestedTry
Division of number occur
java.lang.ArithmetricException: / by zero
java.lang.ArrayIndexOutOfBoundsException: 5
Program Continued..
```

## Java Finally Block

- Java finally block is a block that is used to execute important code such as closing connection, stream, etc.
- Finally, can be used as a substitute or along with a catch.
- There can be only one finally.
- Finally, does not receive any argument/object.
- Java finally block is always executed whether an exception is handled or not.
- If you don't handle the exception, before terminating the program, JVM executes the finally block (if any).
- The finally block will not be executed if the program exits (either by calling System.exit() or by causing a fatal error that causes the process to abort).



```
class FinallyBlock{
    public static void main(String args[]){
        try{
            int data=25/5; //
            System.out.println(data);
        }
        catch(NullPointerException e)
        {
            System.out.println(e);
        }
        finally
        {
            System.out.println("finally block is always executed");
        }
        System.out.println("rest of the code...");
    }
}
```



The screenshot shows a Windows Command Prompt window titled 'C:\Windows\System32\cmd.exe'. The command line shows two executions of Java code. The first execution is 'javac FinallyBlock.java', which compiles the Java source code into bytecode. The second execution is 'java FinallyBlock', which runs the compiled bytecode. The output of the program is displayed below the command line, showing the value '5' followed by the strings 'finally block is always executed' and 'rest of the code...'. The entire window is enclosed in a blue border.

```
C:\Windows\System32\cmd.exe

D:\1 Java\Programs>javac FinallyBlock.java

D:\1 Java\Programs>java FinallyBlock
5
finally block is always executed
rest of the code...
```

```
class FinallyBlock1{
    public static void main(String args[]){
        try{
            int data=25/0; // //throw
            System.out.println(data);
        }
        catch(ArithmeticException e)
        {
            System.out.println(e);
        }
        finally{
            System.out.println("finally block is always executed");
        }
        System.out.println("rest of the code...");
```

```
D:\1 Java\Programs>java FinallyBlock1
java.lang.ArithmetricException: / by zero
finally block is always executed
rest of the code...
```

```
class FinallyBlock1{
    public static void main(String args[]){
        try{
            int data=25/0; //
            System.out.println(data);
        }
        catch(NullPointerException e)
        {
            System.out.println(e);
        }
        finally
        {
            System.out.println("finally block is always executed");
        }
        System.out.println("rest of the code...");
```

```
}
```

```
C:\Windows\System32\cmd.exe
```

```
D:\1 Java\Programs>javac FinallyBlock1.java
```

```
D:\1 Java\Programs>java FinallyBlock1
```

```
finally block is always executed
```

```
Exception in thread "main" java.lang.ArithmetricException: / by  
zero
```

```
        at FinallyBlock1.main(FinallyBlock1.java:4)
```

finally block without try block ☒

Multiple finally block ☒

## **throw Keyword in Java**

Basically, the exceptions are thrown by Java runtime systems. It is not all the time possible for a Java runtime system to throw the exception. This burden may come on the programmer's shoulders. That means sometimes we must explicitly mention the throw of exception. The throw keyword in Java is used to explicitly throw an exception from a method or any block of code.

Syntax: throw exception

### **Program:**

```
import java.util.*;
class TestThrow
{
    public static void main(String args[])
    {
        System.out.println("Enter a even number");
        Scanner s=new Scanner(System.in);
        int n=s.nextInt();
        if(n%2==0)
        {
            System.out.println("Number " +n +" is even");
        }
        else
        {
            throw new ArithmeticException("Invalid Number");
        }
    }
}
```

```
c:\ C:\Windows\System32\cmd.exe

D:\1 Java\Programs>javac TestThrow.java

D:\1 Java\Programs>java TestThrow
Enter a even number
24
Number 24 is even

D:\1 Java\Programs>java TestThrow
Enter a even number
23
Exception in thread "main" java.lang.ArithmaticException: Invalid Number
at TestThrow.main(TestThrow.java:15)
```

**Note:** After the throw statement, no other statement should be placed in that block. If any statement is placed after the throw statement in that block, Java will give an **unreachable statement**, compile time error.

```
import java.util.*;
class TestThrow
{
    public static void main(String args[])
    {
        System.out.println("Enter an even number");
        Scanner s=new Scanner (System.in);
        int n=s.nextInt();
        if(n%2==0)
        {
            System.out.println("Number " +n +" is even");
        }
        else
        {
            throw new ArithmaticException("Invalid Number");
            System.out.println("rest of program");
        }
    }
}
```

```
D:\1 Java\Programs>javac TestThrow.java
TestThrow.java:16: error: unreachable statement
        System.out.println("rest of program");
                  ^
1 error
```

## **throws:**

To prevent this compile time error, we can handle the exception in two ways:

1. By using try catch (Already Discussed)
2. By using the **throws** keyword

- throws is a keyword in Java that is used in the signature of the method to indicate that this method might throw one of the listed type exceptions.
- The caller to these methods must handle the exception using a try-catch block.
- throws keyword is required only to convince the compiler and usage of the throws keyword does not prevent abnormal termination of the program.
- With the help of the throws keyword, we can provide information to the caller about the method of the exception.
- throws keyword is required only for checked exceptions and usage of the throws keyword for unchecked exceptions (run time exception) is meaningless.

## **Syntax:**

**Return type method\_name (parameters) throws exception\_list**

An exception list is a comma-separated list of all the exceptions that a method might throw.

We can use the throws keyword to delegate the responsibility of exception handling to the caller (It may be a method or JVM) then caller method is responsible to handle that exception.

```
import java.util.*;
class ExThrows0
{
    static void divideNum()
    {
        int a=50;
        System.out.println("Enter the value of b");
        Scanner s=new Scanner(System.in);
        int b=s.nextInt();
        int c=a/b;
        System.out.println("Value of c is:"+c);
    }
    public static void main(String args[])
    {
        divideNum();
        System.out.println("Rest of the code");
    }
}
```

```
C:\Windows\System32\cmd.exe

D:\1 Java\Programs>javac ExThrows0.java

D:\1 Java\Programs>java ExThrows0
Enter the value of b
5
Value of c is:10
Rest of the code

D:\1 Java\Programs>java ExThrows0
Enter the value of b
0
Exception in thread "main" java.lang.ArithmeticException: / by zero
        at ExThrows0.divideNum(ExThrows0.java:10)
        at ExThrows0.main(ExThrows0.java:16)
```

```
import java.util.*;
class ExThrows1
{
    static void divideNum() throws ArithmeticException
    {
        int a=50;
        System.out.println("Enter the value of b");
        Scanner s=new Scanner(System.in);
        int b=s.nextInt();
        int c=a/b;
        System.out.println("Value of c is:"+c);
    }

    public static void main(String args[])
    {
        try{
            divideNum();
        }
        catch(ArithmeticException e)
        {
            System.out.println("caught Exception"+e);
        }
        System.out.println("Rest of the code");
    }
}
```

C:\Windows\System32\cmd.exe

```
D:\1 Java\Programs>javac ExThrows1.java
```

```
D:\1 Java\Programs>java ExThrows1
```

```
Enter the value of b
```

```
5
```

```
Value of c is:10
```

```
Rest of the code
```

```
D:\1 Java\Programs>java ExThrows1
```

```
Enter the value of b
```

```
0
```

```
caught Exceptionjava.lang.ArithmetricException: / by zero
```

```
Rest of the code
```

## **Java Custom/User Define Exception**

If you are creating your own Exception that is known as a custom exception or user-defined exception. Java custom exceptions are used to customize the exception according to user needs. With the help of custom exceptions, you can have your own exception and message.

The general form of creating an exception is:

```
Throwable t = new Throwable("my Exception");
Exception t=new Exception("my Exception");
```

### **Constructors:**

- Throwable();
- Throwable(String s);
- Exception();
- Exception (String s);

```
class CustomException
{
public static void main(String args[])
{
Throwable t = new Throwable("CustomException");
try
{
System.out.println("\n A new exception is thrown");
throw t;
}
catch(Throwable e)
{
System.out.println("\n The exception is caught here.");
System.out.println("\n The exception is:"+e);
} }}
```

```
C:\Windows\System32\cmd.exe

D:\1 Java\Programs>javac CustomException.java

D:\1 Java\Programs>java CustomException

A new exception is thrown

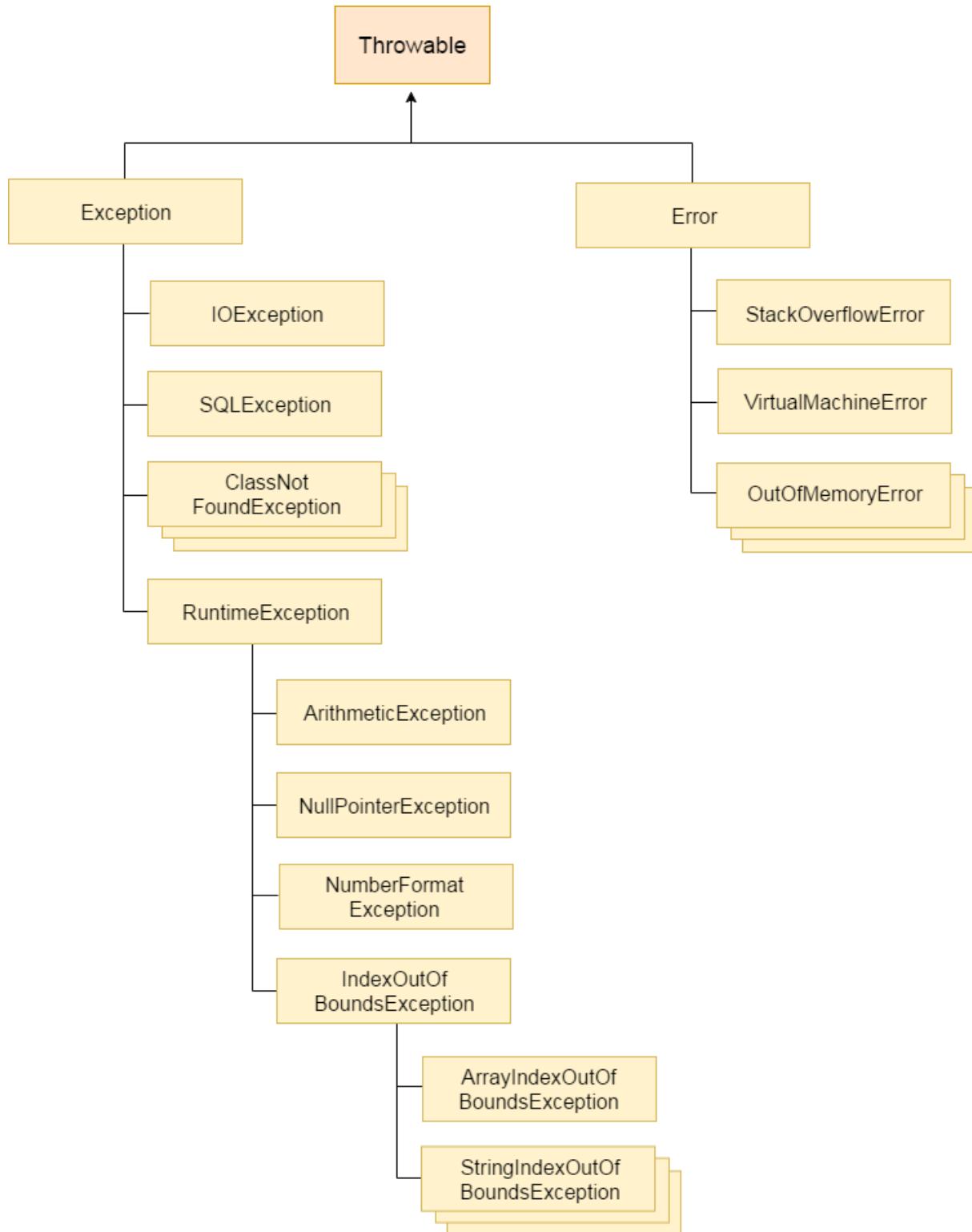
The exception is caught here.

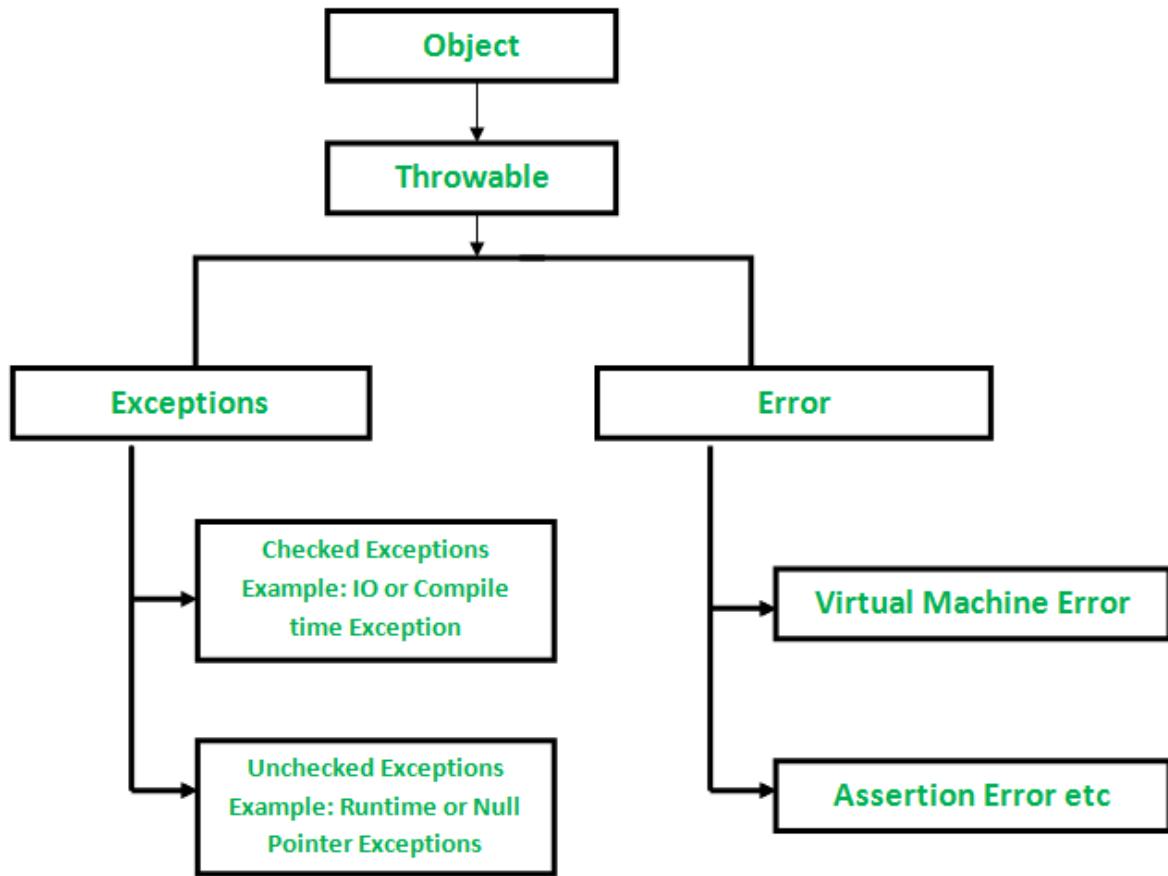
The exception is:java.lang.Throwable: CustomException
```

```
class CustomException
{
public static void main(String args[])
{
Exception t = new Exception("CustomException");
try
{
System.out.println("\n A new exception is thrown");
throw t;
}
catch(Exception e)
{
System.out.println("\n The exception is caught here.");
System.out.println("\n The exception is:"+e);
}
}
}
```

```
D:\1 Java\Programs>javac CustomException.java  
D:\1 Java\Programs>java CustomException  
A new exception is thrown  
The exception is caught here.  
The exception is:java.lang.Exception: CustomException
```

## Hierarchy of Java Exception classes





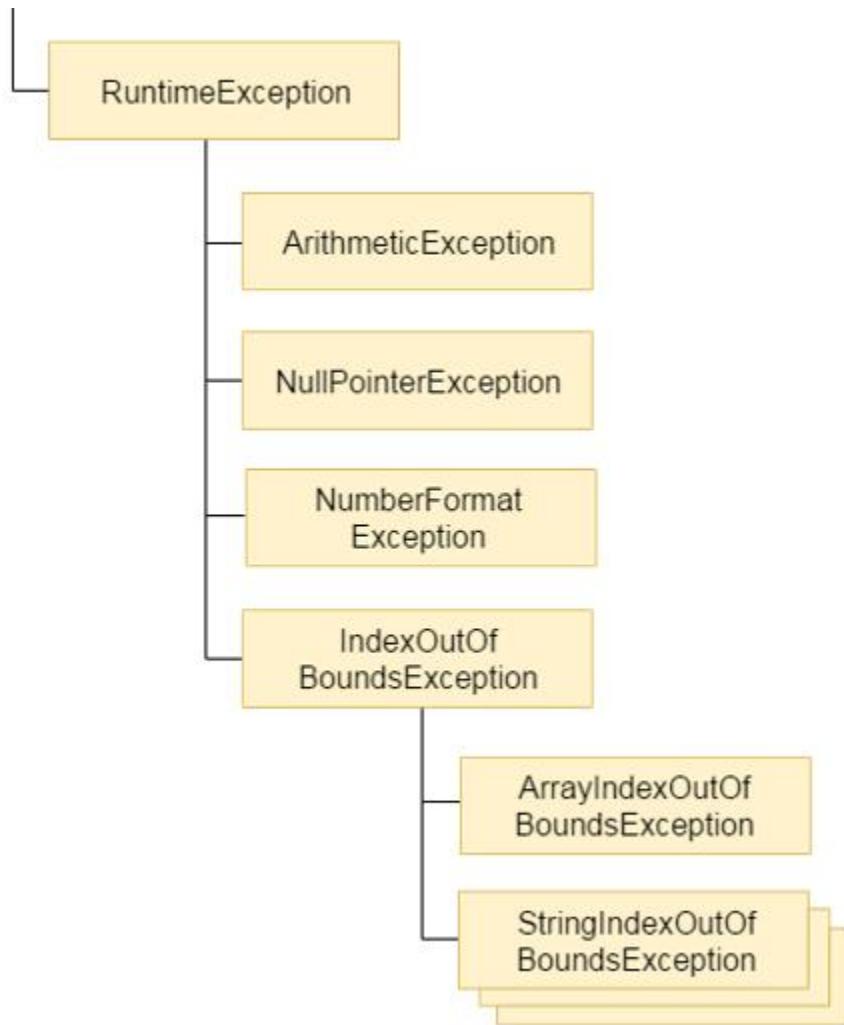
## Checked vs. Unchecked Exceptions in Java

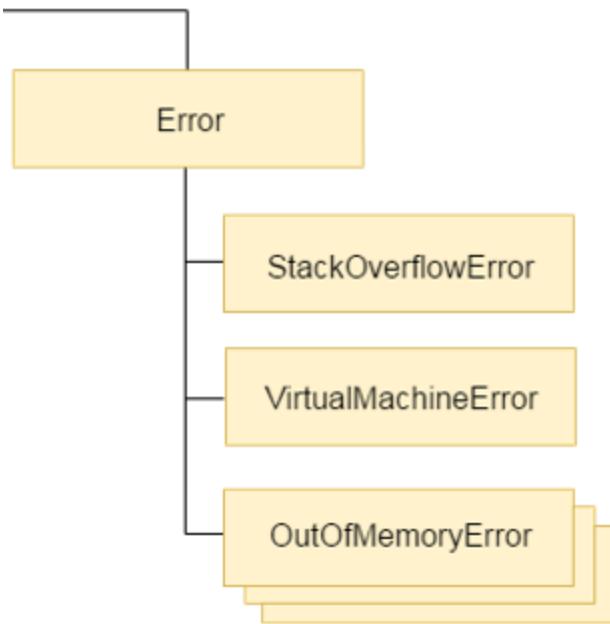
In Java, there are two types of exceptions:

**1) Checked:** are the exceptions that are checked at compile time. If some code within a method throws a checked exception, then the method must either handle the exception or it must specify the exception using *throws* keyword.

- IOException
- SQLException
- ClassNotFoundException

**2) Unchecked** are the exceptions that are not checked at compiled time. In C++, all exceptions are unchecked, so it is not forced by the compiler to either handle or specify the exception. It is up to the programmers to be civilized, and specify or catch the exceptions.





# Multithreading in Java

## Multitasking

- Multitasking is a process of executing multiple tasks simultaneously.
- We use multitasking to utilize the CPU.
- To reduce processor ideal time and to improve performance.

Multitasking can be achieved in two ways:

- Process-based Multitasking (Multiprocessing)
- Thread-based Multitasking (Multithreading)

### 1) Process-based Multitasking (Multiprocessing)

- Each process has its own address in memory i.e., each process allocates a separate memory area.
- Process is heavyweight.
- Cost of communication between the processes is high.
- Switching from one process to another requires time to save and load registers, memory maps, and updated lists.

### 2) Thread-based Multitasking (Multithreading)

- Threads share the same address space. Example MSWORD typing-t1, autosave-t2, spellcheck-t3,page number -t4.....2048(RAM)
- Thread is lightweight.
- Cost of communication between the threads is low.

**Note:** Thread is executed inside the process. There is context-switching between the threads. There can be multiple processes inside the OS, and one process can have multiple threads.

**Multithreading in java** is a process of executing multiple threads simultaneously. Thread is basically a lightweight sub-process, the smallest unit of processing. Multiprocessing and multithreading are both used to achieve multitasking. But we use multithreading than multiprocessing because threads share a common memory area. They don't allocate separate memory areas, save memory, and context-switching between the threads takes less time than the process.

## Advantages of Threading

- Better utilization of system resources, including the CPU.
- Reduces the computation time.
- Improves the performance of an application.

- Threads share the same address space, so it saves memory.
- Context switching between threads is usually less expensive than between processes.
- Cost of communication between threads is relatively low.
- It **doesn't block the user** because threads are independent, and you can perform multiple operations at the same time.
- You **can perform many operations together, so it saves time.**
- Threads are **independent**, so it doesn't affect other threads if exceptions occur in a single thread.

## Thread States or Life cycle of a Thread

JVM controls the life cycle of the thread in java. When any thread is created, it goes to different states before it completes its task and is dead. The different states are:

- 1) New
- 2) Runnable
- 3) Running
- 4) Non-Runnable (Blocked)
- 5) Terminated

**1) New/Born:** When a thread is created, it is in a new state; in this state, thread will not be executed and not share time with the processor.

**2) Runnable/Ready:** When the start() method is called on the thread object, the thread is in a runnable state. In this state, the thread executes and shares time with the processor.

**3) Running:** If the thread scheduler allocates a processor to a thread, it will go into the Running state OR A thread currently being executed by the CPU is in the running state.

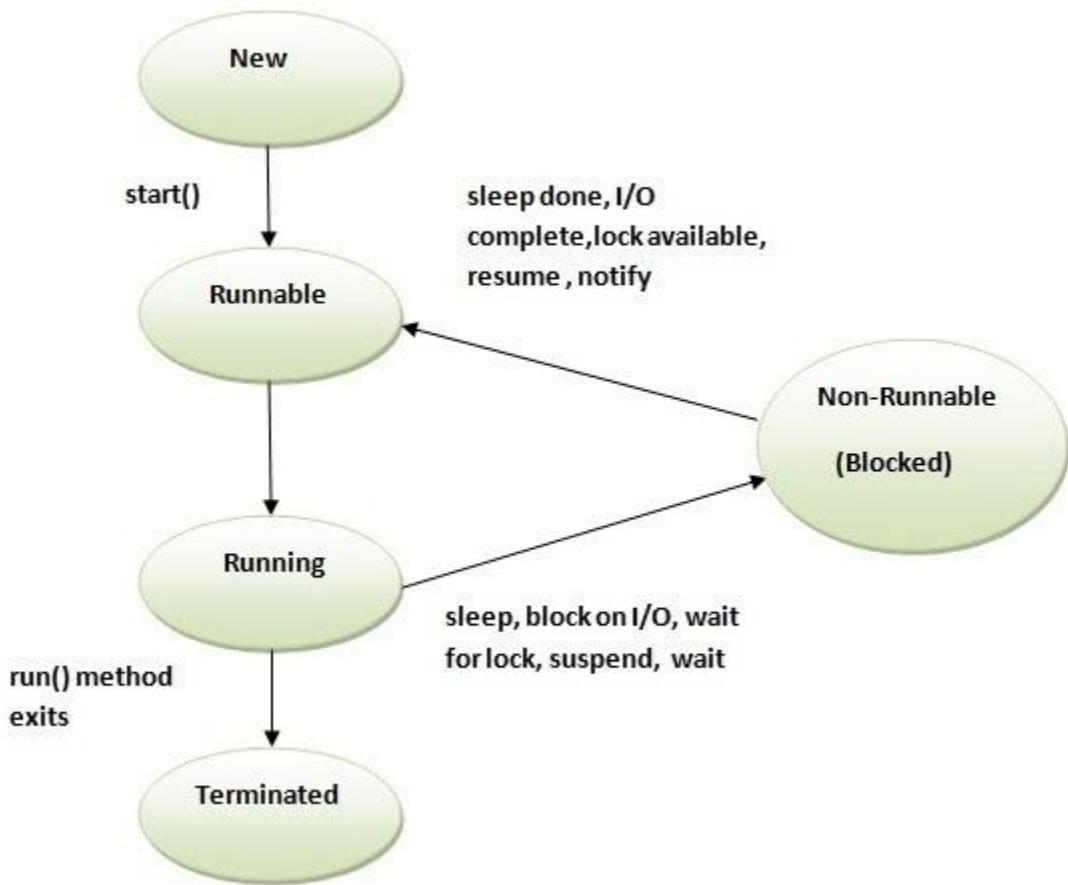
**4) Non-Runnable (Blocked):** This is the state in which the thread is still alive but is currently not eligible to run. A running thread may go to a blocked state due to the following conditions.

- wait() or sleep() method is called
- The thread performs I/O operation.

When a blocked thread is unblocked, it goes to a runnable state and not to a running state.

**5) Terminated:** A thread is in the terminated or dead state when its run() method exits. A thread becomes dead on two occasions.

- If a thread completes its task, exit the running state.
- run() method is aborted(due to exception etc.)



**Note:** According to sun microsystem, there are only 4 states in the thread life cycle in java new, runnable, non-runnable, and terminated. There is **no running** state.

## CREATION OF THREADS

There are two ways to create a thread:

1. By extending the Thread class
2. By implementing a Runnable interface.

## **1. By extending the Thread class:**

The Thread class provides constructors and methods to create and perform operations on a thread. Thread class extends Object class and implements Runnable interface.

Commonly used Constructors of Thread class:

- Thread()
- Thread(String name)
- Thread(Runnable r)
- Thread(Runnable r, String name)

Commonly used methods of Thread class:

**public void run ()**: is used to perform an action for a thread.

**public void start()**: starts the execution of the thread. JVM calls the run() method on the thread.

**public void sleep (long milliseconds)**: Causes the currently executing thread to sleep (temporarily cease execution) for the specified number of milliseconds.

**public void join()**: waits for a thread to die.

**public void join(long milliseconds)**: waits for a thread to die for the specified milliseconds.

**public int getPriority()**: returns the priority of the thread.

**public int setPriority(int priority)**: changes the thread's priority.

**public String getName()**: returns the name of the thread.

**public void setName(String name)**: changes the thread's name.

**public Thread currentThread()**: returns the reference of the currently executing thread.

**public int getId()**: returns the id of the thread.

**public Thread.State getState()**: returns the state of the thread.

**public boolean isAlive()**: tests if the thread is alive.

**public void yield()**: causes the currently executing thread object to temporarily pause and allow other threads to execute.

**public void suspend()**: is used to suspend the thread(deprecated).

**public void resume()**: is used to resume the suspended thread(deprecated).

**public void stop()**: is used to stop the thread(deprecated).

**public boolean isDaemon()**: tests if the thread is a daemon thread.

**public void setDaemon(boolean b)**: marks the thread as daemon or user thread.

**public void interrupt()**: interrupts the thread.

**public boolean isInterrupted()**: tests if the thread has been interrupted.

**public static boolean interrupted()**: tests if the current thread has been interrupted

## Simple Example of Thread creation

```
class MyThread extends Thread{  
    public void run()  
{  
    System.out.println("thread is running...");  
}  
public static void main(String args[]){  
    MyThread t1=new MyThread();  
    t1.run();  
}  
}
```

C:\Windows\System32\cmd.exe

```
D:\1 Java\Programs>javac MyThread.java  
  
D:\1 Java\Programs>java MyThread  
thread is running...
```

## **Example 2:Thread 1,Thread 2 using run() method (call by user).**

```
class MyThread1 extends Thread
{
public void run()
{
for (int i=1;i<=10;i++)
    {System.out.println("Running Thread1:"+i);
 }}
```

```
class MyThread2 extends Thread
{
public void run()
{
for (int i=11;i<=20;i++)
    {System.out.println("Running Thread2:"+i);
 }}
```

```
class TestThread
{
public static void main(String arg[])
{
    MyThread1 mt1=new MyThread1();
    mt1.run(); //1-10
    MyThread2 mt2=new MyThread2();
    mt2.run(); //11-20
}}
```

1) run() ← CPU –cpu exit

2) run() ← CPU

C:\Windows\System32\cmd.exe

F:\Java Code>javac TestThread.java

F:\Java Code>java TestThread

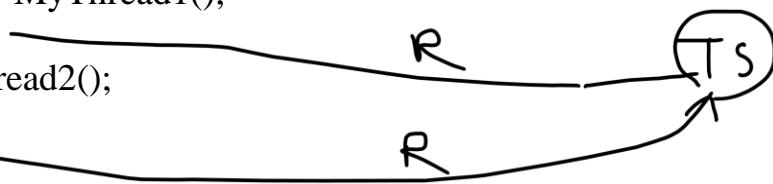
Running Thread1:1  
Running Thread1:2  
Running Thread1:3  
Running Thread1:4  
Running Thread1:5  
Running Thread1:6  
Running Thread1:7  
Running Thread1:8  
Running Thread1:9  
Running Thread1:10  
Running Thread2:11  
Running Thread2:12  
Running Thread2:13  
Running Thread2:14  
Running Thread2:15  
Running Thread2:16  
Running Thread2:17  
Running Thread2:18  
Running Thread2:19  
Running Thread2:20

#### **Example 4: Thread 1, Thread 2 using start() method.**

```
class MyThread1 extends Thread  
{  
public void run()  
{for(int i=1;i<=10;i++)  
 {System.out.println("Running Thread1:"+i);}  
}}  
class MyThread2 extends Thread  
{public void run()  
{for(int i=11;i<=20;i++)  
 {System.out.println("Running Thread2:"+i);}  
}}}
```

```
class TestThread  
{public static void main(String arg[])  
{MyThread1 mt1=new MyThread1();  
 mt1.start(); --run()  
MyThread2 mt2=new MyThread2();  
 mt2.start(); --run()  
} }
```

//will get mixed output.



C:\Windows\System32\cmd.exe

```
F:\Java Code>java TestThread
Running Thread1:1
Running Thread2:11
Running Thread1:2
Running Thread1:3
Running Thread2:12
Running Thread1:4
Running Thread2:13
Running Thread1:5
Running Thread2:14
Running Thread1:6
Running Thread1:7
Running Thread1:8
Running Thread2:15
Running Thread1:9
Running Thread2:16
Running Thread1:10
Running Thread2:17
Running Thread2:18
Running Thread2:19
Running Thread2:20
```

Note: If you run this program again, then this program may give a different output:

```
F:\Java Code 2020>java TestThread
Running Thread1:1
Running Thread1:2
Running Thread2:11
Running Thread2:12
Running Thread2:13
Running Thread2:14
Running Thread2:15
Running Thread2:16
Running Thread1:3
Running Thread1:4
Running Thread2:17
Running Thread2:18
Running Thread1:5
Running Thread2:19
Running Thread1:6
```

Output is different; there is no guarantee of getting a similar output. It varies according to the system.

### **run() vs start()**

t.run()

If we use the run() method instead of the start() method, then the run() method will be executed just like a normal method (called by the main method).

t.start()

if we use the start () method then a new thread will be created, which is responsible for the execution of the run() method.

**Note:**

Internal Definition of start()

```
start()
{
    1. Register this thread with the thread scheduler.
    2. Perform other mandatory activity.
    3. Invoke run() method.
}
```

# Multithreading in Java

**Example 1:Child Thread, main Thread using run() method (call by user).**

```
1. class MyThread extends Thread  
2. {  
3.     public void run()//overridden method  
4.     {  
5.         for(int i=1;i<=10;i++)  
6.     {  
7.         System.out.println("Child Thread is running:"+i);  
8.     }  
9. }  
10.}  
//1-10 lines defining a thread  
//Lines 4-8 is the job of the child thread
```

```
class Test  
{  
    public static void main(String args[])//main thread  
    {  
        MyThread mt=new MyThread(); //thread instance creation,main thread  
        //create a child thread object  
        mt.start(); //main thread start child thread, child thread is responsible for  
        //executing run()method line(4-8)after start method.  
        for(int j=11;j<=20;j++)  
        {  
            System.out.println("Main Thread is running:"+j);  
        }  
        //main thread is responsible for running the remaining code  
    }  
}  
// both jobs will be executed simultaneously, and we will get mixed output.
```

```
C:\Windows\System32\cmd.exe
F:\Java Code>javac Test.java

F:\Java Code>java Test
Main Thread is running:11
Child Thread is running:1
Main Thread is running:12
Child Thread is running:2
Main Thread is running:13
Main Thread is running:14
Main Thread is running:15
Child Thread is running:3
Child Thread is running:4
Main Thread is running:16
Child Thread is running:5
Child Thread is running:6
Child Thread is running:7
Child Thread is running:8
Main Thread is running:17
Child Thread is running:9
Child Thread is running:10
Main Thread is running:18
Main Thread is running:19
Main Thread is running:20
```

## **Example 2:Child Thread, main Thread using run() method.**

```
class MyThread extends Thread
{
    public void run()//overridden method
    {
        for(int i=1;i<=10;i++)
        {
            System.out.println("Child Thread is running:"+i);
        }
    }
}

class Test
{
    public static void main(String args[])//main thread
    {
        MyThread mt=new MyThread();
        mt.run();
        for(int j=11;j<=20;j++)
        {
            System.out.println("Main Thread is running:"+j);
        }
    }
}
```

C:\Windows\System32\cmd.exe

```
F:\Java Code>java Test
Child Thread is running:1
Child Thread is running:2
Child Thread is running:3
Child Thread is running:4
Child Thread is running:5
Child Thread is running:6
Child Thread is running:7
Child Thread is running:8
Child Thread is running:9
Child Thread is running:10
Main Thread is running:11
Main Thread is running:12
Main Thread is running:13
Main Thread is running:14
Main Thread is running:15
Main Thread is running:16
Main Thread is running:17
Main Thread is running:18
Main Thread is running:19
Main Thread is running:20
```

# Thread Scheduler in Java

- It is a part of JVM.
- It is responsible for scheduling the threads. If multiple threads are waiting for a chance to execute, the thread scheduler decides the order in which the threads will execute.
- We can't expect an exact algorithm followed by a thread scheduler; it varies from JVM to JVM. Hence, we can't expect thread execution order and exact output.
- Hence whenever a situation comes to multithreading, there is no guarantee for exact output, but we can provide several possible outputs.

## Overloading of run() method:

Overloading the run method is possible, but the Thread class start() method can invoke no argument run method, the other overloaded method we have to call explicitly like a normal method call. Similarly, if we overload the main() method, but JVM always calls the String args[] method.

### Example:

```
class MyThreadEx extends Thread
{
    public void run()
    {
        System.out.println("No argument run method:");
    }
    public void run(int i)
    {
        System.out.println("int argument run method:");
    }
}

class TestOverload
{
    public static void main(String arg[])
    {
        MyThreadEx t1=new MyThreadEx();

        t1.start(); // run() or run(int i)
    }
}
```

```
F:\Java Code>javac TestOverload.java
```

```
F:\Java Code>java TestOverload  
No argument run method
```

#### If we are not overriding run() method:

If you are not overriding the run method, then the Thread class run method will be executed with an empty implementation. Hence, we will not get any output.

**It's highly recommended to override the run method.** Otherwise, do not go for the multithreading concept.

#### Example:

```
class MyThreadEx extends Thread  
{  
//not override run() method.  
  
}  
class NoOverride  
{  
    public static void main(String arg[])  
    {  
        MyThreadEx t1=new MyThreadEx();  
  
        t1.start(); ---reg---invoke run()(parent)  
    }  
}
```

```
F:\Java Code>javac NoOverride.java
```

```
F:\Java Code>java NoOverride
```

```
F:\Java Code>
```

**Overriding of start() method:**

```
class MyThreadEx1 extends Thread
{
    public void start()
    {
        System.out.println("start method");
    }
    public void run()
    {
        System.out.println("run method");
    }
}

class TestStart
{
    public static void main(String args[])
    {
        MyThreadEx1 t=new MyThreadEx1();
        t.start();
        System.out.println("main method");
    }
}
```

```
F:\Java Code>java TestStart
start method
main method
```

Always get similar output if we run 1000 times.

**Reason:**

If we override the start method, our start method will be executed just like a normal method call, and a new thread will not be created. Total output will be produced by the main thread. So it is recommended **never to override the start method**; otherwise, don't go for the multithreading concept.

Let's do small change in the previous program.

```
class MyThreadEx1 extends Thread
{
public void start()
{
    super.start();
System.out.println("start method");
}
public void run()
{
System.out.println("run method");
}}
class TestStart
{
public static void main(String args[])
{
MyThreadEx1 t=new MyThreadEx1();
t.start();
System.out.println("main method");
}
}
```

We will get Mixed Output

Just because of super.start(),

```
F:\Java Code>javac TestStart.java
```

```
F:\Java Code>java TestStart
start method
run method
main method
```

### **Restart the same thread.**

After starting the thread if we are trying to restart the same thread then we will get run time exception: Illegal Thread state exception.

```
mt1.start();  
System.out.println("main thread");  
mt1.start();
```

```
java.lang.IllegalThreadStateExceptionRunning Thread1:5  
                                at java.lang.Thread.start(Thread.java:708)  
Running Thread1:6          at TestThread1.main(TestThread1.java:18)
```

## **Second Approach:**

### **By implementing the Runnable interface:**

In the first approach

MyThread-----> Thread-----→ Runnable  
Extends                   already implemented

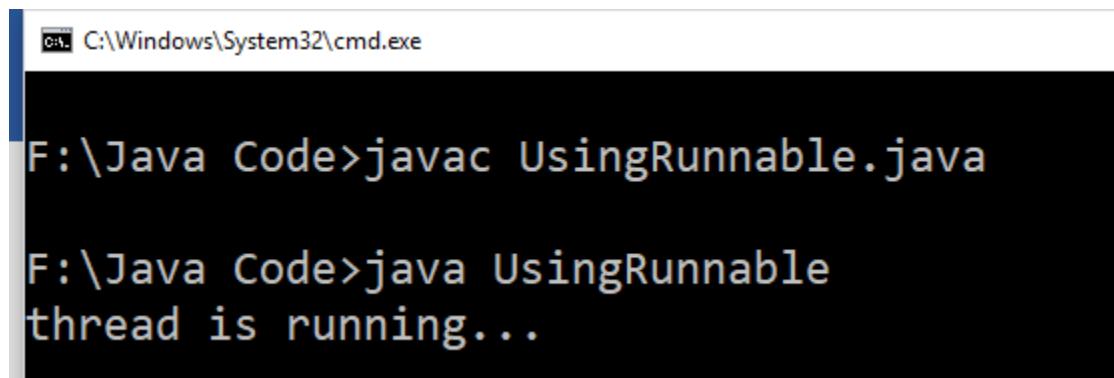
Second Approach  
MyRunnable -----→ Runnable  
Implements

We can define a thread by implementing a runnable interface. Runnable interface present in java.lang package and it contains only one method run() method.

**public void run():** is used to perform an action for a thread.

#### **How we can define a thread through Runnable**

```
class UsingRunnable implements Runnable
{
    public void run()
    {
        System.out.println("thread is running... ");
    }
    public static void main(String args[]){
        UsingRunnable r=new UsingRunnable();
        Thread t =new Thread(r);
        t.start();
    } }
```



The screenshot shows a Windows Command Prompt window titled 'C:\Windows\System32\cmd.exe'. The command 'javac UsingRunnable.java' is entered and executed, followed by the command 'java UsingRunnable', which outputs the text 'thread is running...'.

```
C:\Windows\System32\cmd.exe
F:\Java Code>javac UsingRunnable.java
F:\Java Code>java UsingRunnable
thread is running...
```

```
class UsingRunnable implements Runnable
{
    public void run()
    {
        for(int i=1;i<=10;i++)
        {
            System.out.println("child thread is running..." + i);
        }
    }
    public static void main(String args[])
    {
        UsingRunnable r=new UsingRunnable();
        Thread t =new Thread(r);
        t.start();
        for(int i=11;i<=20;i++)
        {
            System.out.println("main thread is running..." + i);
        }
    }
}
```

C:\Windows\System32\cmd.exe

```
F:\Java Code>java UsingRunnable
main thread is running...11
child thread is running...1
main thread is running...12
child thread is running...2
child thread is running...3
child thread is running...4
child thread is running...5
main thread is running...13
child thread is running...6
main thread is running...14
child thread is running...7
child thread is running...8
main thread is running...15
child thread is running...9
child thread is running...10
main thread is running...16
main thread is running...17
main thread is running...18
main thread is running...19
main thread is running...20
```

## **Case Study:**

```
MyRunnable r=new MyRunnable();
Thread t1=new Thread();
Thread t2=new Thread(r);
```

### **Case 1: t1.start()**

A new thread will be created which is responsible for the execution of the Thread class run() method. Which has an empty implementation.

### **Case 2: t1.run();**

No new thread will be created and Thread class run method will be executed just like a normal method call.

### **Case 3: t2.start();**

A new thread will be created which is responsible for the execution of MyRunnable class run method.

### **Case 4: t2.run();**

A new thread wont be created and MyRunnable run() method will be executed just like a normal method call.

### **Case 5: r.start();**

r have no start capability, we will get a compile-time error saying the MyRunnable class doesn't have start() capability.

### **Case 6: r.run();**

no new thread will be created and MyRunnable run() method will be executed like a normal method call.

## **Which approach is best to define a thread and why?**

Second (Recommended): Implement Runnable approach.

### **Reason:**

In the first approach, our class extends the Thread class there is no chance of extending any other class, hence we are missing inheritance benefits.

But in the second approach while implementing a runnable interface we can extend any other class hence we won't miss any inheritance benefits.

Because of the above reason implementing the Runnable interface approach is recommended than extending the Thread class.

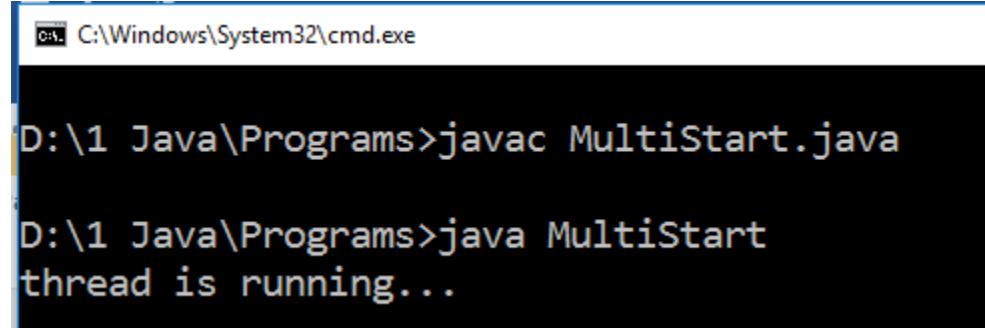
**Program:**

```
class Mythread1 implements Runnable {  
    public void run() {  
        for(int i=0;i<10;i++) {  
            System.out.println("Running Thread1:"+i); }  
    } }  
class Mythread2 extends Thread {  
    public void run() {  
        for(int i=10;i<20;i++)  
        {  
            System.out.println("Running Thread2:"+i); }  
    } }  
class Runthread {  
    public static void main(String arg[]){  
        Mythread1 r1=new Mythread1();  
        Thread t1=new Thread(r1,"thread1");  
        t1.start();  
        Mythread2 r2=new Mythread2();  
        r2.start();  
    } }
```

```
C:\Windows\System32\cmd.exe
D:\1 Java\Programs>javac Runthread.java

D:\1 Java\Programs>java Runthread
Running Thread1:0
Running Thread1:1
Running Thread2:10
Running Thread1:2
Running Thread1:3
Running Thread1:4
Running Thread2:11
Running Thread2:12
Running Thread1:5
Running Thread1:6
Running Thread1:7
Running Thread1:8
Running Thread2:13
Running Thread2:14
Running Thread1:9
Running Thread2:15
Running Thread2:16
Running Thread2:17
Running Thread2:18
Running Thread2:19
}
```

```
class MultiStart extends Thread{
    public void run()
{
    System.out.println("thread is running...");
}
public static void main(String args[]){
    MultiStart t1=new MultiStart();
    t1.start();
}
}
```



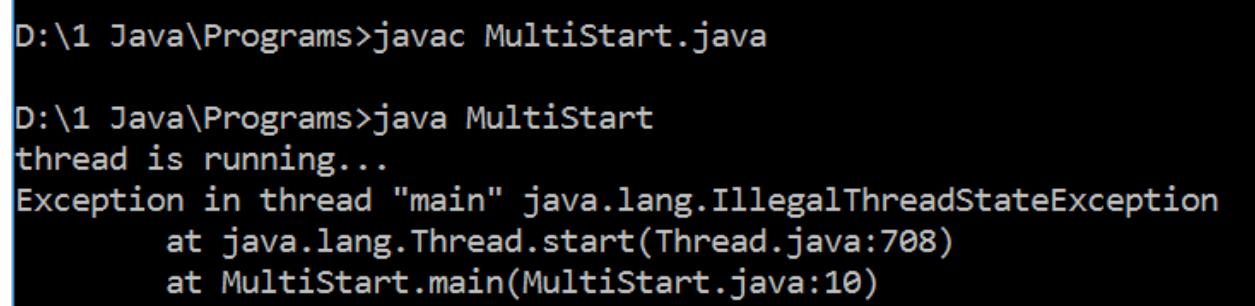
C:\Windows\System32\cmd.exe

```
D:\1 Java\Programs>javac MultiStart.java
D:\1 Java\Programs>java MultiStart
thread is running...
```

If we start a thread twice:

```
class MultiStart extends Thread{
    public void run()
{
    System.out.println("thread is running...");
}
public static void main(String args[]){
    MultiStart t1=new MultiStart();

    t1.start();
    t1.start();
}
}
```



```
D:\1 Java\Programs>javac MultiStart.java
D:\1 Java\Programs>java MultiStart
thread is running...
Exception in thread "main" java.lang.IllegalThreadStateException
        at java.lang.Thread.start(Thread.java:708)
        at MultiStart.main(MultiStart.java:10)
```

## **Important Methods of Thread Class:**

### **sleep() method:**

sleep(): This method causes the currently executing thread to sleep for the specified number of milliseconds, subject to the precision and accuracy of system timers and schedulers.

### **Syntax:**

```
public static void sleep(long milliseconds) throws InterruptedException
```

```
class Mythread1 implements Runnable {  
    public void run() {  
        for(int i=0;i<10;i++) {  
            try{  
                Thread.sleep(1000); // sleeps thread for 1 sec  
            } catch(InterruptedException e){  
                System.out.println(e);  
            }  
            System.out.println("Running Thread1:"+i); }  
        } }  
class Mythread2 extends Thread {  
    public void run() {  
        for(int i=10;i<20;i++)  
        {  
            System.out.println("Running Thread2:"+i); }  
        } }  
class Runthread {  
    public static void main(String arg[]){  
        Mythread1 r1=new Mythread1();  
        Thread t1=new Thread(r1,"thread1");  
        t1.start();  
        Mythread2 r2=new Mythread2();  
        r2.start();  
    }  
}
```

```
C:\Windows\System32\cmd.exe

D:\1 Java\Programs>javac Runthread.java

D:\1 Java\Programs>java Runthread
Running Thread2:10
Running Thread2:11
Running Thread2:12
Running Thread2:13
Running Thread2:14
Running Thread2:15
Running Thread2:16
Running Thread2:17
Running Thread2:18
Running Thread2:19
Running Thread1:0
Running Thread1:1
Running Thread1:2
Running Thread1:3
Running Thread1:4
Running Thread1:5
Running Thread1:6
Running Thread1:7
Running Thread1:8
Running Thread1:9
```

The output shows that thread 2 is executed first, after that thread 1 is executed.

**Did we achieve multithreading through this program—YES/NO**

answer: yes

As soon as the execution of thread 1 is started, the compiler found that it has a sleep() method that tells t1 to stop its task for 1 second, as JVM has stopped the task of t1 for 1 second, The other thread 2 gets a chance to execute and t2 finishes its job within 1 second.

Try this program using increase the load of both the threads and decrease the sleep time of thread1, you will get mixed output means multithreading.

**The following program shows that both threads have a sleep method to stop their task for a few seconds.**

```
class Mythread1 implements Runnable {  
    public void run() {  
        for(int i=0;i<10;i++) {  
            try{  
                Thread.sleep(1000); // sleeps thread for 1 sec  
            } catch(InterruptedException e){  
                System.out.println(e);  
            }  
            System.out.println("Running Thread1:"+i); }  
        } }  
class Mythread2 extends Thread {  
    public void run() {  
        for(int i=10;i<20;i++)  
        {  
            try{  
                Thread.sleep(1000); // sleeps thread for 1 sec  
            } catch(InterruptedException e){  
                System.out.println(e);  
            }  
            System.out.println("Running Thread2:"+i); }  
        } }  
class Runthread {  
    public static void main(String arg[]){  
        Mythread1 r1=new Mythread1();  
        Thread t1=new Thread(r1,"thread1");  
        t1.start();  
        Mythread2 r2=new Mythread2();  
        r2.start();  
    }  
}
```

C:\Windows\System32\cmd.exe

D:\1 Java\Programs>javac Runthread.java

D:\1 Java\Programs>java Runthread

Running Thread1:0

Running Thread2:10

Running Thread1:1

Running Thread2:11

Running Thread1:2

Running Thread2:12

Running Thread1:3

Running Thread2:13

Running Thread1:4

Running Thread2:14

Running Thread2:15

Running Thread1:5

Running Thread2:16

Running Thread1:6

Running Thread1:7

Running Thread2:17

Running Thread2:18

Running Thread1:8

Running Thread1:9

Running Thread2:19

**The following program shows that thread 1 is stopping its work for 2 seconds and thread 1 is stopping its work for 1 second, which means that thread 2 is getting more execution time than thread 1.**

```
class Mythread1 implements Runnable {  
    public void run() {  
        for(int i=0;i<10;i++) {  
            try{  
                Thread.sleep(2000); // sleeps thread for 2 sec  
            } catch(InterruptedException e){  
                System.out.println(e);  
            }  
            System.out.println("Running Thread1:"+i);  
        }  
    }  
}  
class Mythread2 extends Thread {  
    public void run() {  
        for(int i=10;i<20;i++)  
        {  
            try{  
                Thread.sleep(1000); // sleeps thread for 1 sec  
            } catch(InterruptedException e){  
                System.out.println(e);  
            }  
            System.out.println("Running Thread2:"+i);  
        }  
    }  
}  
class Runthread1 {  
    public static void main(String arg[]){  
        Mythread1 r1=new Mythread1();  
        Thread t1=new Thread(r1,"thread1");  
        t1.start();  
        Mythread2 r2=new Mythread2();  
        r2.start();  
    }  
}
```

cmd C:\Windows\System32\cmd.exe

```
F:\Java Code>java Runthread1
Running Thread2:10
Running Thread1:0
Running Thread2:11
Running Thread2:12
Running Thread1:1
Running Thread2:13
Running Thread2:14
Running Thread1:2
Running Thread2:15
Running Thread2:16
Running Thread1:3
Running Thread2:17
Running Thread2:18
Running Thread1:4
Running Thread2:19
Running Thread1:5
Running Thread1:6
Running Thread1:7
Running Thread1:8
Running Thread1:9
```

**This program shows the multithreading among three threads.**

```
class Mythread1 implements Runnable {  
    public void run() {  
        for(int i=0;i<10;i++) {  
            System.out.println("Running Thread1:"+i); }  
    } }  
  
class Mythread2 extends Thread {  
    public void run() {  
        for(int i=10;i<20;i++)  
        {  
            System.out.println("Running Thread2:"+i); }  
    } }  
  
class Mythread3 extends Thread {  
    public void run() {  
        for(int i=20;i<30;i++)  
        {  
            System.out.println("Running Thread3:"+i); }  
    } }  
  
class Jointhread {  
    public static void main(String arg[]){  
        Mythread1 r1=new Mythread1();  
        Thread t1=new Thread(r1,"thread1");  
        t1.start();  
        Mythread2 r2=new Mythread2();  
        r2.start();  
        Mythread3 r3=new Mythread3();  
        r3.start();  
    }  
}
```

C:\Windows\System32\cmd.exe

D:\1 Java\Programs>javac Jointhread.java

D:\1 Java\Programs>java Jointhread

Running Thread1:0

Running Thread1:1

Running Thread2:10

Running Thread3:20

Running Thread1:2

Running Thread3:21

Running Thread2:11

Running Thread3:22

Running Thread1:3

Running Thread3:23

Running Thread2:12

Running Thread3:24

Running Thread1:4

Running Thread3:25

Running Thread2:13

Running Thread3:26

Running Thread1:5

Running Thread3:27

Running Thread2:14

Running Thread3:28

Running Thread1:6

Running Thread3:29

Running Thread2:15

Running Thread1:7

## **join() Method**

When the join() method is invoked, the current thread stops its execution and the thread goes into the wait state. The current thread remains in the wait state until the thread on which the join() method is invoked has achieved its dead state. If an interruption of the thread occurs, then it throws the InterruptedException.

### **Syntax:**

- public void join()throws InterruptedException
- public void join(long milliseconds) throws InterruptedException

**//This program shows the working of join() method.**

```
class Mythread1 implements Runnable {
    public void run() {
        for(int i=1;i<11;i++) {
            System.out.println("Running Thread1:"+i);
        }
    }
}

class Mythread2 extends Thread {
    public void run() {
        for(int i=11;i<21;i++)
        {
            System.out.println("Running Thread2:"+i);
        }
    }
}

class Mythread3 extends Thread {
    public void run() {
        for(int i=21;i<31;i++)
        {
            System.out.println("Running Thread3:"+i);
        }
    }
}

class JoinThread {
    public static void main(String arg[]){
        Mythread1 r1=new Mythread1();
        Thread t1=new Thread(r1,"thread1");
        t1.start();
        try{
            t1.join(); //it force to complete the task of t1 thread.
        }
        catch(Exception e)
        {
            System.out.println(e);
        }
    }
}

Mythread2 t2=new Mythread2();
t2.start();
Mythread3 t3=new Mythread3();
t3.start();
```

```
}
```

```
C:\Windows\System32\cmd.exe
F:\Java Code>java JoinThread
Running Thread1:1
Running Thread1:2
Running Thread1:3
Running Thread1:4
Running Thread1:5
Running Thread1:6
Running Thread1:7
Running Thread1:8
Running Thread1:9
Running Thread1:10
Running Thread2:11
Running Thread2:12
Running Thread3:21
Running Thread3:22
Running Thread3:23
Running Thread2:13
Running Thread3:24
Running Thread2:14
Running Thread2:15
Running Thread2:16
Running Thread2:17
Running Thread2:18
Running Thread2:19
```

The output shows that as soon as thread 1 gets the processor to perform its task, it forces the JVM to complete its work (because of the join() method), and after the full execution of the t1 thread, it allows the processor to perform the task of the other thread.

```

class Mythread1 implements Runnable {
    public void run() {
        for(int i=1;i<11;i++) {
            System.out.println("Running Thread1:"+i); }
    } }
class Mythread2 extends Thread {
    public void run() {
        for(int i=11;i<21;i++)
        {
            System.out.println("Running Thread2:"+i); }
    } }
class Mythread3 extends Thread {
    public void run() {
        for(int i=21;i<31;i++)
        {
            System.out.println("Running Thread3:"+i); }
    } }

class JoinThread {
    public static void main(String arg[]){
        Mythread1 r1=new Mythread1();
        Thread t1=new Thread(r1,"thread1");
        t1.start();
        try{
            t1.join(); //it force to complete the task of t1 thread.
        }
        catch(Exception e)
        {
            System.out.println(e);
        }
    }
}

Mythread2 t2=new Mythread2();
t2.start();
try{
    t2.join(); //it force to complete the task of t2 thread.
}
catch(Exception e)
{
    System.out.println(e);
}
Mythread3 t3=new Mythread3();
t3.start();
}

```

```
C:\Windows\System32\cmd.exe
F:\Java Code>java JoinThread
Running Thread1:1
Running Thread1:2
Running Thread1:3
Running Thread1:4
Running Thread1:5
Running Thread1:6
Running Thread1:7
Running Thread1:8
Running Thread1:9
Running Thread1:10
Running Thread2:11
Running Thread2:12
Running Thread2:13
Running Thread2:14
Running Thread2:15
Running Thread2:16
Running Thread2:17
Running Thread2:18
Running Thread2:19
Running Thread2:20
Running Thread3:21
Running Thread3:22
Running Thread3:23
```

**Did we achieve multithreading through this program? yes/no**  
yes

The output shows that as soon as thread 1 gets the processor to perform its task, it forces the JVM to complete its task (because the join () method) and after the full execution of the t1 thread it

releases the processor. After that, thread 2 gets a chance, it finishes due to the join method, and finally, thread 3 executes its task.

**This program shows how the join(long milliseconds) method works. In this example, when t1 completes its task for 10 milliseconds, then r2 and r3 start executing.**

```
class Mythread1 implements Runnable {  
    public void run() {  
        for(int i=0;i<10;i++) {  
  
            System.out.println("Running Thread1:"+i); }  
    } }  
class Mythread2 extends Thread {  
    public void run() {  
        for(int i=10;i<20;i++) {  
  
            System.out.println("Running Thread2:"+i); }  
    } }  
class Mythread3 extends Thread {  
    public void run() {  
        for(int i=20;i<30;i++) {  
  
            System.out.println("Running Thread3:"+i); }  
    } }  
class Jointhread {  
    public static void main(String arg[]){  
        Mythread1 r1=new Mythread1();  
        Thread t1=new Thread(r1,"thread1");  
        t1.start();  
        try{  
            t1.join(10); //it force to complete the task of t1 thread for 10 milli sec.  
        }  
        catch(Exception e)  
        {  
            System.out.println(e);  
        }  
        Mythread2 r2=new Mythread2();  
        r2.start();  
        Mythread3 r3=new Mythread3();  
        r3.start();  
    } }
```

```
C:\Windows\System32\cmd.exe
D:\1 Java\Programs>javac Jointhread.java
D:\1 Java\Programs>java Jointhread
Running Thread1:0
Running Thread1:1
Running Thread1:2
Running Thread1:3
Running Thread2:10
Running Thread1:4
Running Thread3:20
Running Thread3:21
Running Thread2:11
Running Thread3:22
Running Thread1:5
Running Thread3:23
Running Thread2:12
Running Thread2:13
Running Thread2:14
Running Thread3:24
Running Thread3:25
Running Thread1:6
Running Thread3:26
Running Thread2:15
Running Thread3:27
Running Thread1:7
Running Thread3:28
Running Thread3:29
```

**getName(), setName(String) and getId() ,currentThread() methods:**

```
class Mythread1 implements Runnable {  
    public void run() {  
        System.out.println("Thread 1 is running");  
        System.out.println("Current Thread is:"+Thread.currentThread().getName());  
    } }  
  
class Mythread2 extends Thread {  
    public void run() {  
        System.out.println("Thread 2 is running");  
        System.out.println("Current Thread is:"+Thread.currentThread().getName());  
    } }  
  
class Runthread3 {  
    public static void main(String arg[]){  
        Mythread1 r1=new Mythread1();  
        Thread t1=new Thread(r1,"thread1");  
        t1.start();  
        Mythread2 r2=new Mythread2();  
        Thread t2=new Thread(r2,"thread2");  
        t2.start();  
  
        System.out.println("Name of t1:"+t1.getName());  
        System.out.println("Name of t2:"+t2.getName());  
        System.out.println("id of t1:"+t1.getId());  
        System.out.println("id of t2:"+t2.getId());  
        t1.setName("IFM");  
        System.out.println("After changing name of t1:"+t1.getName());  
        System.out.println("Current Thread is:"+Thread.currentThread().getName());  
    } }
```

```
D:\1 Java\Programs>java Runthread
Thread 1 is running
Name of t1:thread1
Thread 2 is running
Current Thread is:thread1
Current Thread is:thread2
Name of t2:thread2
id of t1:10
id of t2:12
After changing name of t1:IFM
Current Thread is:main
```

## **Thread Priorities:**

Every Java Thread has a priority that helps the Operating System determine the order in which threads are scheduled.

An Operating system thread scheduler determines which thread runs next.

Thread scheduling is platform-dependent; the behavior of a multithread program could vary across different platforms.

Threads can be assigned priorities. The JVM/OS uses the priorities to determine which runnable thread will be given the CPU time.

Priorities are given by priority numbers ranging from 1 to 10.

| <b>Predefined priority</b> | <b>value</b> |
|----------------------------|--------------|
| MIN_PRIORITY               | 1            |
| NORM_PRIORITY              | 5            |
| MAX_PRIORITY               | 10           |

Thread Scheduler will use priorities while allocating processors. The thread with the highest priority will get the chance first.

### **Which thread will get a chance first if two threads have equal priority?**

We can't expect an exact execution order; it depends on the Thread Scheduler.

Thread class defines the following methods to get and set the priority of your thread.

**public final int getPriority();**  
**public final void setPriority(int p);**

Range of p:1-10 other than 1-10, it will give IllegalArgumentException.

The default priority only for the main thread is 5, but for all remaining threads, the default priority will be inherited from parent to child i.e., Whatever priority the parent thread has the same priority will be there for the child thread.

```
class MyThread extends Thread
{
}
class Test1{
public static void main(String args[])
{
System.out.println(Thread.currentThread().getPriority());//5
//Thread.currentThread().setPriority(15);//IllegalArgumentException
Thread.currentThread().setPriority(7);
MyThread t=new MyThread();//main thread create a child thread. who execute this line main()
System.out.println(t.getPriority());//7
}}
```

```
F:\Java Code>javac Test1.java
```

```
F:\Java Code>java Test1
```

```
5  
7
```

The priority of the thread can be set using the method:

- `setPriority(int pnum);`

To obtain priority of a thread:

- `Int getPriority();`

**Example:**

```
MyThread m = new MyThread()// has a priority of 5.
```

```
MyThread m1 = new MyThread(8)// has a priority of 8.
```

The thread with highest priority does not claim all the CPU time; the nature of thread behavior will vary greatly from system to system.

**Program**

**//This program shows the use of `setPriority()` and `getPriority()` method.**

```
class MyThread1 extends Thread  {  
    public void run()  {  
        System.out.println("Thread is running");  
    } }  
class SetThreadPriority  {  
    public static void main(String args[])  {  
        MyThread1 t1=new MyThread1();//by default priority is 5  
        MyThread1 t2=new MyThread1();//by default priority is 5  
        MyThread1 t3=new MyThread1();//by default priority is 5  
        t1.setPriority(Thread.MIN_PRIORITY);//1  
        t2.setPriority(Thread.MAX_PRIORITY);//10  
        //t1.setPriority(1);  
        //t2.setPriority(10);  
        System.out.println("Priority Value of t1 thread="+t1.getPriority());  
        System.out.println("Priority Value of t2 thread="+t2.getPriority());  
        System.out.println("Priority Value of t3 thread="+t3.getPriority());  
        t1.start();  
        t2.start();  
        t3.start();  
    } }
```

```
D:\1 Java\Programs>javac SetThreadPriority.java
```

```
D:\1 Java\Programs>java SetThreadPriority
Priority Value of t1 thread=1
Priority Value of t2 thread=10
Priority Value of t3 thread=5
Thread is running
Thread is running
Thread is running
```

```
// This program shows multithreading between two threads with default priority.
class MyThread1 extends Thread {
    public void run() {
        for(int i=1;i<=10;i++){
            System.out.println("Running Thread 1:"+i);
        }
    }
}
class MyThread2 extends Thread {
    public void run() {
        for(int i=11;i<=20;i++){
            System.out.println("Running Thread 2:"+i);
        }
    }
}

class ThreadPriority1 {
    public static void main(String args[]) {
        MyThread1 t1=new MyThread1();
        MyThread2 t2=new MyThread2();
        System.out.println("Thread created is: "+t1.getName());
        System.out.println("Thread created is: "+t2.getName());
        System.out.println("Priority Value of t1 thread="+t1.getPriority());
        System.out.println("Priority Value of t2 thread="+t2.getPriority());
        t1.start();
        t2.start();
    }
}
```

C:\Windows\System32\cmd.exe

```
D:\1 Java\Programs>java ThreadPriority1
Thread created is: Thread-0
Thread created is: Thread-1
Priority Value of t1 thread=5
Priority Value of t2 thread=5
Running Thread 2:11
Running Thread 1:1
Running Thread 2:12
Running Thread 1:2
Running Thread 1:3
Running Thread 2:13
Running Thread 1:4
Running Thread 2:14
Running Thread 1:5
Running Thread 2:15
Running Thread 1:6
Running Thread 2:16
Running Thread 1:7
Running Thread 2:17
Running Thread 1:8
Running Thread 2:18
Running Thread 1:9
Running Thread 2:19
Running Thread 1:10
Running Thread 2:20
```

```
// This program shows multithreading between two threads with different priorities.

class MyThread1 extends Thread {
    public void run() {
        for(int i=1;i<=10;i++){
            System.out.println("Running Thread 1:"+i);
    } }
    class MyThread2 extends Thread {
        public void run() {
            for(int i=11;i<=20;i++){
                System.out.println("Running Thread 2:"+i);
    } }

class ThreadPriority2 {
    public static void main(String args[]) {
        MyThread1 t1=new MyThread1();
        MyThread2 t2=new MyThread2();

t1.setPriority(1); //t1.Thread.MIN_PRIORITY;
t2.setPriority(10); //t2.setPriority(Thread.MAX_PRIORITY);

        System.out.println("Thread created is: "+t1.getName());
        System.out.println("Thread created is: "+t2.getName());
        System.out.println("Priority Value of t1 thread="+t1.getPriority());
        System.out.println("Priority Value of t2 thread="+t2.getPriority());
        t1.start();
        t2.start();
    } }
```

```
D:\1 Java\Programs>java ThreadPriority2
Thread created is: Thread-0
Thread created is: Thread-1
Priority Value of t1 thread=1
Priority Value of t2 thread=10
Running Thread 2:11
Running Thread 2:12
Running Thread 2:13
Running Thread 2:14
Running Thread 2:15
Running Thread 2:16
Running Thread 2:17
Running Thread 2:18
Running Thread 2:19
Running Thread 2:20
Running Thread 1:1
Running Thread 1:2
Running Thread 1:3
Running Thread 1:4
Running Thread 1:5
Running Thread 1:6
Running Thread 1:7
Running Thread 1:8
Running Thread 1:9
Running Thread 1:10
```

### Note:

- Thread priorities influence execution order but do not guarantee it (depends on OS and JVM).
- Higher-priority jobs (like urgent prints) are executed earlier.
- If priorities are equal, jobs are executed in a FIFO or round-robin manner.

## Daemon Thread in Java

- **Daemon thread in java** is a service provider thread that provides services to user threads for background supporting tasks. It has no role in life; it provides services to the user thread.
- Its life depends on the mercy of user threads, i.e., when all the user threads die, JVM terminates this thread automatically.
- It is a low-priority thread.
- There are many Java daemon threads running automatically, e.g., gc, finalizer, etc.
- Type jconsole on command prompt: It will show information about the loaded classes, memory usage, running threads etc.

### Methods:

1. **public void setDaemon(boolean status)** : marks the current thread as a daemon or user thread.
2. **public boolean isDaemon()** : is used to check that current is daemon.

```
class MyThread1 extends Thread {  
    public void run() {  
        if(Thread.currentThread().isDaemon()){//checking for daemon thread  
            System.out.println("daemon thread work");  
        }  
        else{  
            System.out.println("user thread1 work");  
        }  
    } }  
  
class MyThread2 extends Thread {  
    public void run() {  
        System.out.println("user thread2 work");  
    } }  
  
class TestDaemonThread {  
    public static void main(String args[]) {  
        MyThread1 t1=new MyThread1();  
        MyThread2 t2=new MyThread2();  
        t1.setDaemon(true);//now t1 is daemon thread  
        t1.start();  
        t2.start();  
    } }
```

```
D:\1 Java\Programs>javac TestDaemonThread.java
```

```
D:\1 Java\Programs>java TestDaemonThread
```

```
user thread2 work
```

```
daemon thread work
```

```
D:\1 Java\Programs>javac TestDaemonThread.java
```

```
D:\1 Java\Programs>java TestDaemonThread
```

```
daemon thread work
```

```
user thread2 work
```

## **Exploring Java I/O, Directories**

- The data is lost when a variable goes out of scope, or the program is terminated; this storage is temporary.
- It is difficult to handle large volumes of data using variables and arrays.

We can overcome these problems by storing data on secondary devices such as floppy or hard disks. The data is stored in these devices using the concept of files.

### **Concept of File:**

A file is a data structure where a considerable amount of data can be stored. Typically, any programming language supports two types of files-

#### **Text Files**

- Definition: Data is stored in a human-readable format (plain text).
- **Characteristics:**
  - Easily readable and editable using any text editor.
  - Slightly slower to process compared to binary files.
  - Ideal for source code, configuration files, logs, etc.
- **Examples:**
  - .txt (Text Document)
  - .java (Java source file)
  - .csv (Comma Separated Values)
  - .xml, .html, .json (structured text formats)

#### **Binary Files**

- Definition: Data is stored in a machine-readable binary format (not human-readable).
- **Characteristics:**
  - Faster and more efficient to process.
  - Not readable using a plain text editor without decoding.
  - Suitable for compiled code, images, audio, videos, etc.
- **Examples:**
  - .class (Compiled Java class files)

- .exe (Executable file)
- .jpg, .png (Images)
- .mp3, .mp4 (Audio/Video)
- .dat (Raw binary data)

## File Handling in Java

File handling is a core concept in Java that allows programs to interact with files stored on a computer. Java provides the `File` class, part of the `java.io` package, to perform various file-related operations such as creating, reading, writing, and deleting files.

### Why is File Handling Important?

File handling is essential in programming because it allows you to:

- **Store data permanently:** Unlike variables that lose their data when a program ends, files retain data even after the program stops.
- **Read and write data:** You can save program outputs, configuration settings, logs, or user data to files.
- **Process large data sets:** Files are useful when working with extensive data that cannot be handled entirely in memory.

### In Simple Terms

File handling means:

- Reading data from a file
- Writing data to a file

### What is the File Class in Java?

- The `File` class in Java represents a file or directory in the file system.
- The `File` class is used to access file information (such as name, path, size, and permissions) but cannot directly read or write the file content.
- It works with the attributes or properties of a file, not with its actual data.
- It helps check file existence, type (file or directory), and other metadata before performing file operations.
- The `File` class belongs to the `java.io` package.

- While Java is platform-independent, file naming conventions depend on the underlying Operating System.
- In Java, even directories are treated as File objects, often called "directory files".
- To work with a file, you first create an object of the File class and pass the name or path of the file as a parameter. This object refers to the file or directory you want to work with.

### **Constructors of the File Class**

- `File(String dirPath);` // Creates a file object with full path
- `File(String dirPath, String fileName);` // Creates a file object by combining directory path and file name
- `File(File parent, String fileName);` // Uses another File object as the directory

### **Parameter Meanings:**

- `dirPath` → The directory path
- `fileName` → The name of the file
- `parent` → A File object referring to the parent directory

### **Examples:**

```
File f1 = new File("D:\\Java Code\\IO\\First.java");
File f2 = new File("D:\\Java Code\\IO", "First.java");
File f3 = new File(f2, "First.java");
```

**Note:** Use double backslashes (\\) in Windows paths inside Java strings.

### **This works (no escape sequences involved):**

```
File f1 = new File("D:\\JavaCode\\IO\\First.java");
```

### **This will cause an error or unexpected behavior:**

```
File f1 = new File("D:\\JavaCode\\IO\\notes.java"); // '\\n' is a newline!
```

## File class methods

- 1.String getName():** Return the name of the directory or file of the invoking object.
- 2.String getPath():** Return the path of the invoking object.
- 3.String getAbsolutePath():** Return the absolute path of the invoking object.
- 4.String getParent():** return the name of the parent directory of the invoking object
- 5.boolean exists():** Return true if the dir/file object exists; otherwise, it is false.
- 6.boolean canWrite():** Return true if the file object is writable, otherwise false.
- 7.boolean canRead():** Return true if the file object is readable otherwise, false.
- 8.boolean isDirectory():**Return true if invoking object is a directory,otherwise false
- 9.boolean isFile():** Return true if invoking object is a file ,otherwise false
- 10.boolean isHidden():**Return true if invoking file is a object, otherwise false
- 11.long length():** Return a long value representing the size of the file in bytes.
- 12.long lastModified():** return a long value representing the last modified date in milliseconds.
- 13.boolean renameTo():** Renames the file object with the new name; return true upon successful renaming otherwise false.
- 14.boolean delete():** deletes the file of the invoking object; returns true on successful deletion, otherwise false.
- 15.boolean setReadOnly():** Set the file of the invoking object to read only mode; return true on successful setting otherwise false.
- 16. boolean deleteOnExit():** Deletes the file of invoking object upon the termination of Java virtual machine.
- 17.String[]list :** This methods returns all the content of directory file.

## EXAMPLES

Using functions exists(),getName(),getParent(),length() & lastModified().

### Program Code:

```
TestFileProp.java
1 import java.io.*;
2 import java.util.*;
3 public class TestFileProp
4 {
5     public static void main(String args[])
6     {
7         File fobj = new File("saurabh.txt");
8         if (fobj.exists())
9         {
10             System.out.println("Name of file is "+fobj.getName());
11             System.out.println("Parent directory is "+fobj.getParent());
12             System.out.println("File is "+fobj.length()+" bytes long");
13             long time = fobj.lastModified();
14             Date date = new Date(time);
15             System.out.println("File was last modified "+date);
16         }
17         else
18         {
19             System.out.println("File does not exist");    } }
```

Output:

```
D:\Java Code\IO>javac TestFileProp.java

D:\Java Code\IO>java TestFileProp
Name of file is saurabh.txt
Parent directory is null
File is 27 bytes long
File was last modified Thu Mar 27 09:52:11 IST 2025
```

## 2.Using canRead(), canWrite(), exists(), isAbsolute(), isFile() & isHidden().

### Program Code

```
TestFileChar.java
1 import java.io.*;
2 import java.util.*;
3 public class TestFileChar {
4     public static void main(String args[]) {
5         File fobj = new File("saurabh.txt");
6         if (fobj.canRead()) {
7             System.out.println(fobj + " can be read");
8         }
9         if (fobj.canWrite()) {
10            System.out.println(fobj + " can be write");
11        }
12        if (fobj.exists()) {
13            System.out.println(fobj + " exists");
14        }
15        if (fobj.isAbsolute()) {
16            System.out.println(fobj + " represents an absolute path");
17        }
18        if (fobj.isFile()) {
19            System.out.println(fobj + " is a file");
20        }
21        if (fobj.isHidden()) {
22            System.out.println(fobj + " is a hidden file");
23        }
24        System.out.println("Length is " + fobj.length());
25        System.out.println("Last Modified Date is " + new Date(fobj.lastModified()));
26    }
}
```

### Output:

```
D:\Java Code\IO>javac TestFileChar.java

D:\Java Code\IO>java TestFileChar
saurabh.txt can be read
saurabh.txt can be write
saurabh.txt exists
saurabh.txt is a file
Length is 27
Last Modified Date is Thu Mar 27 09:52:11 IST 2025
```

### 3.getAbsolutepath() absolute path.

#### Program Code

```
TestAbs.java |  
1 import java.io.*;  
2  
3 public class TestAbs {  
4     public static void main(String args[]) {  
5         File f = new File("java.txt");  
6  
7         System.out.println("Path of " + f.getName() + " is: " + f.getPath());  
8         System.out.println("Parent of " + f.getName() + " is: " + f.getParent());  
9         System.out.println("Absolute Path is: " + f.getAbsolutePath());  
10    }  
11 }
```

Output:

```
D:\Java Code\IO>javac TestAbs.java  
  
D:\Java Code\IO>java TestAbs  
Path of java.txt is: java.txt  
Parent of java.txt is: null  
Absolute Path is: D:\Java Code\IO\java.txt
```

### 4. mkdir() & isDirectory()

```
TestFileDir.java |  
1 import java.io.*;  
2  
3 public class TestFileDir {  
4     public static void main(String args[]) {  
5         File fobj = new File("OOP"); // Creating a File object for the directory  
6  
7         // Attempt to create the directory  
8         if (fobj.mkdir()) {  
9             System.out.println("Directory " + fobj.getName() + " created successfully.");  
10        } else {  
11            System.out.println("Directory already exists or could not be created.");  
12        }  
13  
14         // Checking if it is a valid directory  
15         if (fobj.isDirectory()) {  
16             System.out.println(fobj.getName() + " is a directory");  
17         }  
18     }  
19 }
```

Output:

```
C:\WINDOWS\system32\cmd. X + ^

D:\Java Code\IO>javac TestFileDir.java

D:\Java Code\IO>java TestFileDir
Directory OOP created successfully.
OOP is a directory

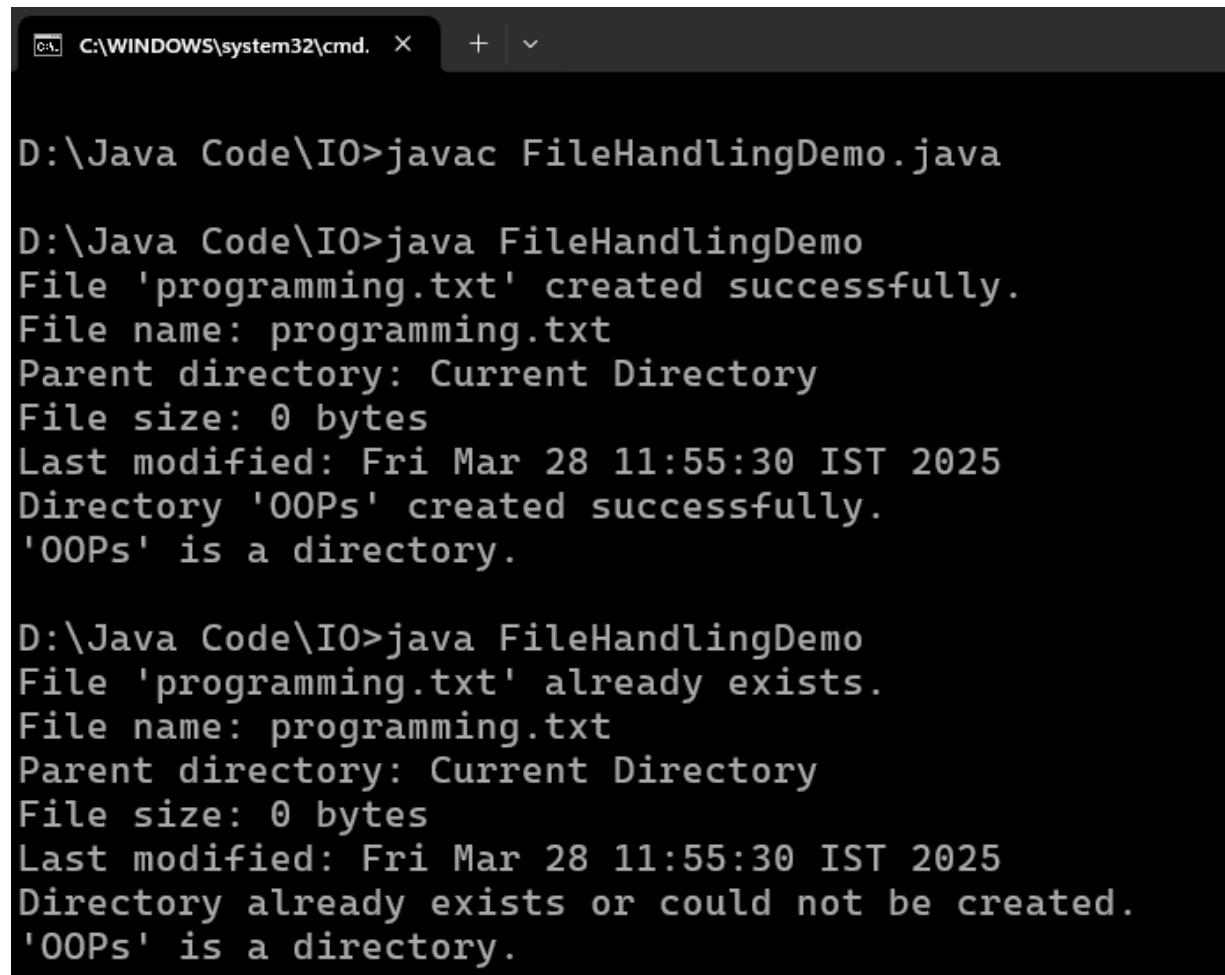
D:\Java Code\IO>java TestFileDir
Directory already exists or could not be created.
OOP is a directory
```

## All Methods Example

```
FileHandlingDemo.java x
1 import java.io.*;
2 import java.util.*;
3
4 public class FileHandlingDemo {
5     public static void main(String args[]) {
6         // File handling: Creating and checking file properties
7         File fileObj = new File("programming.txt");
8
9         try {
10             if (fileObj.exists()) {
11                 System.out.println("File '" + fileObj.getName() + "' already exists.");
12             } else {
13                 if (fileObj.createNewFile()) {
14                     System.out.println("File '" + fileObj.getName() + "' created successfully.");
15                 } else {
16                     System.out.println("File creation failed.");
17                 }
18             }
19         } catch (IOException e) {
20             System.out.println("An error occurred while handling the file: " + e.getMessage());
21         }
}
```

```
22 // Checking file properties if it exists
23 if (fileObj.exists()) {
24     System.out.println("File name: " + fileObj.getName());
25     System.out.print("Parent directory: " );
26     System.out.println((fileObj.getParent() != null ? fileObj.getParent() : "Current Directory"));
27     System.out.println("File size: " + fileObj.length() + " bytes");
28     System.out.println("Last modified: " + new Date(fileObj.lastModified()));
29 } else {
30     System.out.println("File does not exist.");
31 }
32
33 // Directory handling: Creating and checking directories
34 File dirObj = new File("OOPS");
35
36 if (dirObj.mkdir()) {
37     System.out.println("Directory '" + dirObj.getName() + "' created successfully.");
38 } else {
39     System.out.println("Directory already exists or could not be created.");
40 }
41
42 // Checking if the directory exists
43 if (dirObj.isDirectory()) {
44     System.out.println("'" + dirObj.getName() + "' is a directory.");
45 }
46
47 }
```

Output:



D:\Java Code\IO>javac FileHandlingDemo.java

D:\Java Code\IO>java FileHandlingDemo

```
File 'programming.txt' created successfully.
File name: programming.txt
Parent directory: Current Directory
File size: 0 bytes
Last modified: Fri Mar 28 11:55:30 IST 2025
Directory 'OOPs' created successfully.
'OOPs' is a directory.
```

D:\Java Code\IO>java FileHandlingDemo

```
File 'programming.txt' already exists.
File name: programming.txt
Parent directory: Current Directory
File size: 0 bytes
Last modified: Fri Mar 28 11:55:30 IST 2025
Directory already exists or could not be created.
'OOPs' is a directory.
```

## Streams in Java

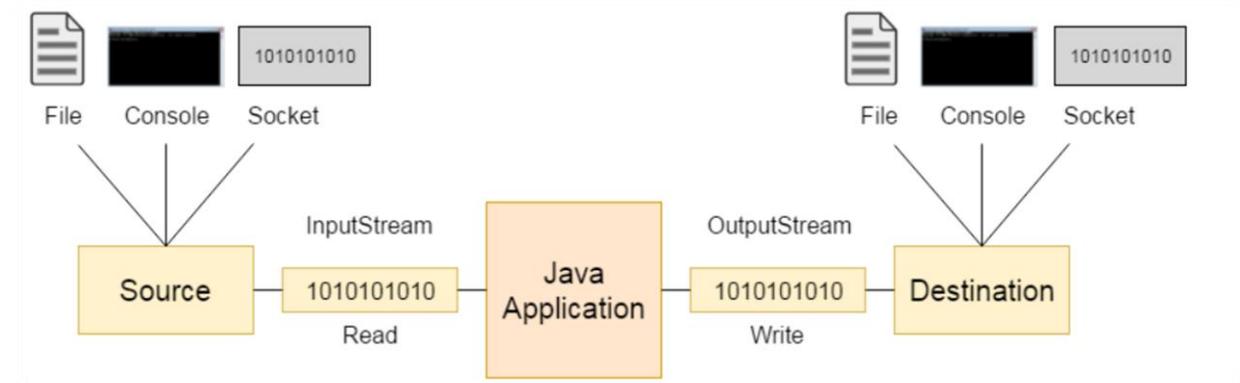
- A stream is a sequence of data. In Java, a stream is composed of bytes. It's called a stream because it is like a stream of water that continues to flow.
- It is used for input and output (I/O) operations such as reading data from a file or writing data to a file, network socket, or console.
- Java uses the concept of a stream to make I/O operations fast. The `java.io` package contains all the classes required for input and output operations.

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

1. **System.in:** Standard input stream that reads characters from the keyboard or any other standard input device.
2. **System.out:** A standard output stream produces the result of a program on an output device like a computer screen.
3. **System.err:** A standard error stream outputs all the error data a program might throw on a computer screen or any standard output device.

Depending on the type of operations, streams can be divided into two primary classes:

- `InputStream`
- `OutputStream`



In Java, there are two main types of streams based on the type of data they handle:

| Type              | Handles                                     | Examples                  |
|-------------------|---|---------------------------|
| Byte Streams      | Raw bytes (binary data like images, videos) | InputStream, OutputStream |
| Character Streams | Characters (text data like .txt, .csv)      | FileReader, FileWriter    |

**InputStream:** Java application uses an **input stream** to read data from a source; it may be a file, an array, a peripheral device, or a socket.

- An InputStream reads data from a source (such as a file, keyboard, or network).
- It is an abstract class in the java.io package.
- As an abstract class, InputStream cannot be used directly. Instead, we use its subclasses, which provide concrete implementations.
- Common Subclasses of `InputStream`:

| Subclass                             | Description   |
|--------------------------------------|---|
| <code>FileInputStream</code>         | Reads data from files   |
| <code>ByteArrayInputStream</code>    | Reads data from byte arrays   |
| <code>AudioInputStream</code>        | Reads audio data  |
| <code>FilterInputStream</code>       | Provides filtered input streams                                       |
| <code>StringBufferInputStream</code> | Deprecated – used to read characters from a <code>StringBuffer</code> |
| <code>ObjectInputStream</code>       | Reads objects previously written by <code>ObjectOutputStream</code>   |

Example: Creating an InputStream

```
InputStream input = new FileInputStream("example.txt");
```

### Common Methods of InputStream

| Method                               | Description  |
|--------------------------------------|--|
| <code>int read()</code>              | Reads a single byte of data  |
| <code>int read(byte[] array)</code>  | Reads multiple bytes into an array   |
| <code>int available()</code>         | Returns the number of available bytes  |
| <code>void close()</code>            | Closes the input stream  |
| <code>boolean markSupported()</code> | Checks if the stream supports the <code>mark()</code> and <code>reset()</code> methods |

| <b>Method</b>            | <b>Description</b>                       |
|--------------------------|--|
| void mark(int readlimit) | Marks the current position in the stream |
| void reset()             | Resets the stream to the marked position |
| long skip(long n)        | Skips and discards n bytes of data       |

## Output Stream

- An **OutputStream** writes **data** to a destination (such as a file, monitor, or network).
- Like InputStream, OutputStream is also an **abstract class** and must be subclassed to be used.

### Common Subclasses of OutputStream:

| <b>Subclass</b>          | <b>Description</b>                                      |
|--------------------------|---|
| FileOutputStream         | Writes data to files                                    |
| ByteArrayOutputStream    | Writes data to byte arrays                              |
| DataOutputStream         | Writes primitive data types                             |
| PrintStream              | Prints formatted representations of objects             |
| ObjectOutputStream       | Writes objects to the output stream                     |
| StringBufferOutputStream | Deprecated – used to write characters to a StringBuffer |

Example: Creating an OutputStream

```
OutputStream output = new FileOutputStream("example.txt");
```

### Common Methods of OutputStream

| <b>Method</b>            | <b>Description</b>                                 |
|--------------------------|--|
| void write(int b)        | Writes a single byte                               |
| void write(byte[] array) | Writes a sequence of bytes from an array           |
| void flush()             | Forces all buffered data to be written immediately |

| <b>Method</b> | <b>Description</b>       |
|---------------|--------------------------|
| void close()  | Closes the output stream |

**OutputStream:** Java application uses an **output stream** to write data to a destination; it may be a file, an array, a peripheral device, or a socket.

### What is FileReader?

- FileReader is a character-based stream in Java.
- It is used to read data from a file — character by character.
- It is mainly used for reading text files (like .txt, .csv).
- It automatically handles character encoding (like UTF-8).

### How to Create a FileReader?

```
FileReader fr = new FileReader("filename.txt");
```

- This opens the file filename.txt for reading.

If the file doesn't exist, it throws a FileNotFoundException.

### Common Methods of FileReader

| <b>Method</b>     | <b>Description</b>   |
|-------------------|--|
| read()            | Reads a single character and returns it as an integer. Returns -1 when the end of file is reached. |
| read(char[] cbuf) | Reads characters into a char array.  |
| close()           | Closes the FileReader to free resources.   |

### Example of FileReader

```
1 import java.io.FileReader;
2 public class ReadExample {
3     public static void main(String[] args) {
4         try {
5             FileReader reader = new FileReader("example.txt");
6             int character;
7             while ((character = reader.read()) != -1) {
8                 System.out.print((char) character);
9             }
10            reader.close();
11        } catch (Exception e) {
12            System.out.println(e);
13        }
14    }
15 }
```

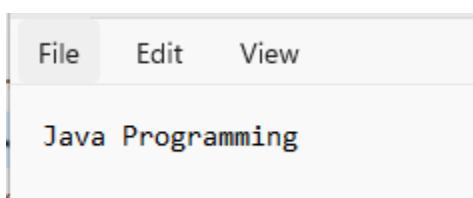
Note: if The File example.txt is not created by the user in the current working directory.

```
D:\Java Code\IO>javac ReadExample.java
D:\Java Code\IO>java ReadExample
java.io.FileNotFoundException: example.txt (The system cannot find the file specified)
```

When file example.txt is created, and it is blank

```
D:\Java Code\IO>javac ReadExample.java
D:\Java Code\IO>java ReadExample
```

When file example.txt has some data



```
D:\Java Code\IO>javac ReadExample.java
```

```
D:\Java Code\IO>java ReadExample  
Java Programming
```

Java Programming

Object-oriented programming

```
D:\Java Code\IO>java ReadExample  
Java Programming
```

Object-oriented programming

Java Programming

Object-oriented programming

12345%^^%&&^\*\*\*&12300i08877

```
D:\Java Code\IO>java ReadExample  
Java Programming
```

Object-oriented programming

12345%^^%&&^\*\*\*&12300i08877

### **Explanation:**

```
import java.io.FileReader; // Importing the FileReader class from java.io package
public class ReadExample {
    public static void main(String[] args) {
        try { // Using try-catch to handle exceptions (like file not found)
            // Creating a FileReader object and opening "example.txt" for reading
            FileReader reader = new FileReader("example.txt");
            int character; // Variable to store the character read (as an integer)
            // Reading characters one by one until end of file (EOF) is reached
            while ((character = reader.read()) != -1) {
                // reader.read() reads one character at a time and returns its ASCII value
                // We cast it to (char) to print the actual character instead of ASCII number
                System.out.print((char) character);
            }
            // Closing the FileReader after reading is complete
            reader.close();
        }
        // Catching any exception that occurs during reading (like file not found)
        catch (Exception e) {
            // Printing the stack trace for debugging if an error happens
            System.out.println(e);
        }
    }
}
```

### **What is FileWriter?**

- FileWriter is a character-based output stream in Java.
- It is used to write data to a file — character by character.
- It is used for creating new files or writing to existing text files.

### **How to Create a FileWriter?**

```
FileWriter fw = new FileWriter("filename.txt");
```

- This creates a new file or opens the existing file for writing.
- If the file exists, its content will be overwritten unless you specify append mode.

### To append data instead of overwriting:

```
FileWriter fw = new FileWriter("filename.txt", true);
```

### Common Methods of FileWriter

| Method             | Description                    |
|--------------------|--------------------------------|
| write(int c)       | Writes a single character.     |
| write(char[] cbuf) | Writes an array of characters. |
| write(String str)  | Writes a string.               |
| close()            | Closes the FileWriter.         |

### Example of FileWriter

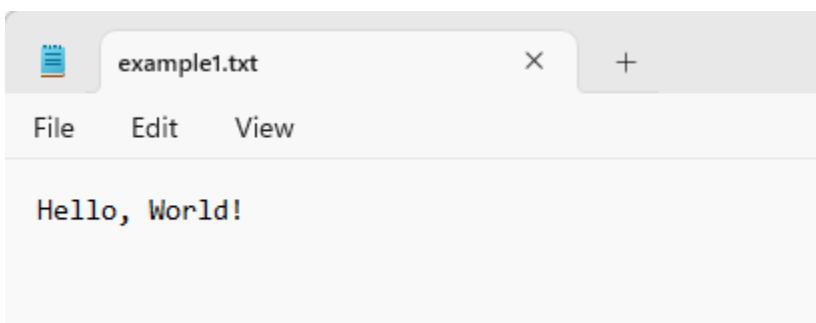
```
File Edit View  
this content is already present in the file
```

```
1 import java.io.FileWriter;  
2 public class WriteExample {  
3     public static void main(String[] args) {  
4         try {  
5             FileWriter writer = new FileWriter("example1.txt");  
6             writer.write("Hello, World!");  
7             writer.close();  
8             System.out.println("Data written successfully!");  
9         } catch (Exception e) {  
10             System.out.println(e);  
11         }  
12     }  
13 }
```

### Output:

```
D:\Java Code\IO>javac WriteExample.java
```

```
D:\Java Code\IO>java WriteExample  
Data written successfully!
```



A screenshot of a code editor window titled "WriteExample1.java". The code is a Java program that uses a `FileWriter` to append the string "Hello, this is a new line added to the file!\n" to a file named "filename.txt". The code is color-coded for syntax highlighting.

```
1 import java.io.FileWriter;  
2 public class WriteExample1 {  
3     public static void main(String[] args) {  
4         try {  
5             // Creating a FileWriter object  
5             // "filename.txt" is the file where data will be written  
5             // 'true' means data will be appended (not overwritten)  
6             FileWriter fw = new FileWriter("filename.txt", true);  
7             // Writing a string into the file  
8             fw.write("Hello, this is a new line added to the file!\n");  
9             // Closing the FileWriter after writing is done  
10            fw.close();  
11            System.out.println("Data successfully written to the file.");  
12        }  
13        catch (Exception e) {  
14            System.out.println(e);  
15        }  
16    }  
17}  
18}  
19}  
20}
```

"filename.txt" is the file where data will be written

```
D:\Java Code\IO>javac WriteExample1.java  
D:\Java Code\IO>java WriteExample1  
Data successfully written to the file.
```

"filename.txt" is the file where data will be writtenHello, this is a new line added to the file!

## **Practice Programs:**

1. Write a Java program that reads a file name from the user and attempts to open and read the file. Define a method `readFile()` that throws a `FileNotFoundException` using the `throws` keyword. In the main method, call this method and handle the exception using a try-catch block. Display an appropriate message if the file is not found. Use a finally block to ensure a message like "File operation attempted" is printed.
2. Write a Java program that takes user input for a student's name, roll number, and grade, and writes this information to a file named `student.txt` using `FileWriter`. Ensure the program appends the data to the file if it already exists. Handle any exceptions using try-catch and display an appropriate message if an error occurs.

### *Sample File Content:*

Name: Aman, Roll Number: 120112, Grade: A

Name: Parul, Roll Number: 120131, Grade: B

3. Write a Java program that reads the contents of a file named `student.txt` using `FileReader` and displays the data on the console. Handle `FileNotFoundException` if the file does not exist and display an appropriate error message. Use a try-catch block for exception handling.

## **Java Collection Framework**

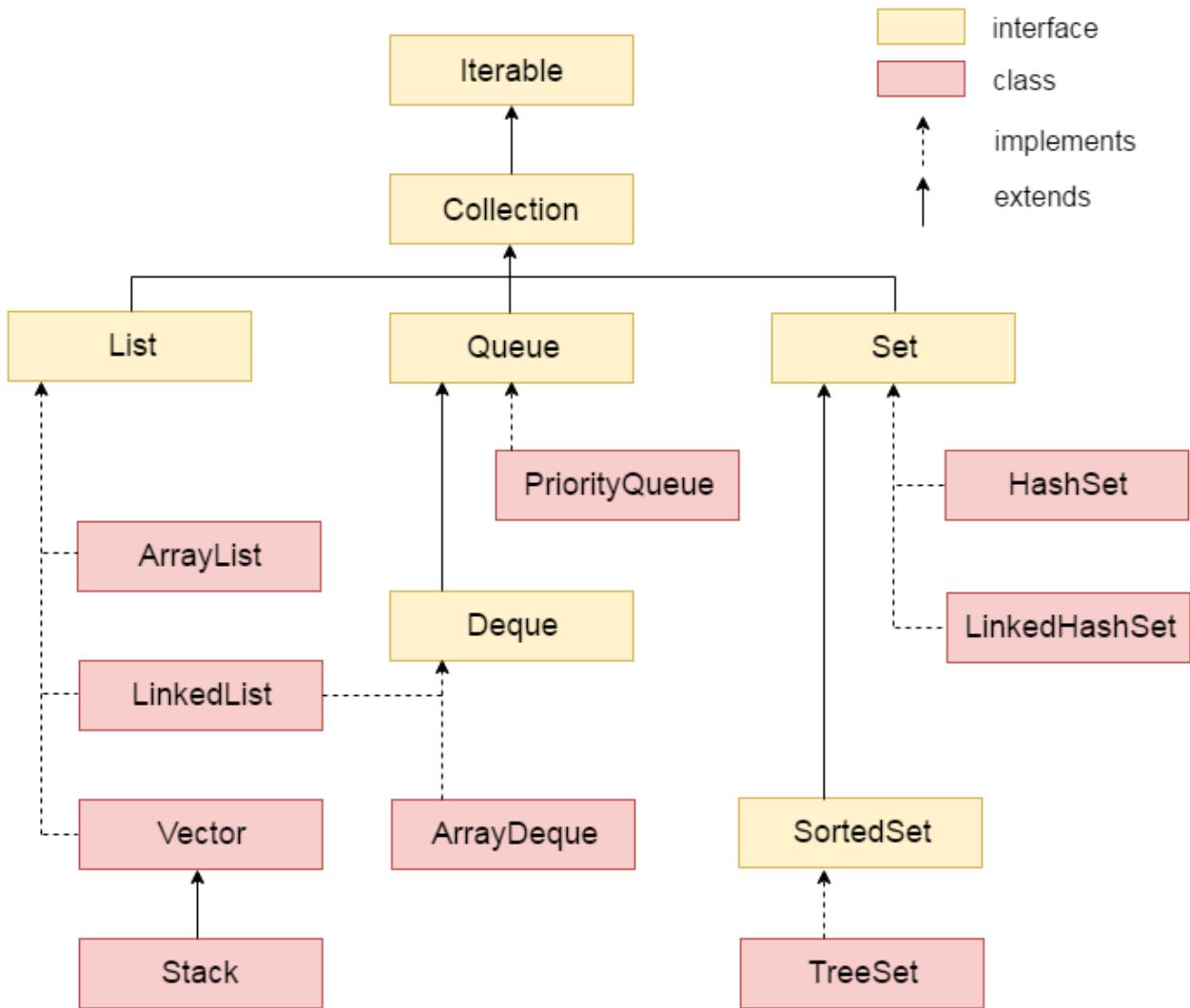
- A Group of objects is called a collection.
- Standard techniques or guidelines are called Frameworks.

It provides:

1. Organization of Objects (Data Structures): Arrangement
  - Array
  - Stack
  - LinkedList
  - Map(Key-Value)
  - Tree etc...
2. Operations on data structures:
  - Insert
  - Delete
  - Traverse
  - Add
  - Edit
  - Find
  - Duplicate
  - Shuffle
  - Max
  - Min
  - Sort
  - Search
  - Merge etc.

# Hierarchy of Collection Framework

Let us see the hierarchy of the collection framework. The **java.util** package contains all the classes and interfaces for the Collection framework.



## **Some behavior of Collection:**

1. All are resizable, have No issue with size, and are flexible.
2. Auto grows and shrinks.
3. Allow multi types of objects
4. Ordered/unordered
5. Sorted/Unsorted
6. Allow Unique/Duplicate
7. Allow null or not
8. Synchronous/Asynchronous
9. Key Value(K-V) pair (in only map)

## **Methods of Collection interface:**

| No. | Method                                 | Description  |
|-----|--|--|
| 1   | public boolean add(Object element)     | is used to insert an element in this collection.   |
| 2   | public boolean addAll(Collection c)    | is used to insert the specified collection elements in the invoking collection.            |
| 3   | public boolean remove(Object element)  | is used to delete an element from this collection.   |
| 4   | public boolean removeAll(Collection c) | is used to delete all the elements of specified collection from the invoking collection.   |
| 5   | public boolean retainAll(Collection c) | is used to delete all the elements of invoking collection except the specified collection. |
| 6   | public int size()                      | return the total number of elements in the collection.                                     |
| 7   | public void clear()                    | removes the total no of element from the collection.                                       |

|    |  |  |
|----|--|--|
| 8  | public boolean contains(Object element)  | is used to search an element.                                  |
| 9  | public boolean containsAll(Collection c) | is used to search the specified collection in this collection. |
| 10 | public Iterator iterator()               | returns an iterator.   |
| 11 | public Object[] toArray()                | converts collection into array.                                |
| 12 | public boolean isEmpty()                 | checks if collection is empty.                                 |
| 13 | public boolean equals(Object element)    | matches two collection.  |
| 14 | public int hashCode()                    | returns the hashcode number for collection.                    |

### Bubble sort (Not in Syllabus but discussing for the problem requirement)

- Compare adjacent elements in the array.
- Swap them if they are in the wrong order.
- Repeat the process for all elements until the array is sorted.
- Largest elements "bubble up" to their correct positions in each pass.
- Stop when no more swaps are needed.

### Time Complexity:

- Worst/Average Case:  $O(n^2)$
- Best Case (Already Sorted):  $O(n)$

### Properties:

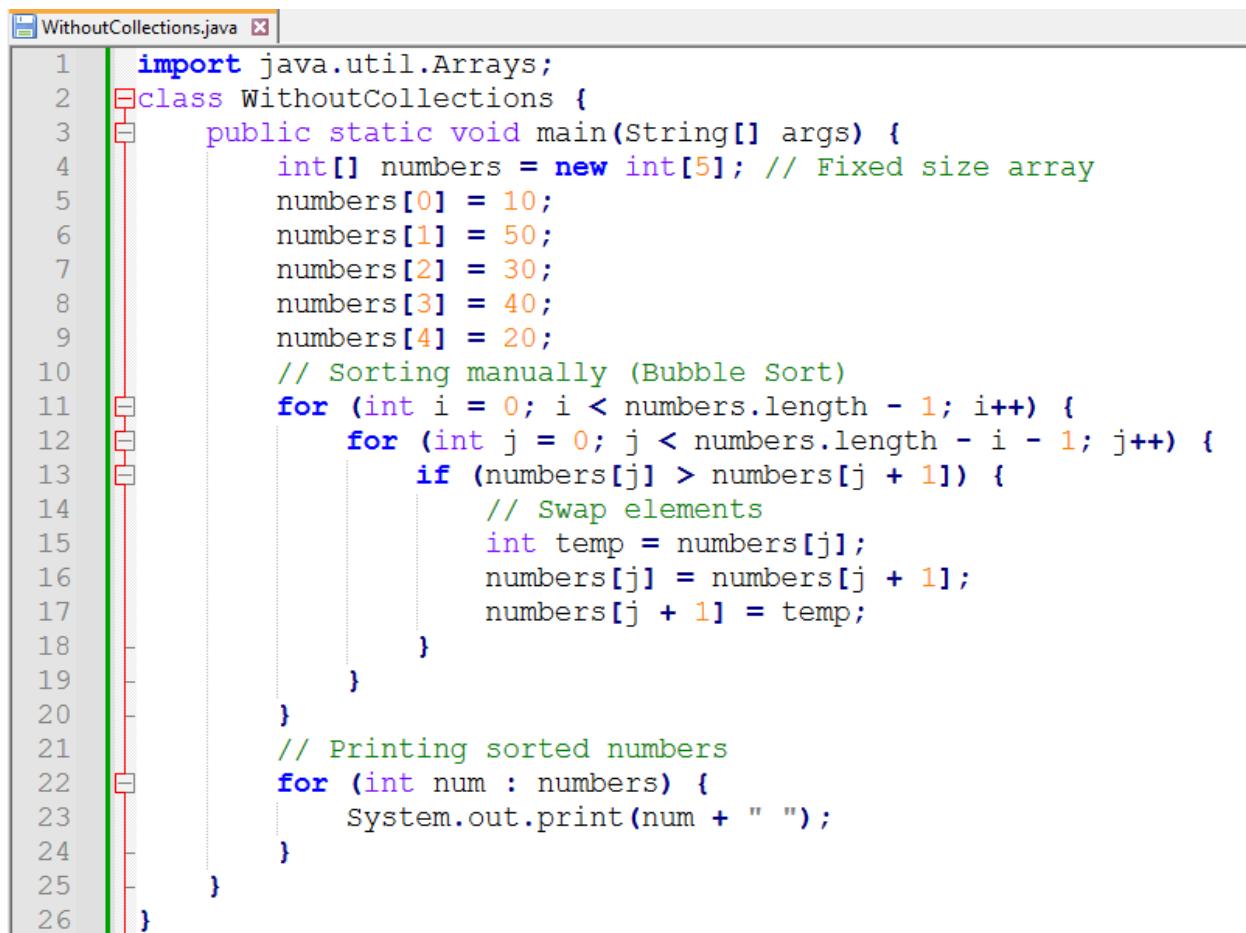
- Fixed Size: Arrays require a predefined size.
- Manual Sorting: We need to write custom sorting logic.
- Adding/Removing Elements: Requires shifting elements manually.
- Bubble Sort is one of the simplest sorting algorithms.
- It uses basic loops and swapping, making it easy to implement and understand.
- Bubble Sort shows each swap step clearly, making it a good choice for demonstrating sorting logic.

- Bubble Sort is inefficient for large datasets ( $O(n^2)$  complexity), but its performance is acceptable for small arrays like 5 integers or 3 students.

## Questions:

- Write a Java program that stores five integer values in an array, sorts them in ascending order without using the Java Collection Framework, and then prints the sorted numbers. Implement the sorting logic manually using Bubble Sort.
- Write a Java program to store student details (name and age) in a fixed-size array. Implement a sorting mechanism to arrange the students in ascending order based on their age without using the Java Collection Framework. Use Bubble Sort to perform the sorting and then display the sorted list of students.

## Program without Using Collections (Using Arrays)



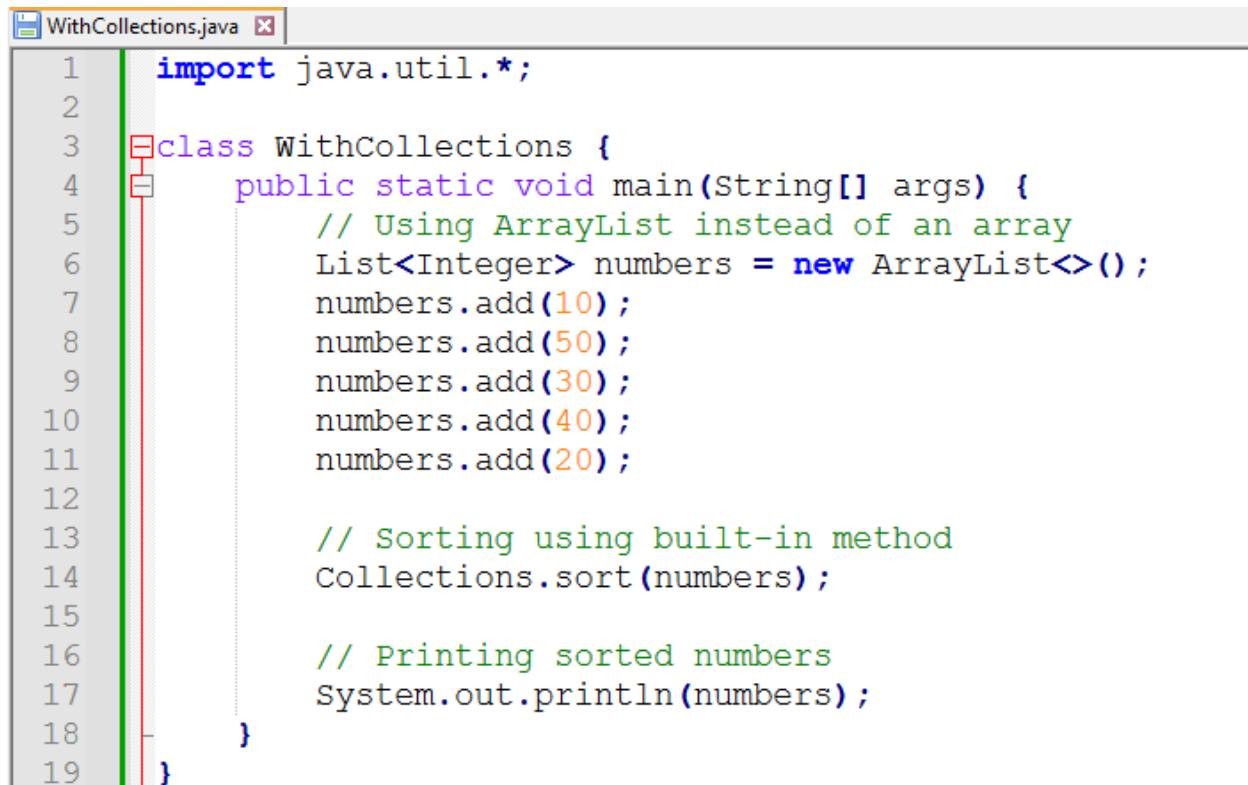
```

1 import java.util.Arrays;
2 class WithoutCollections {
3     public static void main(String[] args) {
4         int[] numbers = new int[5]; // Fixed size array
5         numbers[0] = 10;
6         numbers[1] = 50;
7         numbers[2] = 30;
8         numbers[3] = 40;
9         numbers[4] = 20;
10        // Sorting manually (Bubble Sort)
11        for (int i = 0; i < numbers.length - 1; i++) {
12            for (int j = 0; j < numbers.length - i - 1; j++) {
13                if (numbers[j] > numbers[j + 1]) {
14                    // Swap elements
15                    int temp = numbers[j];
16                    numbers[j] = numbers[j + 1];
17                    numbers[j + 1] = temp;
18                }
19            }
20        }
21        // Printing sorted numbers
22        for (int num : numbers) {
23            System.out.print(num + " ");
24        }
25    }
26 }
```

Output:

```
D:\Java Code\collection>javac WithoutCollections.java
D:\Java Code\collection>java WithoutCollections
10 20 30 40 50
```

### Program code Using Collections (ArrayList)



```
WithCollections.java
1 import java.util.*;
2
3 class WithCollections {
4     public static void main(String[] args) {
5         // Using ArrayList instead of an array
6         List<Integer> numbers = new ArrayList<>();
7         numbers.add(10);
8         numbers.add(50);
9         numbers.add(30);
10        numbers.add(40);
11        numbers.add(20);
12
13        // Sorting using built-in method
14        Collections.sort(numbers);
15
16        // Printing sorted numbers
17        System.out.println(numbers);
18    }
19}
```

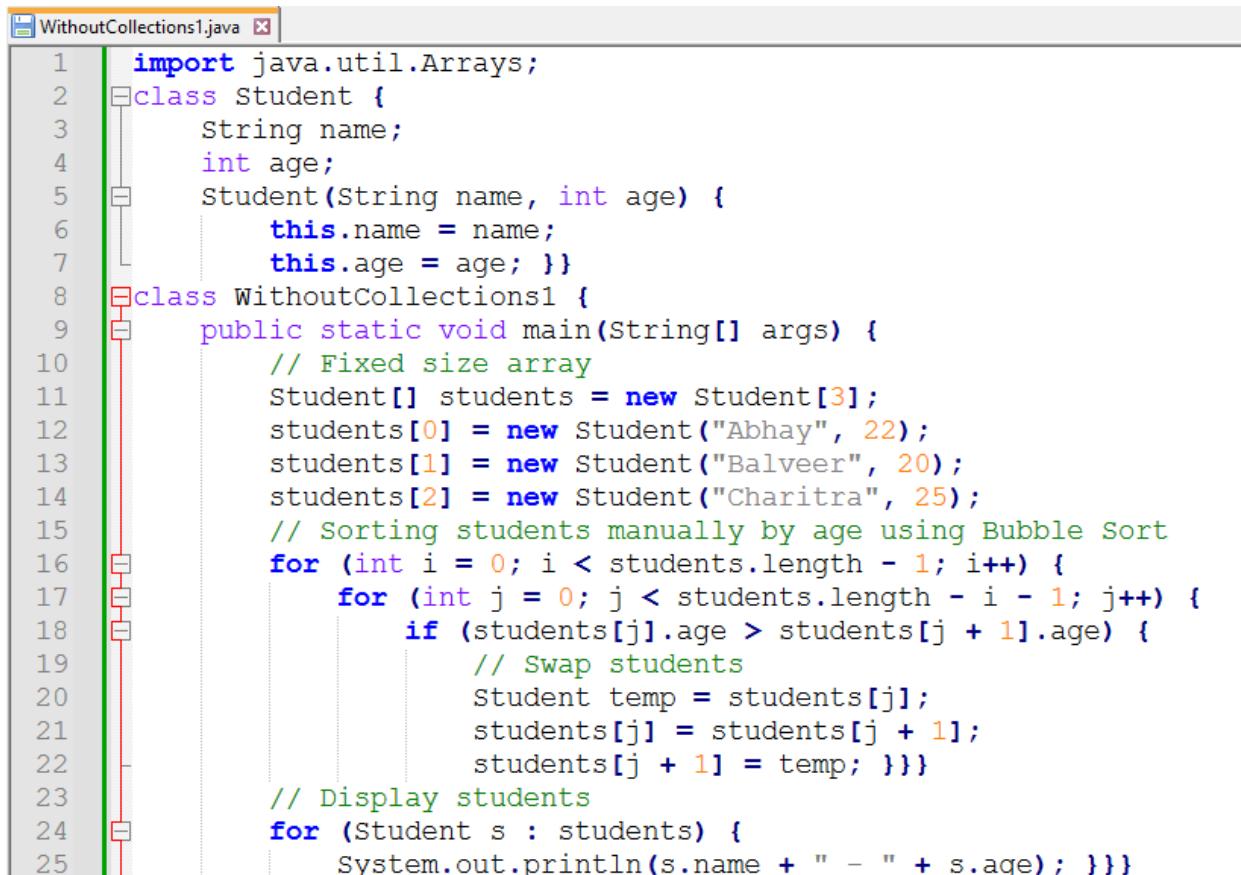
```
D:\Java Code\collection>javac WithCollections.java
D:\Java Code\collection>java WithCollections
[10, 20, 30, 40, 50]
```

### Comparison Table

| Feature            | Without Collections (Array)       | With Collections (ArrayList)    |
|--------------------|-----------------------------------|---------------------------------|
| Size               | Fixed                             | Dynamic                         |
| Sorting            | Manual Bubble Sort                | Collections.sort()              |
| Insertion/Deletion | Complex, manual shifting required | Simple add() / remove() methods |
| Flexibility        | Requires manual management        | Built-in operations             |

Write a Java program to store student details (name and age) in a fixed-size array. Implement a sorting mechanism to arrange the students in ascending order based on their age without using the Java Collection Framework. Use Bubble Sort to perform the sorting and then display the sorted list of students.

### Without Using Collections (Using Arrays)



```
WithoutCollections1.java
1 import java.util.Arrays;
2 class Student {
3     String name;
4     int age;
5     Student(String name, int age) {
6         this.name = name;
7         this.age = age; }
8 class WithoutCollections1 {
9     public static void main(String[] args) {
10         // Fixed size array
11         Student[] students = new Student[3];
12         students[0] = new Student("Abhay", 22);
13         students[1] = new Student("Balveer", 20);
14         students[2] = new Student("Charitra", 25);
15         // Sorting students manually by age using Bubble Sort
16         for (int i = 0; i < students.length - 1; i++) {
17             for (int j = 0; j < students.length - i - 1; j++) {
18                 if (students[j].age > students[j + 1].age) {
19                     // Swap students
20                     Student temp = students[j];
21                     students[j] = students[j + 1];
22                     students[j + 1] = temp; }}
23         // Display students
24         for (Student s : students) {
25             System.out.println(s.name + " - " + s.age); }}
```

```
D:\Java Code\collection>javac WithoutCollections1.java

D:\Java Code\collection>java WithoutCollections1
Balveer - 20
Abhay - 22
Charitra - 25
```

## Using Collection framework

```
WithCollections1.java
1 import java.util.*;
2 class Student {
3     String name;
4     int age;
5     Student(String name, int age) {
6         this.name = name;
7         this.age = age;
8     }
9
10 class WithCollections1 {
11     public static void main(String[] args) {
12         // Using ArrayList to store students
13         List<Student> students = new ArrayList<>();
14         students.add(new Student("Abhay", 22));
15         students.add(new Student("Balveer", 20));
16         students.add(new Student("Charitra", 25));
17
18         // Sorting students by age using Collections.sort()
19         students.sort(Comparator.comparingInt(s -> s.age));
20
21         // Display students
22         for (Student s : students) {
23             System.out.println(s.name + " - " + s.age);
24         }
25     }
26 }
```

s -> s.age is a lambda expression.

```
D:\Java Code\collection>java WithCollections1
Balveer - 20
Abhay - 22
Charitra - 25
```

## Comparison Table

| Feature         | Without Collections (Array) | With Collections (ArrayList)       |
|-----------------|-----------------------------|------------------------------------|
| Size Management | Fixed, hardcoded size       | Dynamic, can grow/shrink           |
| Sorting         | Manual Bubble Sort          | Collections.sort() with Comparator |
| Adding/Removing | Complex, requires shifting  | Simple add() and remove()          |
| Code Complexity | More lines, error-prone     | Cleaner, easier to maintain        |

## Iterable and Iterator Interface

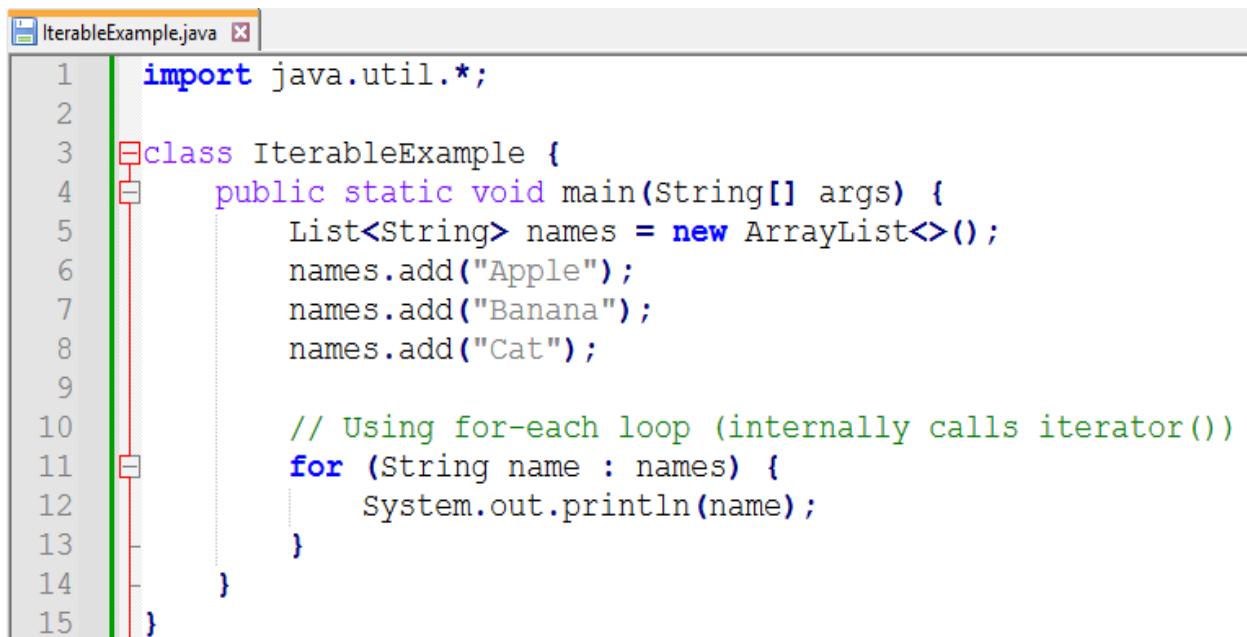
- The Iterable and Iterator interfaces are part of the Java Collections Framework, providing a way to traverse or iterate over elements in a collection.
- While they are related, they have distinct purposes and roles.

### Iterable Interface

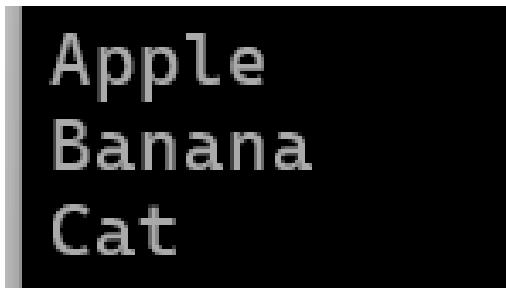
The Iterable interface is present in `java.lang` package and is the root interface for all collection classes. It allows an object to be the target of the for-each loop.

#### Key Points:

- It has only one method: `iterator()`, which returns an Iterator.
- Implementing `Iterable<T>` allows the collection to be iterated using an enhanced for loop.
- Common collections like `ArrayList`, `HashSet`, and `LinkedList` implement Iterable.



```
IterableExample.java
1 import java.util.*;
2
3 class IterableExample {
4     public static void main(String[] args) {
5         List<String> names = new ArrayList<>();
6         names.add("Apple");
7         names.add("Banana");
8         names.add("Cat");
9
10        // Using for-each loop (internally calls iterator())
11        for (String name : names) {
12            System.out.println(name);
13        }
14    }
15 }
```

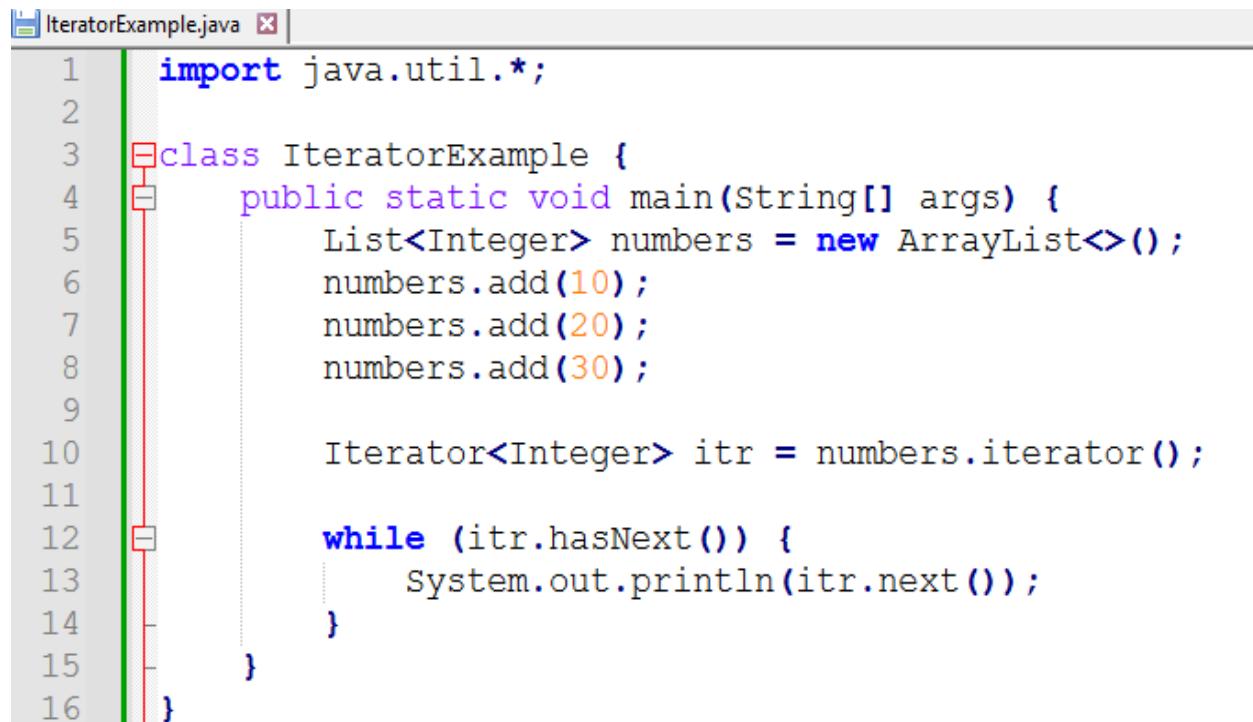


## Iterator Interface

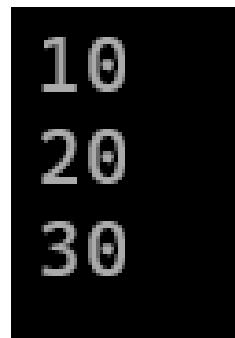
- The Iterator interface is present in the Java.util package and provides methods to iterate over a collection.
- The iterator interface allows iterating the elements in the forward direction only.

### Key Methods:

- hasNext(): Returns true if more elements exist.
- next(): Returns the next element.
- remove(): Removes the last element returned by next().



```
1 import java.util.*;
2
3 class IteratorExample {
4     public static void main(String[] args) {
5         List<Integer> numbers = new ArrayList<>();
6         numbers.add(10);
7         numbers.add(20);
8         numbers.add(30);
9
10        Iterator<Integer> itr = numbers.iterator();
11
12        while (itr.hasNext()) {
13            System.out.println(itr.next());
14        }
15    }
16}
```



```
10
20
30
```

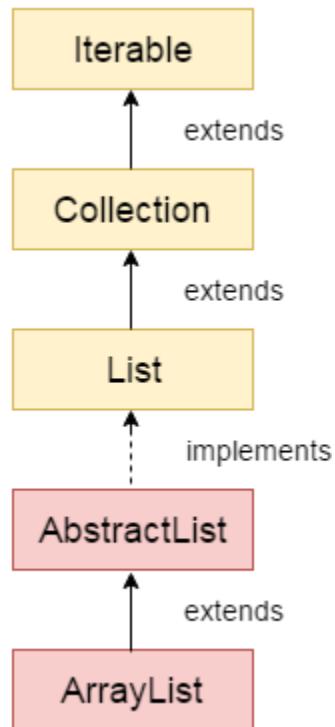
```
IteratorMethodsExample.java
1 import java.util.*;
2 class IteratorMethodsExample {
3     public static void main(String[] args) {
4         // Creating a list of integers
5         List<Integer> numbers = new ArrayList<>();
6         numbers.add(10);
7         numbers.add(20);
8         numbers.add(30);
9         numbers.add(40);
10        numbers.add(50);
11        // Getting an iterator for the list
12        Iterator<Integer> itr = numbers.iterator();
13        // Using hasNext() to check if elements are available
14        while (itr.hasNext()) {
15            // Using next() to get the next element
16            int num = itr.next();
17            System.out.println("Current Element: " + num);
18
19            // Using remove() to remove elements that meet a condition
20            if (num == 30) {
21                itr.remove(); // Removes the last returned element (30)
22                System.out.println("Element 30 removed");
23            }
24        }
25    }
26 }
```

```
Current Element: 10
Current Element: 20
Current Element: 30
Element 30 removed
Current Element: 40
Current Element: 50
Final List: [10, 20, 40, 50]
```

## Java ArrayList class

Java ArrayList class uses a dynamic array for storing the elements. It inherits AbstractList class and implements List interface.

- Java ArrayList class can contain duplicate elements.
- Java ArrayList class maintains insertion order.
- Java ArrayList class is non-synchronized.
- Java ArrayList allows random access because the array works on an index basis.
- ArrayList is Duplicate, Ordered, Unsorted, Asynchronous, Allow Null



## **ArrayList class declaration:**

```
public class ArrayList<E> extends AbstractList<E> implements List<E>
```

### Constructors of Java ArrayList

| <b>Constructor</b>      | <b>Description</b>   |
|-------------------------|--|
| ArrayList()             | It is used to build an empty array list.   |
| ArrayList(Collection c) | It is used to build an array list that is initialized with the elements of the collection c. |
| ArrayList(int capacity) | It is used to build an array list that has the specified initial capacity.                   |

### Methods of Java ArrayList

| <b>Method</b>                       | <b>Description</b>  |
|-------------------------------------|---|
| void add(int index, Object element) | It is used to insert the specified element at the specified position index in a list.   |
| boolean addAll(Collection c)        | It is used to append all of the elements in the specified collection to the end of this list, in the order that they are returned by the specified collection's iterator. |
| void clear()                        | It is used to remove all of the elements from this list.  |

|   |  |
|---|--|
| int lastIndexOf(Object o)               | It is used to return the index in this list of the last occurrence of the specified element, or -1 if the list does not contain this element.  |
| Object[] toArray()                      | It is used to return an array containing all of the elements in this list in the correct order.  |
| Object[] toArray(Object[] a)            | It is used to return an array containing all of the elements in this list in the correct order.  |
| boolean add(Object o)                   | It is used to append the specified element to the end of a list.   |
| boolean addAll(int index, Collection c) | It is used to insert all of the elements in the specified collection into this list, starting at the specified position.                       |
| Object clone()                          | It is used to return a shallow copy of an ArrayList.   |
| int indexOf(Object o)                   | It is used to return the index in this list of the first occurrence of the specified element, or -1 if the List does not contain this element. |
| void trimToSize()                       | It is used to trim the capacity of this ArrayList instance to be the list's current size.  |

## Java Non-generic Vs. Generic Collection

Java collection framework was non-generic before JDK 1.5. Since 1.5, it is generic.

Java's new generic collection allows you to have **only one type of object** in the collection. Now it is type-safe, so typecasting is not required at run time.

In the generic collection, we specify the type in **angular braces**. Now ArrayList is forced to have only specified types of objects in it. If you try to add another type of object, it gives *compile time error*.

1. ArrayList al=**new** ArrayList ( );//creating old non-generic arraylist

Let's see the new generic example of creating java collection.

1. ArrayList<String> al=**new** ArrayList<String>();//creating new generic arraylist

### ArrayList Program:

```
import java.util.*;
class TestCollection1{
public static void main(String args[]){
ArrayList<String> list=new ArrayList<String>();//Creating array list
list.add("Ravi");//Adding object in array list
list.add("Vijay");
list.add("Ravi");
list.add("Ajay");
//Traversing list through Iterator
```

```
Iterator itr=list.iterator();
while(itr.hasNext())// It returns true if iterator has more elements {
System.out.println(itr.next()); //fetch item.
}
```



```
}
```

```
C:\Windows\System32\cmd.exe
```

```
D:\1 Java\Programs>javac TestCollection1.java
```

```
D:\1 Java\Programs>java TestCollection1
```

```
Ravi
```

```
Vijay
```

```
Ravi
```

```
Ajay
```

### **//USER DEFINED ARRAYLIST**

```
class Student{  
    int rollno;  
    String name;  
    int age;  
    Student(int rollno,String name,int age){  
        this.rollno=rollno;  
        this.name=name;  
        this.age=age;  
    }  
}  
  
import java.util.*;  
class UserAL{  
    public static void main(String args[]){  
        //Creating user-defined class objects  
        Student s1=new Student(101,"Saurabh",23);  
        Student s2=new Student(102,"Ravi",21);  
        Student s3=new Student(103,"Vineet",25);  
        //creating arraylist  
        ArrayList<Student> al=new ArrayList<Student>();  
        al.add(s1);//adding Student class object  
        al.add(s2);  
        al.add(s3);  
        //Getting Iterator  
        Iterator itr=al.iterator();  
        //traversing elements of ArrayList object  
        while(itr.hasNext()){
```

```
        Student st=(Student)itr.next(); //type casting
        System.out.println(st.rollno+" "+st.name+" "+st.age);
    }
}
}
```

```
D:\1 Java\Programs\ArrayList>javac UserAL.java

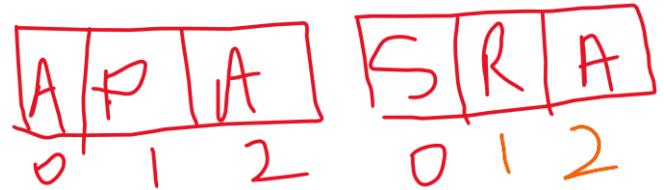
D:\1 Java\Programs\ArrayList>java UserAL
101 Saurabh 23
102 Ravi 21
103 Vineet 25
```

```
//addAll() method
```

```
import java.util.*;
class AddAll{
    public static void main(String args[]){
        ArrayList<String> al=new ArrayList<String>();
        al.add("Anurag");
        al.add("Praksah");
        al.add("Ajay");
```

```
ArrayList<String> al2=new ArrayList<String>();
al2.add("Saurabh");
al2.add("Rohit");
al2.add("Ajay");
```

```
al.addAll(al2);//adding second list al2 in first list al
Iterator itr=al.iterator();
while(itr.hasNext()){
    System.out.println(itr.next());
}
}
```



```
D:\1 Java\Programs\ArrayList>javac AddAll.java
```

```
D:\1 Java\Programs\ArrayList>java AddAll
```

```
Anurag
Praksah
Ajay
Saurabh
Rohit
Ajay
```

```
//Example of removeAll() method
```

```
import java.util.*;
class RemoveAll{
    public static void main(String args[]){
        ArrayList<String> al=new ArrayList<String>();
        al.add("Anurag");
        al.add("Praksah");
        al.add("Ajay");

        ArrayList<String> al2=new ArrayList<String>();
        al2.add("Saurabh");
        al2.add("Rohit");
        al2.add("Ajay");
        al.removeAll(al2); //it removes duplicate element "Ajay" from al.
        System.out.println("iterating the elements after removing the elements of al2...");
        Iterator itr=al.iterator();
        while(itr.hasNext()){
            System.out.println(itr.next());
        }
    }
}
```

```
D:\1 Java\Programs\ArrayList>javac RemoveAll.java

D:\1 Java\Programs\ArrayList>java RemoveAll
iterating the elements after removing the elements of al2...
Anurag
Praksah
```

### //Example of retainAll() method

```
import java.util.*;
class RetainAll{
    public static void main(String args[]){
        ArrayList<String> al=new ArrayList<String>();
        al.add("Anurag");
        al.add("Praksah");
        al.add("Ajay");

        ArrayList<String> al2=new ArrayList<String>();
        al2.add("Saurabh");
        al2.add("Rohit");
        al2.add("Ajay");
        al.removeAll(al2);//It will retain all duplicate elements only.
        System.out.println("iterating the elements after retain the elements of al2...");
        Iterator itr=al.iterator();
        while(itr.hasNext()){
            System.out.println(itr.next());
        }
    }
}
```

```
D:\1 Java\Programs\ArrayList>java RetainAll
iterating the elements after removing the elements of al2...
Ajay
```

## Java HashSet class

Java HashSet class is used to create a collection that uses a hash table for storage. It inherits the AbstractSet class and implements Set interface.

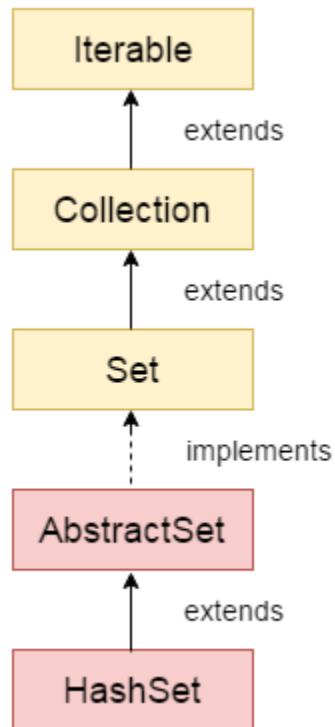
The important points about Java HashSet class are:

- HashSet stores the elements by using a mechanism called **hashing**.
- HashSet contains unique elements only.

## Difference between List and Set

List can contain duplicate elements whereas Set contains unique elements only.

Hierarchy of HashSet class



## HashSet class declaration

Let's see the declaration for java.util.HashSet class.

1. **public class HashSet<E> extends AbstractSet<E> implements Set<E>**

Constructors of Java HashSet class:

| Constructor           | Description   |
|-----------------------|---|
| HashSet()             | It is used to construct a default HashSet.  |
| HashSet(Collection c) | It is used to initialize the hash set by using the elements of the collection c.  |
| HashSet(int capacity) | It is used to initialize the capacity of the hash set to the given integer value capacity. The capacity grows automatically as elements are added to the HashSet. |

Methods of Java HashSet class:

| Method                     | Description  |
|----------------------------|--|
| void clear()               | It is used to remove all of the elements from this set.                            |
| boolean contains(Object o) | It is used to return true if this set contains the specified element.              |
| boolean add(Object o)      | It is used to adds the specified element to this set if it is not already present. |
| boolean isEmpty()          | It is used to return true if this set contains no elements.                        |

|                                |   |
|--------------------------------|---|
| boolean<br>remove(Object<br>o) | It is used to remove the specified element from this set if it is present.                            |
| Object clone()                 | It is used to return a shallow copy of this HashSet instance: the elements themselves are not cloned. |
| Iterator iterator()            | It is used to return an iterator over the elements in this set.                                       |
| int size()                     | It is used to return the number of elements in this set.  |

```

import java.util.*;
class TestHashSet{
    public static void main(String args[]){
        //Creating HashSet and adding elements
        HashSet<String> set=new HashSet<String>();
        set.add("Ravi");
        set.add("Vijay");
        set.add("Ravi");
        set.add("Ajay");
        //Traversing elements
        Iterator<String> itr=set.iterator();
        while(itr.hasNext()){
            System.out.println(itr.next());
        }
    }
}

```

```
C:\Windows\System32\cmd.exe
```

```
D:\1 Java\Programs\HashSet>javac TestHashSet.java
```

```
D:\1 Java\Programs\HashSet>java TestHashSet
```

```
Vijay
```

```
Ravi
```

```
Ajay
```

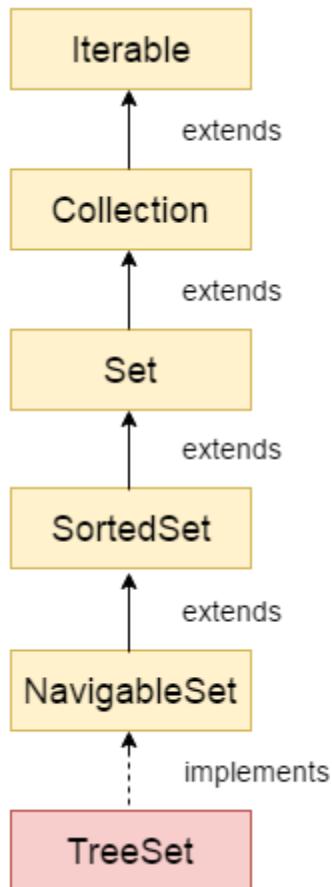
### Java TreeSet class

Java TreeSet class implements the Set interface that uses a tree for storage. It inherits AbstractSet class and implements NavigableSet interface. The objects of TreeSet class are stored in ascending order.

The important points about Java TreeSet class are:

- Contains unique elements only like HashSet.
- Access and retrieval times are quiet fast.
- Maintains ascending order.

Hierarchy of TreeSet class



### TreeSet class declaration

```
public class TreeSet<E> extends AbstractSet<E> implements NavigableSet<E>
```

Constructors of Java TreeSet class

| Constructor              | Description   |
|--------------------------|---|
| TreeSet()                | It is used to construct an empty tree set that will be sorted in an ascending order according to the natural order of the tree set. |
| TreeSet(Collection c)    | It is used to build a new tree set that contains the elements of the collection c.  |
| TreeSet(Comparator comp) | It is used to construct an empty tree set that will be sorted according to given comparator.  |

|                       |  |
|-----------------------|--|
| TreeSet(SortedSet ss) | It is used to build a TreeSet that contains the elements of the given SortedSet. |
|-----------------------|--|

## Methods of Java TreeSet class

| Method                       | Description   |
|------------------------------|---|
| boolean addAll(Collection c) | It is used to add all of the elements in the specified collection to this set.    |
| boolean contains(Object o)   | It is used to return true if this set contains the specified element.             |
| boolean isEmpty()            | It is used to return true if this set contains no elements.                       |
| boolean remove(Object o)     | It is used to remove the specified element from this set if it is present.        |
| void add(Object o)           | It is used to add the specified element to this set if it is not already present. |
| void clear()                 | It is used to remove all of the elements from this set.                           |
| Object clone()               | It is used to return a shallow copy of this TreeSet instance.                     |
| Object first()               | It is used to return the first (lowest) element currently in this sorted set.     |
| Object last()                | It is used to return the last (highest) element currently in this sorted set.     |
| int size()                   | It is used to return the number of elements in this set.                          |

### //TreeSet Example

```
import java.util.*;
class TestTreeSet{
    public static void main(String args[]){
        //Creating and adding elements
        TreeSet<String> al=new TreeSet<String>();
```

```
al.add("Ravi");
al.add("Vijay");
al.add("Ravi");
al.add("Ajay");
//Traversing elements
Iterator<String> itr=al.iterator();
while(itr.hasNext()){
    System.out.println(itr.next());
}
}
```

---

```
C:\ Windows\System32\cmd.exe

D:\1 Java\Programs\TreeSet>javac TestTreeSet.java

D:\1 Java\Programs\TreeSet>java TestTreeSet
Ajay
Ravi
Vijay
```

# Introduction to Advanced Java

In core Java, we can develop standalone applications  
standalone —

1. GUI Applications (Desktop Applications) like MSWord, Calculator, Games etc
2. CUI Based Application (Console Application) like Cmd > Java Test

Through Advance java, we can develop **web applications** like Gmail,irctc,upes.ac.in etc.

client/s-----request (from across the globe) → Server

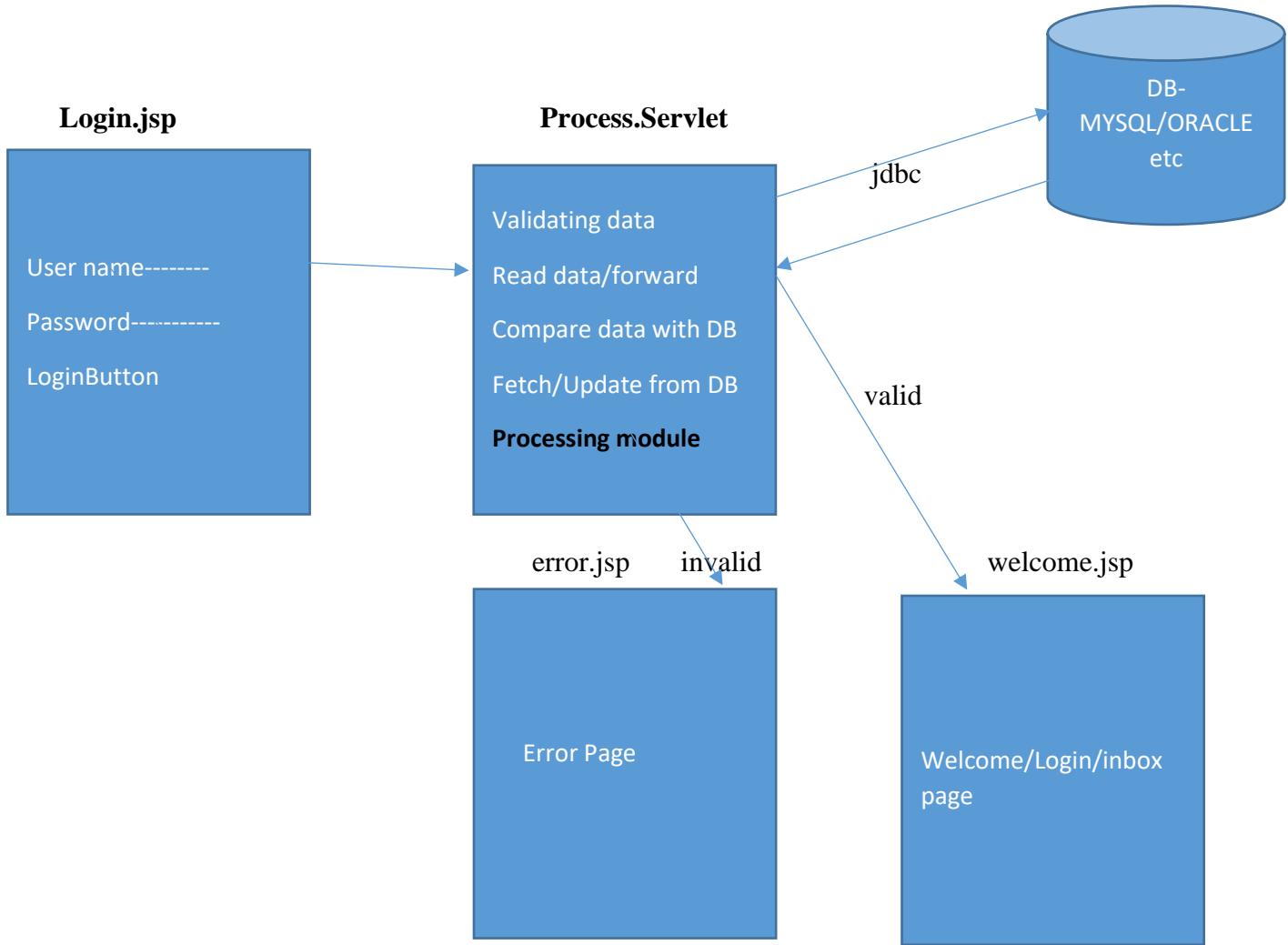
Server-----response→client

In advance java, we will learn 3 things, or We can develop Web applications through the following 3 technologies:

- JDBC: for connection with Java/Servlet
- JSP: Presentation logic/view
- Servlet: processing logic/business logic

According to Sun Microsystem, total Java is available in 3 editions:

- 1) Java Standard Edition(JSE/J2SE) (Core Java+ JDBC)
- 2) Java Enterprise Edition (JEE/J2EE) (Servlet + JSP)
- 3) Java micro edition (JME)/J2ME: mobile application, embedded microcontroller, cards, etc.



### Simple Web application example

**JSP: JAVA SERVER PAGES-** Meant for presentation logic to display something to the end user.

Wherever presentation logic is required, or we want to display something to the end user, we should go for JSP. End users can able to view inbox.jsp /error.jsp /login... these pages are view components, which contain presentation logic. So if you want to develop such view/presentation component, you should go for JSP.

### Servlet: It meant for processing

Wherever some processing/business logic is required, we should go for Servlet.

Whenever we enter the submit button, the internal request goes to some component, and we can't see that component; it means someone reading data/comparing data/validating data/fetching

/updating data, it means someone internally processing our requests to perform something. This component is called a process servlet or servlet.

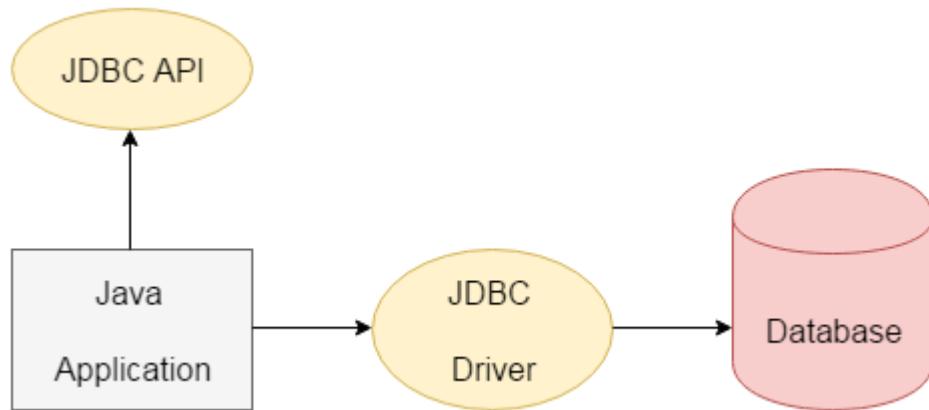
Wherever processing logic is required, we go for the servlet.

**JDBC: if the Java application wants to communicate with the database, then we should go for JDBC.**

Sometimes servlet (java component) is required to communicate with DB. Therefore, java to DB connection, if u want to communicate with DB, then we go for JDBC. Some technology must be required; this technology is nothing but JDBC.

# JDBC

Java JDBC is a Java API to connect and execute queries with the database. JDBC API uses JDBC drivers to connect with the database.



## Why use JDBC

Before JDBC (Java Database connectivity), ODBC (Open database connectivity) API was the database API to connect and execute queries with the database. But ODBC API uses ODBC driver, which is written in C language (i.e., platform dependent and unsecured). That is why Java has defined its own API (JDBC API) that uses JDBC drivers (written in Java language).

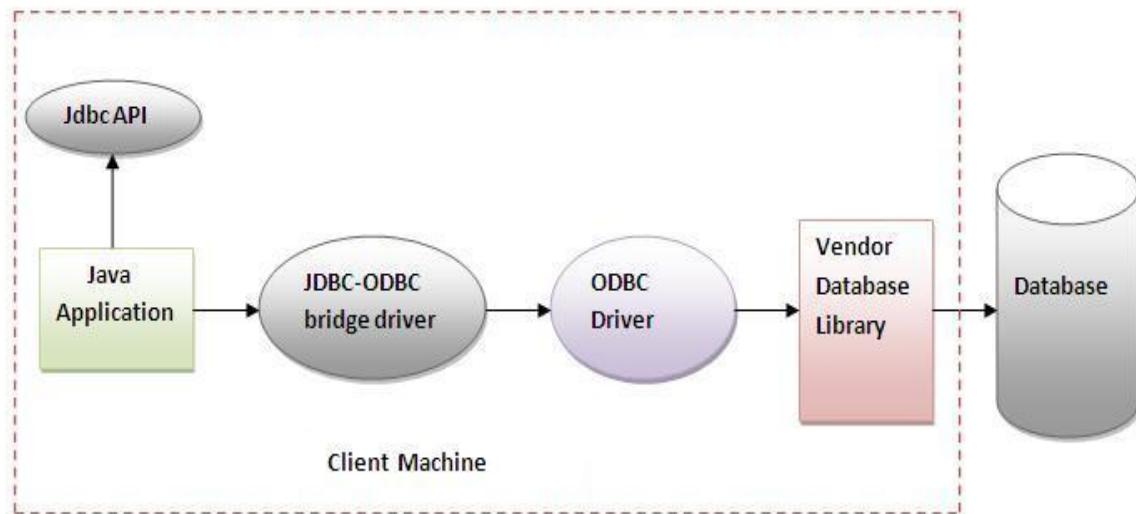
## JDBC Driver

JDBC Driver is a software component that enables Java applications to interact with the database. There are four types of JDBC drivers:

1. JDBC-ODBC bridge driver
2. Native-API driver (partially java driver)
3. Network Protocol driver (fully java driver)
4. **Thin driver (fully java driver)//Popular**

### 1) JDBC-ODBC bridge driver

The JDBC-ODBC bridge driver uses the ODBC driver to connect to the database. The JDBC-ODBC bridge driver converts JDBC method calls into ODBC function calls. This is now discouraged because of the thin driver.



**Figure-JDBC-ODBC Bridge Driver**

### **Advantages:**

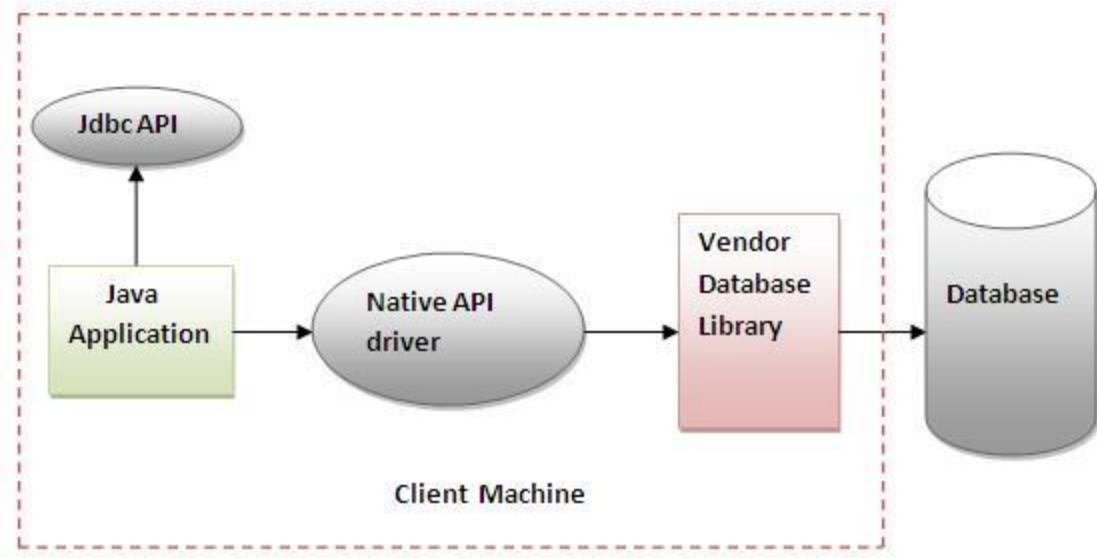
- easy to use.
- can be easily connected to any database.

### **Disadvantages:**

- Performance degraded because the JDBC method call is converted into the ODBC function calls.
- The ODBC driver needs to be installed on the client machine.

## **2) Native-API driver**

The Native API driver uses the client-side libraries of the database. The driver converts JDBC method calls into native calls of the database API. It is not written entirely in java.



**Figure- Native API Driver**

### **Advantage:**

- Performance upgraded than JDBC-ODBC bridge driver.

### **Disadvantage:**

- The Native driver needs to be installed on each client machine.
- The Vendor client library needs to be installed on the client machine.

### **3) Network Protocol driver**

The Network Protocol driver uses middleware (application server) that converts JDBC calls directly or indirectly into the vendor-specific database protocol. It is fully written in java.

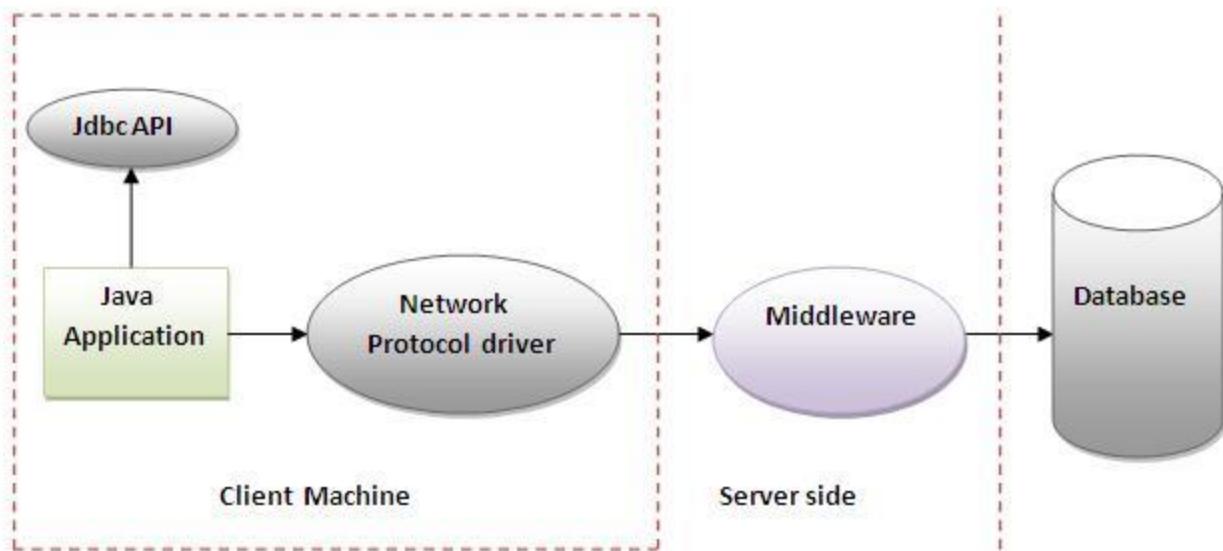


Figure- Network Protocol Driver

### **Advantage:**

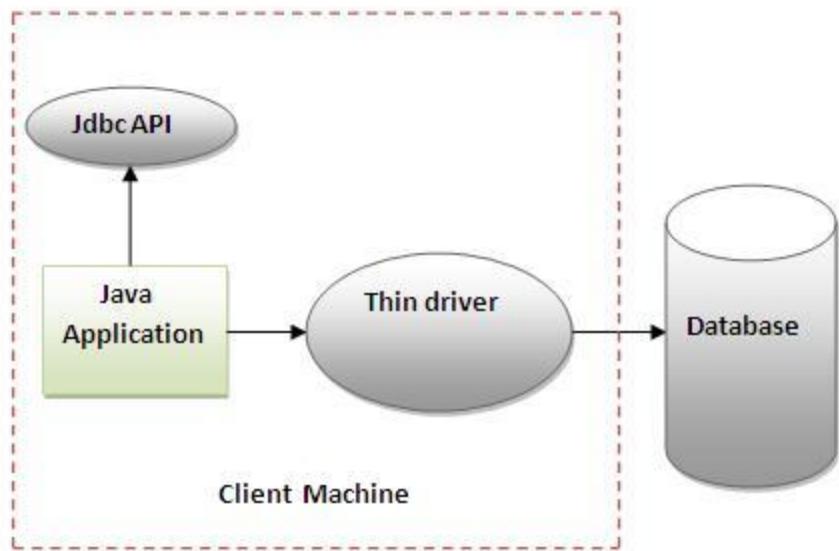
- No client-side library is required because of the application server can perform many tasks like auditing, load balancing, logging, etc.

### **Disadvantages:**

- Network support is required on the client machine.
- Requires database-specific coding to be done in the middle tier.
- Maintenance of the Network Protocol driver becomes costly because it requires database-specific coding to be done in the middle tier.

### **4) Thin driver**

The thin driver converts JDBC calls directly into the vendor-specific database protocol. That is why it is known as a thin driver. It is fully written in Java language.



**Figure- Thin Driver**

### **Advantage:**

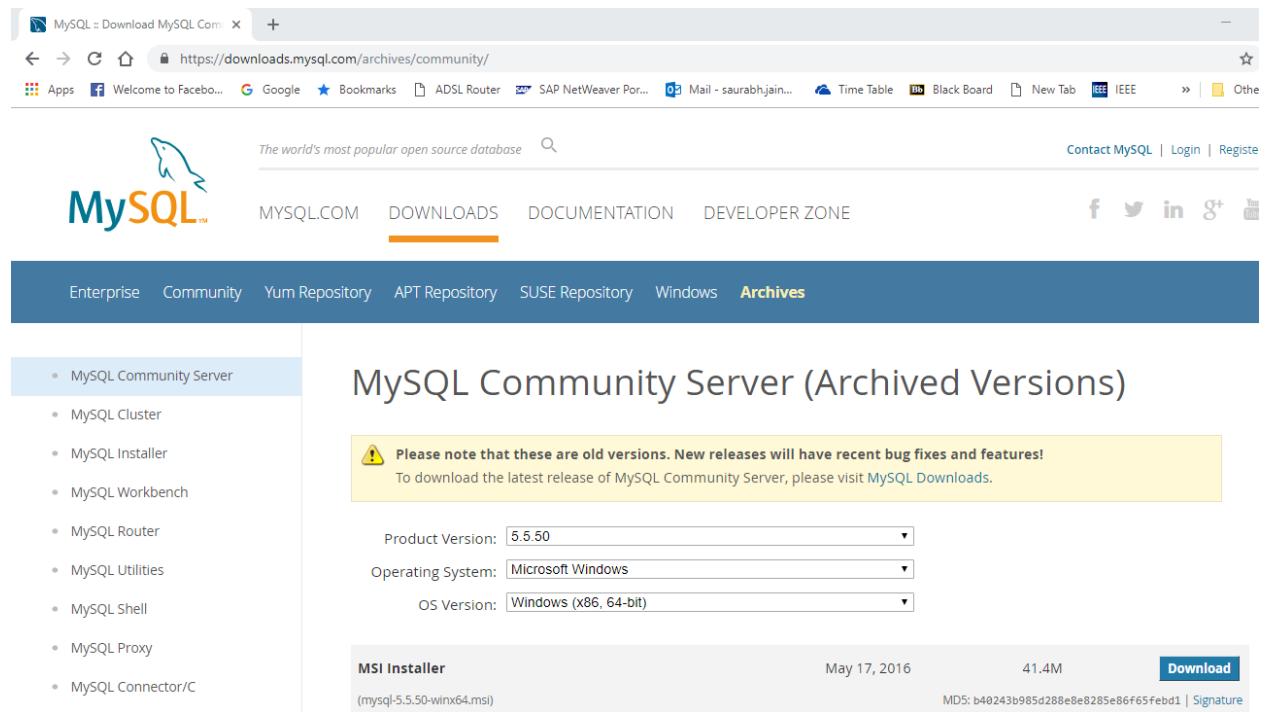
- Better performance than all other drivers.
- No software is required on the client side or server side.

### **Disadvantage:**

- Drivers depend on the Database.

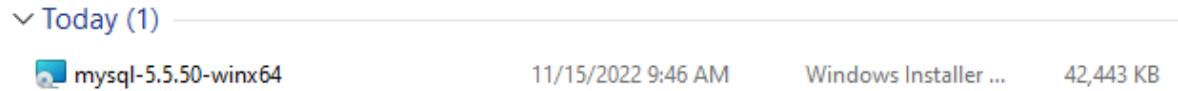
# Installation of MySQL Server

Download: mysql-5.5.50-winx64.msi installer (Approx 40Mb) from following website:  
[http://downloads.mysql.com/archives/community/](https://downloads.mysql.com/archives/community/)



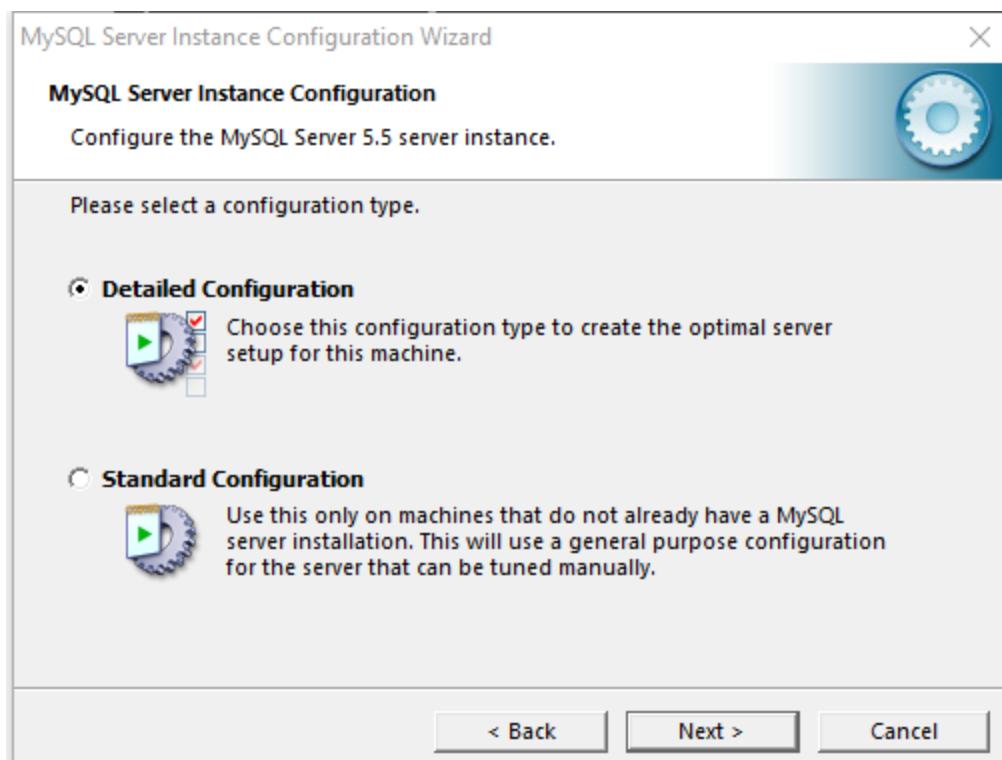
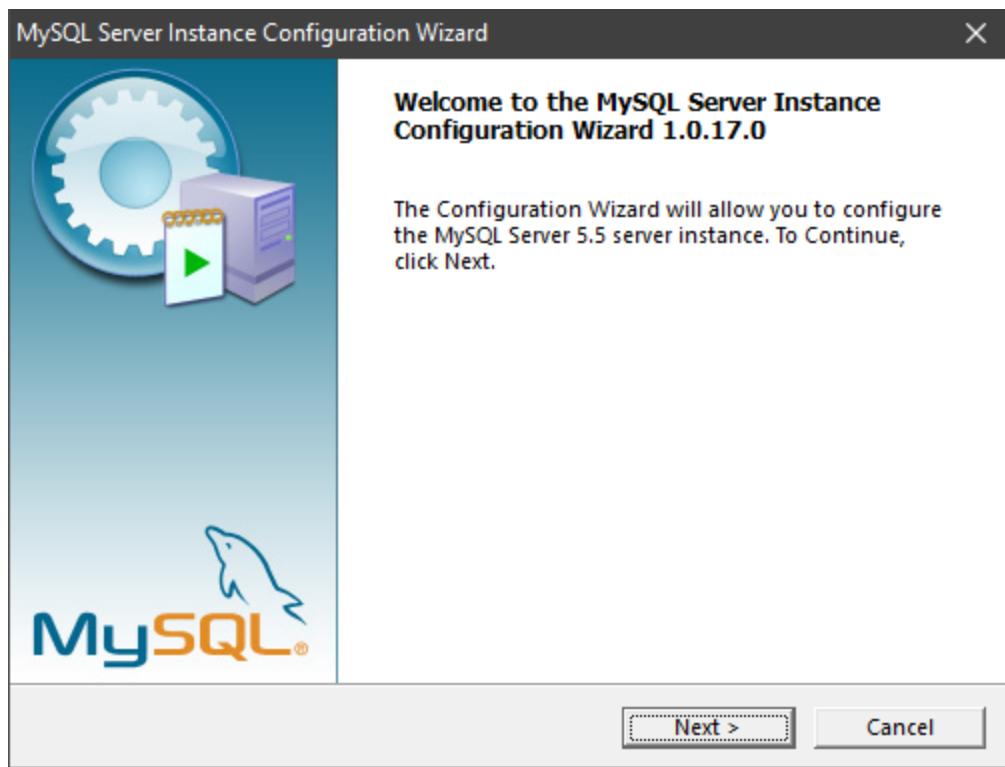
The screenshot shows a web browser window with the MySQL website. The URL is https://downloads.mysql.com/archives/community/. The page title is "MySQL :: Download MySQL Community Server". The main navigation menu includes MySQL.COM, DOWNLOADS (which is highlighted), DOCUMENTATION, and DEVELOPER ZONE. Below the menu, there's a secondary navigation bar with links for Enterprise, Community, Yum Repository, APT Repository, SUSE Repository, Windows, and Archives (which is also highlighted). On the left, a sidebar lists various MySQL products. The main content area is titled "MySQL Community Server (Archived Versions)". It contains a note: "Please note that these are old versions. New releases will have recent bug fixes and features! To download the latest release of MySQL Community Server, please visit MySQL Downloads." Below this, there are dropdown menus for Product Version (set to 5.5.50), Operating System (set to Microsoft Windows), and OS Version (set to Windows (x86, 64-bit)). A specific download link for the "MSI Installer" (mysql-5.5.50-winx64.msi) is shown, with details: May 17, 2016, 41.4M, and a "Download" button.

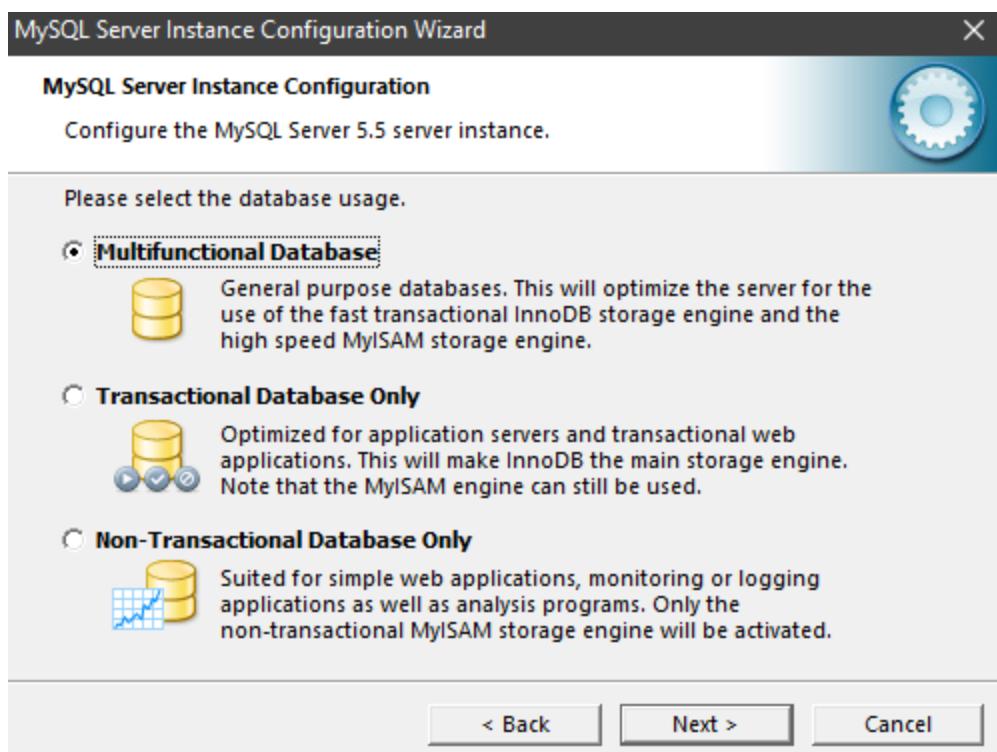
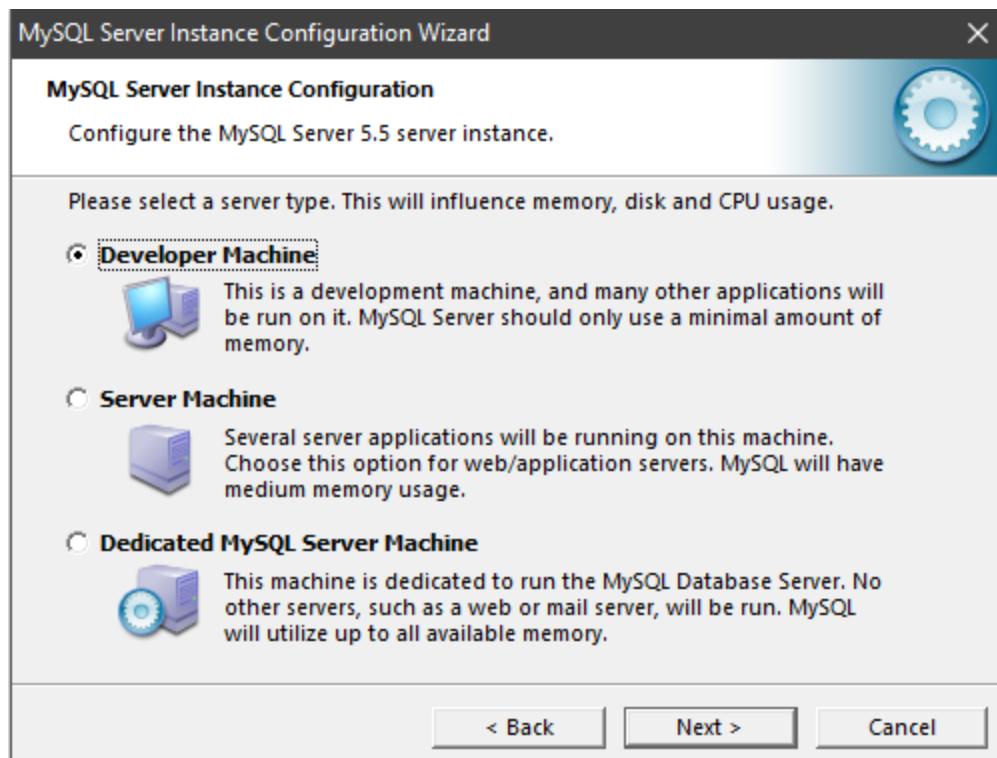
Download Installer.

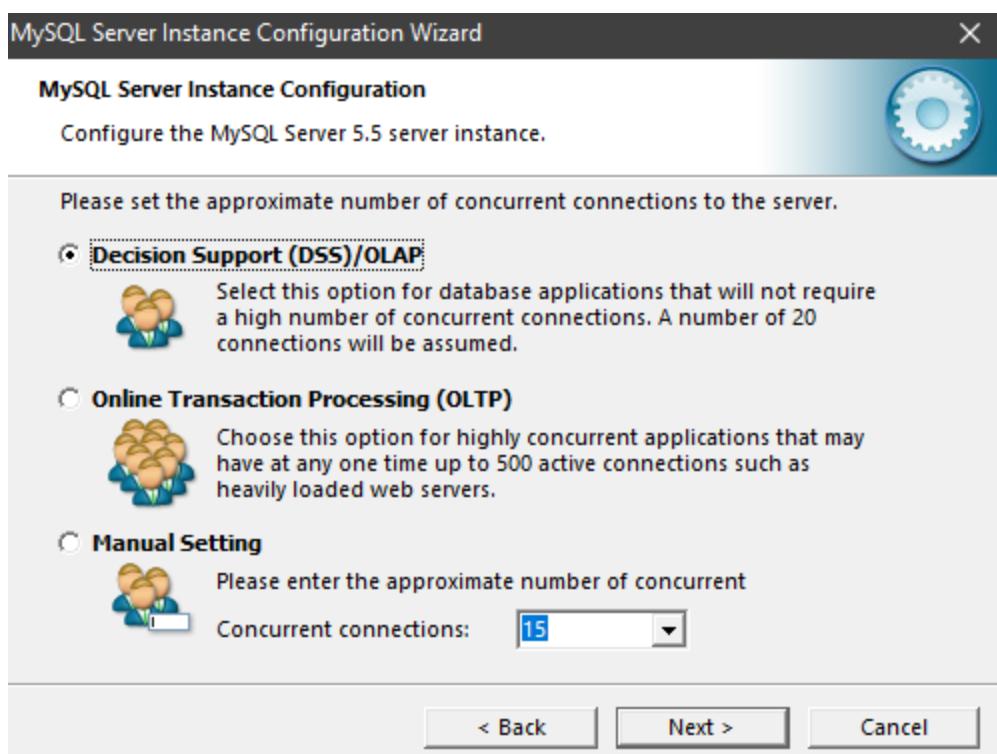
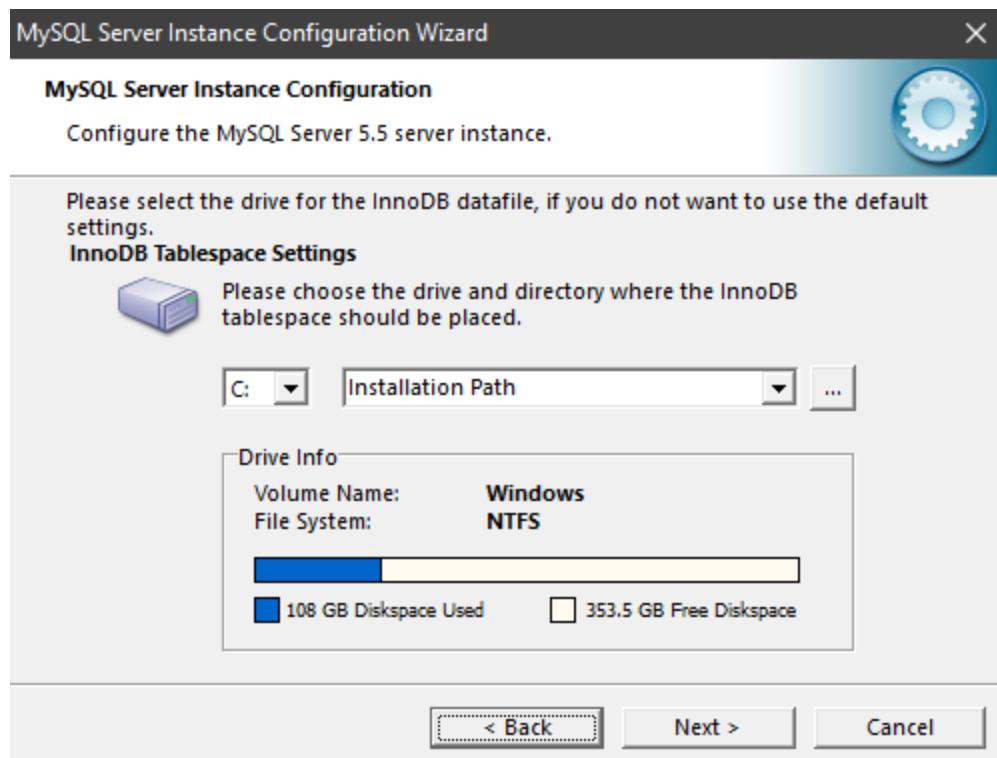


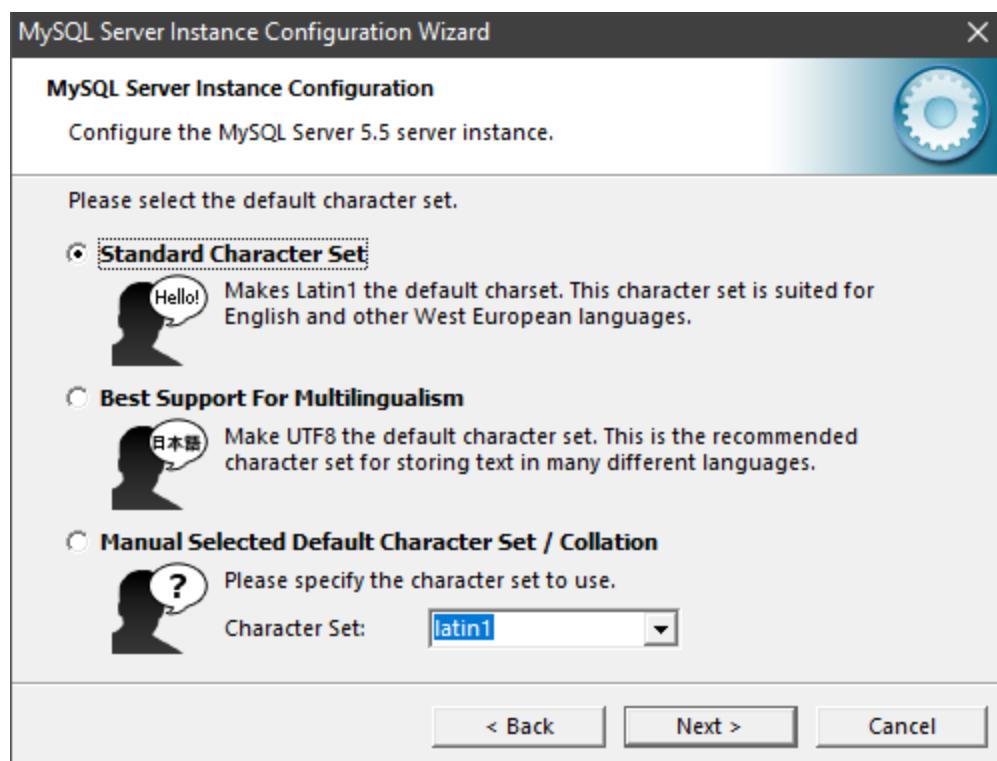
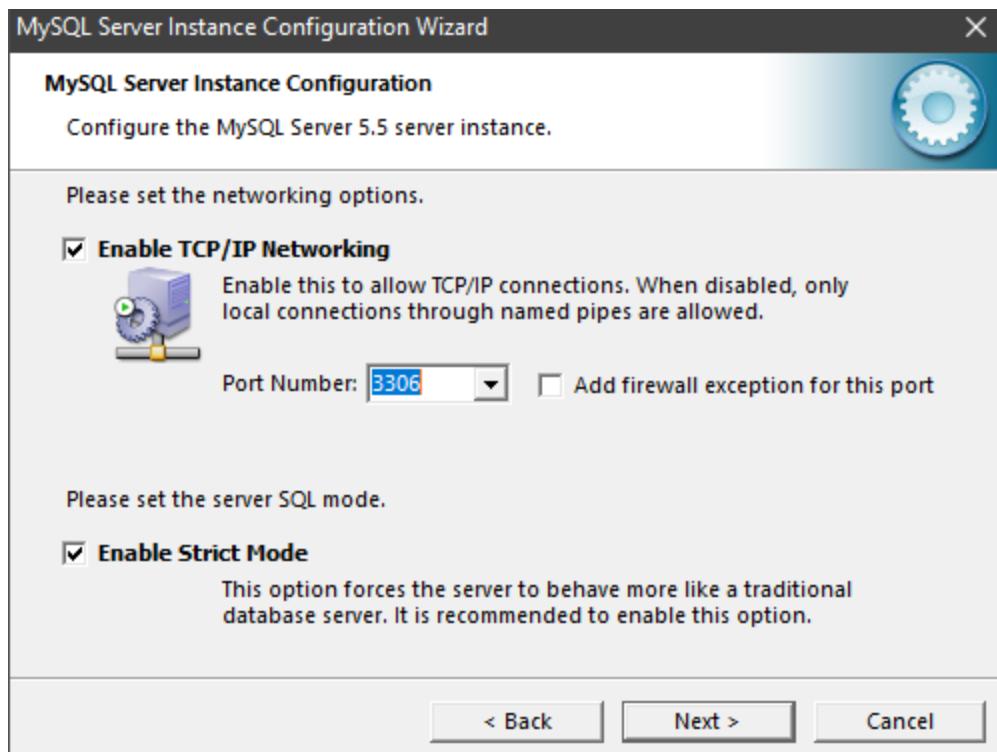
The screenshot shows a list of files downloaded today. There is one item: "mysql-5.5.50-winx64.msi" which was downloaded on 11/15/2022 at 9:46 AM. The file is a Windows Installer and is 42,443 KB in size. The "Download" button is visible next to the file name.

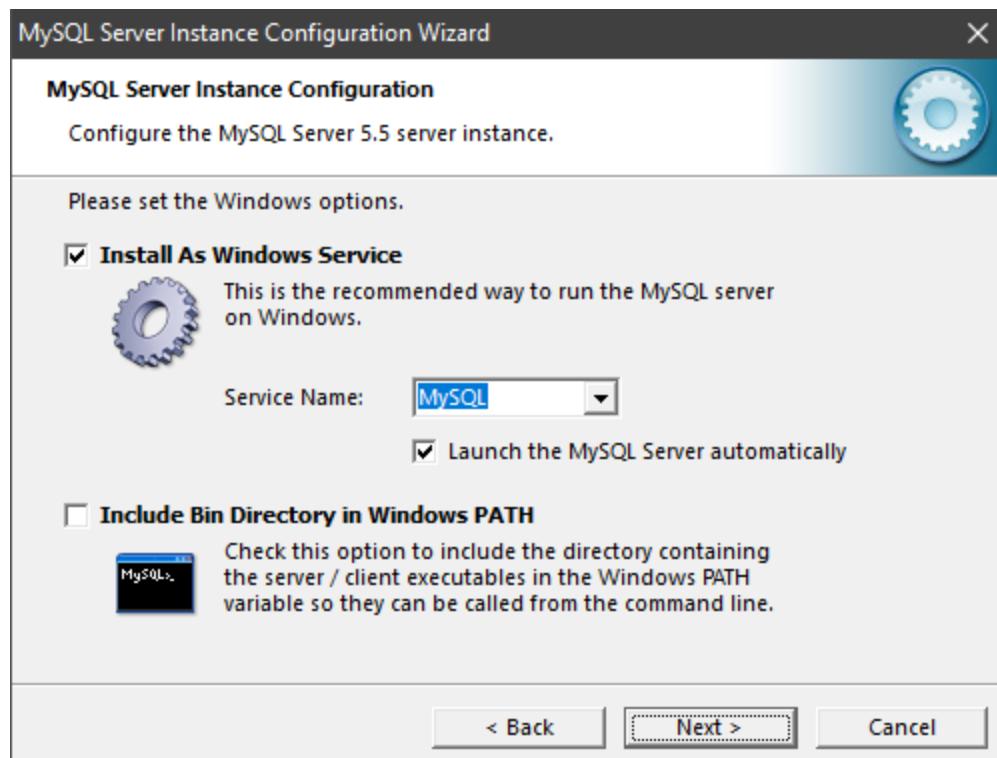
Double click on installer → Next → next..... → choose password and remember → next → finish.

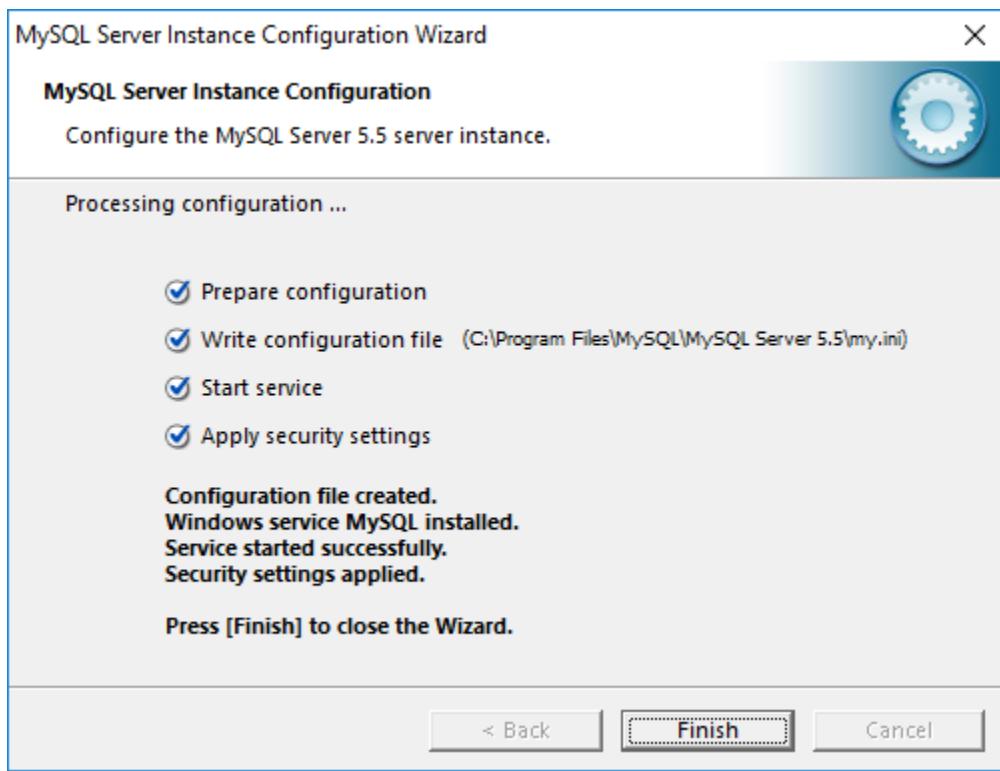
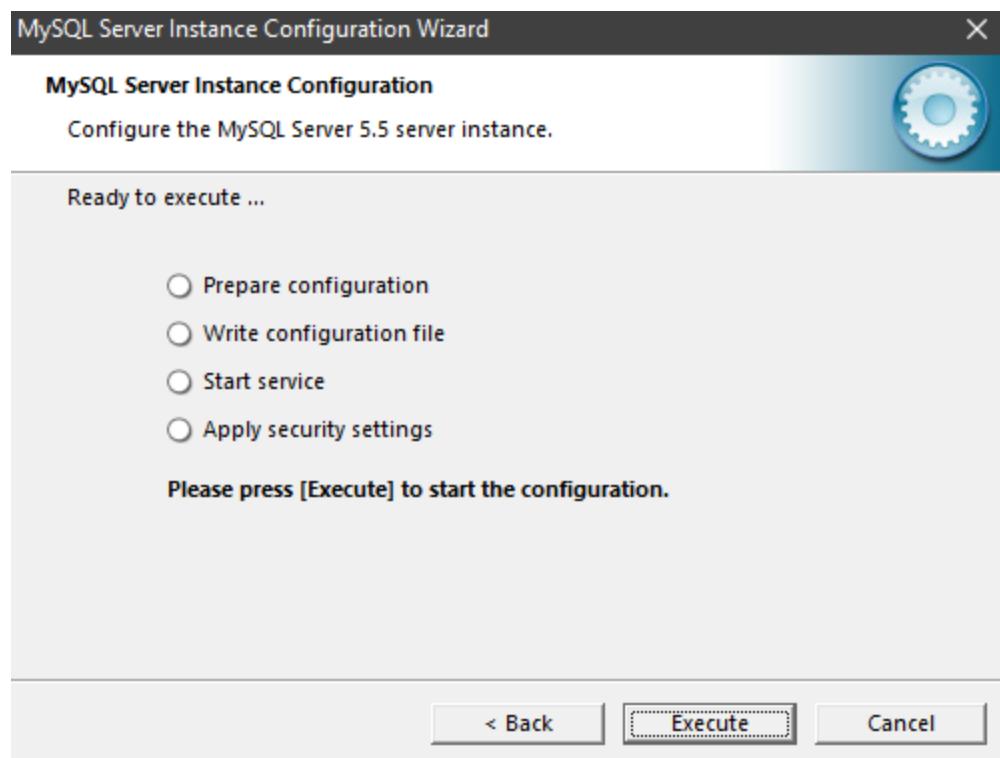






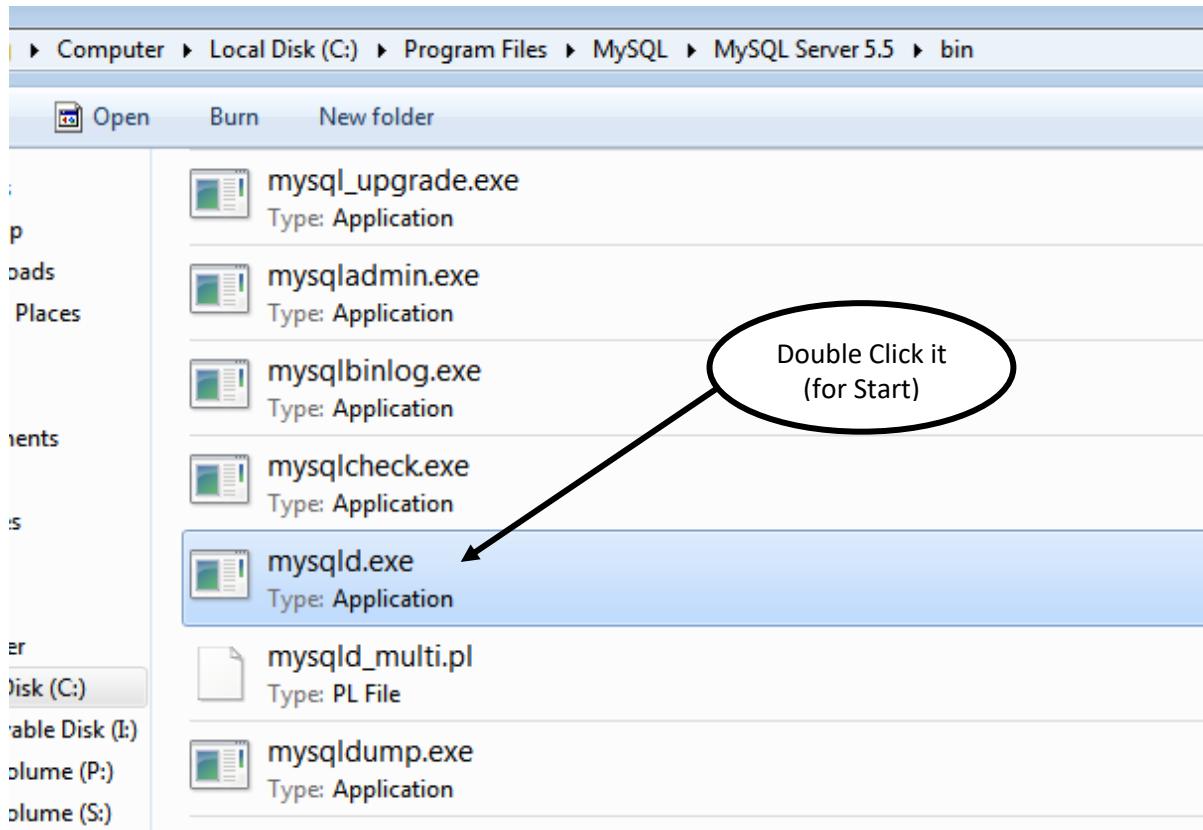






## Start MySQL Server

C:\Program Files\MySQL\MySQL Server 5.5\bin → Double click on → mysqld.exe



Check whether the server is started or not:

Go in Control panel → Administrative tools → Services → mysql service (automatic mode)

Services

File Action View Help

Services (Local)

MySQL

| Name                             | Description     | Status          | Startup Type     | Log On As           |
|----------------------------------|-----------------|-----------------|------------------|---------------------|
| Internet Connection Sharin...    | Provides ne...  | Manual (Trig... | Local System     |                     |
| IP Helper                        | Provides tu...  | Running         | Automatic        | Local System        |
| IPsec Policy Agent               | Internet Pro... |                 | Manual (Trig...  | Network Service     |
| KtmRm for Distributed Tran...    | Coordinates...  |                 | Manual (Trig...  | Network Service     |
| Link-Layer Topology Discov...    | Creates a N...  |                 | Manual           | Local Service       |
| Local Session Manager            | Core Windo...   | Running         | Automatic        | Local System        |
| MessagingService_3bb7254         | Service sup...  |                 | Manual (Trig...  | Local System        |
| Microsoft (R) Diagnostics H...   | Diagnostics ... |                 | Manual           | Local System        |
| Microsoft Account Sign-in ...    | Enables use...  |                 | Manual (Trig...  | Local System        |
| Microsoft App-V Client           | Manages A...    |                 | Disabled         | Local System        |
| Microsoft iSCSI Initiator Ser... | Manages In...   |                 | Manual           | Local System        |
| Microsoft Passport               | Provides pr...  |                 | Manual (Trig...  | Local System        |
| Microsoft Passport Container     | Manages lo...   |                 | Manual (Trig...  | Local Service       |
| Microsoft Software Shadow...     | Manages so...   |                 | Manual           | Local System        |
| Microsoft Storage Spaces S...    | Host service... |                 | Manual           | Network Service     |
| Microsoft Windows SMS Ro...      | Routes mes...   |                 | Manual (Trig...  | Local System        |
| MySQL Maintenance Service        | The Mozilla ... |                 | Manual           | Local System        |
| <b>MySQL</b>                     |                 | <b>Running</b>  | <b>Automatic</b> | <b>Local System</b> |
| Net.Tcp Port Sharing Service     | Provides abi... |                 | Disabled         | Local Service       |
| Netlogon                         | Maintains a ... | Running         | Automatic        | Local System        |
| Network Connected Device...      | Network Co...   |                 | Manual (Trig...  | Local Service       |
| Network Connection Broker        | Brokers con...  | Running         | Manual (Trig...  | Local System        |

Extended Standard

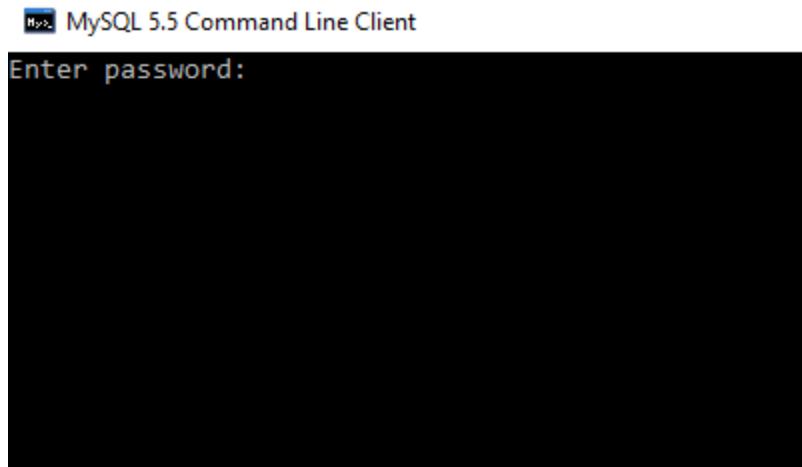
OR

Open Task manager using (Alt+Ctrl+delete) keys → Processes → see mysql.exe

### **Steps to create a table in MYSQL:**

1. Start the Server (double-click on mysqld.exe).
2. Start MySQL CLI client.
3. Enter the Password.
4. Create a Database.
5. Create a Table.

Double-click on the **MySQL CLI** icon to open MySQL Command Line Client:



**Enter password:**

If you have changed your password during installation, enter your new password; otherwise, the default password is the **root**.

```
MySQL 5.5 Command Line Client  
Enter password: *****  
Welcome to the MySQL monitor. Commands end with ; or \g.  
Your MySQL connection id is 3  
Server version: 5.5.50 MySQL Community Server (GPL)  
  
Copyright (c) 2000, 2016, Oracle and/or its affiliates. All rights reserved.  
  
Oracle is a registered trademark of Oracle Corporation and/or its  
affiliates. Other names may be trademarks of their respective  
owners.  
  
Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.  
mysql>
```

Run the following query on MySQL Command Line Client to show the list of databases:

```
mysql> show databases;
```

```
mysql> show databases;  
+-----+  
| Database |  
+-----+  
| information_schema |  
| mysql |  
| performance_schema |  
| test |  
+-----+  
4 rows in set (0.00 sec)  
  
mysql>
```

**Create your own Database:**

```
mysql> create database jdbc;
Query OK, 1 row affected (0.00 sec)

mysql> use jdbc;
Database changed
mysql>
```

**To create a table in your Database:**

```
create table emp1(empId int primary key auto_increment, name varchar(50), salary float(10,3));
```

```
mysql> create table emp1(empId int primary key auto_increment, name varchar(50),salary float(10,3));
Query OK, 0 rows affected (0.16 sec)
```

**To show tables:**

```
mysql> show tables;
+-----+
| Tables_in_jdbc |
+-----+
| emp           |
| emp1          |
+-----+
```

To insert a row in the table:

```
mysql> insert into emp(name,salary) value('deepak',75000.25);
Query OK, 1 row affected (0.14 sec)

mysql> insert into emp(name,salary) value('rohan',65000.000);
Query OK, 1 row affected (0.08 sec)

mysql> insert into emp(name,salary) value('aditi',87000.345);
Query OK, 1 row affected (0.08 sec)
```

To display table records:

```
select * from emp;
```

```
mysql> select * from emp;
+-----+-----+-----+
| empId | name   | salary   |
+-----+-----+-----+
|     1 | deepak | 75000.250 |
|     2 | rohan  | 65000.000 |
|     3 | aditi  | 87000.344 |
+-----+-----+-----+
3 rows in set (0.05 sec)
```

To display specific record/s:

```
mysql> select empId from emp;
+-----+
| empId |
+-----+
|     1 |
|     2 |
|     3 |
+-----+
3 rows in set (0.00 sec)

mysql> select empId, name from emp;
+-----+-----+
| empId | name   |
+-----+-----+
|     1 | deepak |
|     2 | rohan  |
|     3 | aditi   |
+-----+-----+
3 rows in set (0.00 sec)
```

```
mysql> select empId from emp where empId =2;
+-----+
| empId |
+-----+
|      2 |
+-----+
1 row in set (0.00 sec)

mysql> select empId,name from emp where empId=3;
+-----+-----+
| empId | name  |
+-----+-----+
|      3 | aditi |
+-----+-----+
1 row in set (0.00 sec)
```

Through the above steps, we have created our database(jdbc) and table(emp).

Now we will create our Java application (JDBC Program) to communicate (insert, delete, etc.) with the above database.

## **TestJDBC.java**

```
import java.sql.*;  
class TestJDBC{  
    public static void main(String[] args) throws Exception  
{  
    //1. Collect Database and Driver Info  
    String driverClassName = "com.mysql.jdbc.Driver";  
  
    String url="jdbc:mysql://localhost/jdbc";  
    String user="root";  
    String pwd= "saurabh"; //by default it is root  
  
    //2. Load JDBC Driver / Register a Driver  
    Class.forName(driverClassName).newInstance();  
  
    //3. Open a Connection  
    Connection con = DriverManager.getConnection(url,user,pwd);  
    System.out.println("con---->" + con);  
  
    //4. Create Statement and execute sql  
    Statement st = con.createStatement();  
    String sql= "INSERT INTO emp(name,salary)  
    VALUE('Aman',256856.05)";  
    st.executeUpdate(sql);  
  
    //5. close st and con : use finally block  
    st.close();  
    con.close();  
    System.out.println("---SQL executed successfully---");  
}  
}
```

**Save it:** D:\Programs\TestJDBC.java

**Compile it.** javac TestJDBC.java

**Execute it:** java TestJDBC---show the following run time error.

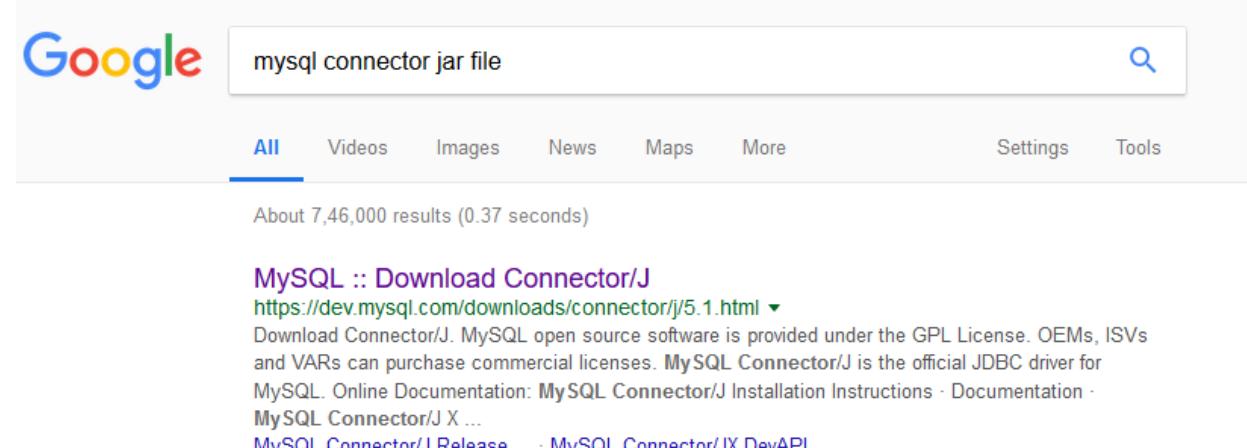
```
D:\1 Java\Programs>javac TestJDBC.java

D:\1 Java\Programs>java TestJDBC
Exception in thread "main" java.lang.ClassNotFoundException: com.mysql.jdbc.Driver
        at java.net.URLClassLoader.findClass(URLClassLoader.java:381)
        at java.lang.ClassLoader.loadClass(ClassLoader.java:424)
        at sun.misc.Launcher$AppClassLoader.loadClass(Launcher.java:335)
        at java.lang.ClassLoader.loadClass(ClassLoader.java:357)
        at java.lang.Class.forName0(Native Method)
        at java.lang.Class.forName(Class.java:264)
        at TestJDBC.main(TestJDBC.java:12)
```

**Problem:** This means that the JVM is not able to find the drivers, so we need to download the driver and save it to a particular location.

**Solution:**

1. Download mysql connector jar file



A screenshot of a Google search results page. The search query 'mysql connector jar file' is entered in the search bar. The results are filtered by 'All' and show approximately 7,46,000 results in 0.37 seconds. The top result is a link to MySQL's Connector/J download page, titled 'MySQL :: Download Connector/J' and linking to <https://dev.mysql.com/downloads/connector/j/5.1.html>. Below the search bar, there are navigation links for Videos, Images, News, Maps, More, Settings, and Tools.

**Connector/J 5.1.46**

Select Operating System:

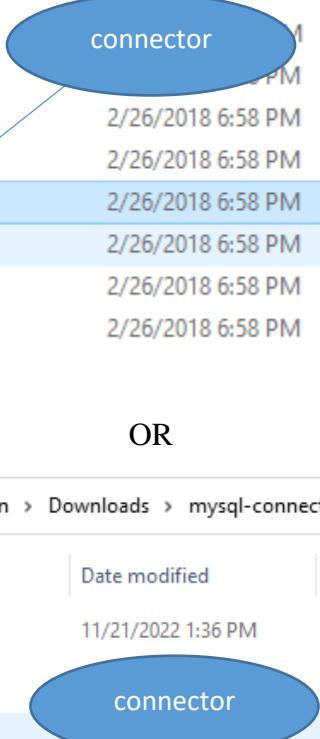
Platform Independent

| Platform Independent (Architecture Independent), Compressed TAR Archive<br>(mysql-connector-java-5.1.46.tar.gz) | 5.1.46 | 4.2M | Download |
|---|--------|------|----------|
| MD5: 6419f5a73c76991d73e27a93babca263   Signature   |        |      |          |

<https://downloads.mysql.com/archives/c-j/>

| Name                            | Date modified      | Type           | Size     |
|---------------------------------|--------------------|----------------|----------|
| ▼ Today (1)                     |                    |                |          |
| mysql-connector-java-5.1.46.tar | 11/21/2022 1:35 PM | WinRAR archive | 4,331 KB |

2. Extract the downloaded file:



|                                 |                     |              |
|---------------------------------|---------------------|--------------|
| src                             | connector           | File folder  |
| build                           | File                | XML Document |
| CHANGES                         | File                | 242 KB       |
| COPYING                         | File                | 18 KB        |
| mysql-connector-java-5.1.46     | Executable Jar File | 982 KB       |
| mysql-connector-java-5.1.46-bin | Executable Jar File | 982 KB       |
| README                          | File                | 60 KB        |
| README                          | Text Document       | 63 KB        |

OR

This PC > Local Disk (C:) > Users > saurabh.jain > Downloads > mysql-connector-java-5.1.46 > mysql-connector-java-5.1.46

| Name                            | Date modified      | Type                | Size   |
|---------------------------------|--------------------|---------------------|--------|
| src                             | 11/21/2022 1:36 PM | File folder         |        |
| build                           |                    | XML Document        | 90 KB  |
| CHANGES                         |                    | File                | 242 KB |
| COPYING                         | 2/26/2018 6:58 PM  | File                | 18 KB  |
| mysql-connector-java-5.1.46     | 2/26/2018 6:58 PM  | Executable Jar File | 982 KB |
| mysql-connector-java-5.1.46-bin | 2/26/2018 6:58 PM  | Executable Jar File | 982 KB |
| README                          | 2/26/2018 6:58 PM  | File                | 60 KB  |
| README                          | 2/26/2018 6:58 PM  | Text Document       | 63 KB  |

3. Copy this connector and paste it into the place where you saved the TestJDBC.java program
4. Set the classpath to the place where the connector is placed.

**Temporary:**

```
D:\Programs>SET CLASSPATH=D:\Programs\mysql-connector-java-5.1.46.jar;
```

**Permanent:**

Go to the environment variable, then click on the new tab. In variable name, write **classpath** and in variable value, paste the path to the mysqlconnector.jar file by appending mysqlconnector.jar;; as

```
D:\1 Java\Programs>SET CLASSPATH=D:\1 Java\Programs\mysql-connector-java-5.1.46.jar;

D:\1 Java\Programs>javac TestJDBC.java

D:\1 Java\Programs>java TestJDBC
con---->com.mysql.jdbc.JDBC4Connection@6267c3bb
---SQL executed successfully---
```

**Note:**

At the time of execution, it may show the following message:

```
E:\myjava500063451>java Jdbc_Connectivity
Loading class `com.mysql.jdbc.Driver'. This is deprecated. The new driver class is `com.mysql.cj.jdbc.Driver'. The driver is automatically registered via the SPI and manual loading of the driver class is generally unnecessary.
this is query is executed successfully re....
```

**Solution:**

The screenshot shows a Stack Overflow question page with the URL [stackoverflow.com/questions/52032739/loading-class-com-mysql-jdbc-driver-this-is-deprecated-the-new-driver-class](https://stackoverflow.com/questions/52032739/loading-class-com-mysql-jdbc-driver-this-is-deprecated-the-new-driver-class). The page title is "Loading class com.mysql.jdbc.Driver this is deprecated. The new driver class". The question has 27 upvotes and asks: "According to Changes in the Connector/J API, the name of the class that implements java.sql.Driver in MySQL Connector/J has changed from `com.mysql.jdbc.Driver` to `com.mysql.cj.jdbc.Driver`. The old class name has been deprecated." It also notes that you just need to change the name of the driver. Below the question, there is a code snippet showing the replacement:

```
Class.forName("com.mysql.jdbc.Driver");
to
Class.forName("com.mysql.cj.jdbc.Driver");
```

change the name of driverClassName:

```
public static void main(String[] args) throws Exception
{
    //1. Collect Database and Driver Info
    //String driverClassName = "com.mysql.jdbc.Driver";
    String driverClassName = "com.mysql.cj.jdbc.Driver";
    ...
}
```

After successfully executing the TestJDBC.java program, another row will be inserted into our Emp table.

```
mysql> select * from emp;
+-----+-----+-----+
| empId | name   | salary |
+-----+-----+-----+
| 1     | deepak | 75000.250 |
| 2     | rohan  | 65000.000 |
| 3     | aditi   | 87000.344 |
| 4     | Aman    | 256856.047 |
+-----+-----+-----+
4 rows in set (0.00 sec)
```

New row  
Inserted

## **Basic Components of JDBC Application:**

### **Driver: Java Calls ↔ DB Calls**

JAVA and Database are both different languages. To convert Java calls into DB calls and DB calls into Java calls, we require one translator; that translator is nothing but the Driver.

### **Connections: Network Socket**

**Statement Object:** It is responsible for sending the Query to the DB, and the DB engine must execute the SQL query and bring the results from the DB to the Java application.

**ResultSet:** It holds the results of a SQL query; Java applications can get the results from the ResultSet.

## **5 Steps to connect to the database in Java**

### **Java Database Connectivity**



There are five steps to connect any Java application with the database in Java using JDBC. They are as follows:

- Register for the driver's class
- Creating a connection
- Creating a statement
- Executing queries
- Closing connection

## **1) Register for the driver class**

The `forName()` method of class `Class` is used to register the driver class. This method is used to load the driver class dynamically.

### **Syntax of `forName()` method**

1. `public static void forName(String className) throws ClassNotFoundException`

### **Example**

1. `Class.forName("com.mysql.jdbc.Driver ");`

## **2) Create the connection object**

The `getConnection()` method of `DriverManager` class is used to establish a connection with the database.

### **Syntax of `getConnection()` method**

1. `public static Connection getConnection(String url) throws SQLException`
2. `public static Connection getConnection(String url, String user_name, String password) throws SQLException`

### **Example to establish a connection with the Oracle database**

```
Connection con = DriverManager.getConnection(url,user, pwd);
```

## **3) Create the Statement object**

The `createStatement()` method of `Connection` interface is used to create a statement. The object of the statement is responsible to execute queries with the database.

### **Syntax of `createStatement()` method**

1. `public Statement createStatement() throws SQLException`

### **Example to create the statement object**

1. `Statement stmt=con.createStatement();`

## **4) Execute the query**

The `executeQuery()` method of the `Statement` interface executes queries to the database. This method returns the object of `ResultSet`, which can be used to get all the table records.

## **Syntax of executeQuery() method**

1. public ResultSet executeQuery(String sql) throws SQLException

## **Example to execute query**

1. ResultSet rs=stmt.executeQuery("select \* from emp");
2. while(rs.next()){
3. System.out.println(rs.getInt(1)+" "+rs.getString(2));
4. }

## **5) Close the connection object**

By closing the connection object statement, the ResultSet will be closed automatically. The close() method of the Connection interface is used to close the connection.

## **Syntax of close() method**

1. public void close()throws SQLException

## **Example to close connection**

1. con.close();

### Program: TestJDBC1.java

// We can insert another row in the database table using the following Java program.

```
import java.sql.*;
class TestJDBC1 {
    public static void main(String[] args) throws Exception {
        Class.forName("com.mysql.jdbc.Driver").newInstance();
        Connection con =
        DriverManager.getConnection("jdbc:mysql://localhost:3306/jdbc?user=root&password=saurabh");
        Statement st = con.createStatement();
        String sql = "INSERT INTO emp(name,salary) VALUE('Sumit',32568.05)";
        st.executeUpdate(sql);
        st.close();
        con.close();
        System.out.println("---SQL executed successfully---");
    }
}
```

**OR**

```
import java.sql.*;
class TestJDBC1 {
    public static void main(String[] args) throws Exception {
```

**/\* 1) Register the driver class:** The forName() method of Class class is used to register the driver class. This method is used to dynamically load the driver class. The driver class for the mysql database is com.mysql.jdbc.Driver.\*/

```
Class.forName("com.mysql.jdbc.Driver").newInstance();
```

**/\* 2) Create the connection object:** The getConnection() method of DriverManager class is used to establish a connection with the database. The connection URL for the mysql database is **jdbc:mysql://localhost:3306/jdbc** where **jdbc** is the API, mysql is the database, localhost is the server name on which mysql is running; we may also use IP address, 3306 is the port number, and **jdbc** is the database name. We may use any database; in such case, we need to replace the **jdbc** with our database name.

**Username:** The default username for the mysql database is **root**.

**Password:** It is the password given by the user at the time of installing the mysql database. In this example, we are going to use **saurabh** as the password.\*/

```
Connection con =
DriverManager.getConnection("jdbc:mysql://localhost:3306/jdbc?user=root&password =
saurabh");
```

**/\* 3) Create the Statement object:** The createStatement() method of Connection interface is used to create statement. The object of the statement is responsible to execute queries with the database.  
public Statement createStatement(): creates a statement object that can be used to execute SQL queries.\*/

```
Statement st = con.createStatement();
```

**/\* 4) Execute the query:** The executeUpdate() method of the Statement interface is used to execute queries to the database. It is used to execute the specified query; it may be create, drop, insert, update, delete etc.\*/

```
String sql = "INSERT INTO emp(name,salary) VALUE('Sumit',32568.05);  
st.executeUpdate(sql);
```

**/\* 5) Close the connection object:** By closing the connection object statement will be closed automatically. The close() method of the Connection interface is used to close the connection.\*/

```
st.close();  
con.close();  
System.out.println("---SQL executed successfully---");  
}  
}
```

```
D:\1 Java\Programs>javac TestJDBC1.java  
  
D:\1 Java\Programs>java TestJDBC1  
---SQL executed successfully---
```

```
mysql> select * from emp;  
+-----+-----+-----+  
| empId | name   | salary  |  
+-----+-----+-----+  
|      1 | deepak | 75000.250 |  
|      2 | rohan  | 65000.000 |  
|      3 | aditi   | 87000.344 |  
|      4 | Aman    | 256856.047 |  
|      5 | Sumit   | 32568.051 |  
+-----+-----+-----+  
5 rows in set (0.00 sec)
```

### Program: TestDelete.java

// We can delete a row from the database table using the following Java program.

```
import java.sql.*;
class TestDelete {
    public static void main(String[] args) throws Exception {
        Class.forName("com.mysql.jdbc.Driver").newInstance();
        Connection con =
DriverManager.getConnection("jdbc:mysql://localhost:3306/jdbc?user=root&password=saurabh");
        Statement st = con.createStatement();
        String sql = "DELETE FROM emp where empId=1";
        st.executeUpdate(sql);
        st.close();
        con.close();
        System.out.println("---SQL executed successfully---");
    }
}
```

```
D:\1 Java\Programs>javac TestDelete.java

D:\1 Java\Programs>java TestDelete
---SQL executed successfully---
```

```
mysql> select * from emp;
+-----+-----+-----+
| empId | name  | salary   |
+-----+-----+-----+
|      2 | rohan | 65000.000 |
|      3 | aditi  | 87000.344 |
|      4 | Aman   | 256856.047|
|      5 | Sumit  | 32568.051 |
+-----+-----+-----+
4 rows in set (0.00 sec)
```

### **Program: TestUpdate.java**

**//The following Java program can update the record in the database table.**

```
import java.sql.*;
class TestUpdate {
    public static void main(String[] args) throws Exception {
        Class.forName("com.mysql.jdbc.Driver").newInstance();
        Connection con =
DriverManager.getConnection("jdbc:mysql://localhost:3306/jdbc?user=root&password=saurabh");
        Statement st = con.createStatement();
        String sql = "UPDATE emp SET name='Raj', salary=50000 where empId=2";
        st.executeUpdate(sql);
        st.close();
        con.close();
        System.out.println("---SQL executed successfully---");
    }
}
```

```
D:\1 Java\Programs>javac TestUpdate.java
```

```
D:\1 Java\Programs>java TestUpdate
---SQL executed successfully---
```

MySQL 5.5 Command Line Client

```
mysql> select * from emp;
```

| empId | name  | salary     |
|-------|-------|------------|
| 2     | rohan | 65000.000  |
| 3     | aditi | 87000.344  |
| 4     | Aman  | 256856.047 |
| 5     | Sumit | 32568.051  |

Before  
update

```
4 rows in set (0.00 sec)
```

```
mysql> select * from emp;
```

| empId | name  | salary     |
|-------|-------|------------|
| 2     | Raj   | 50000.000  |
| 3     | aditi | 87000.344  |
| 4     | Aman  | 256856.047 |
| 5     | Sumit | 32568.051  |

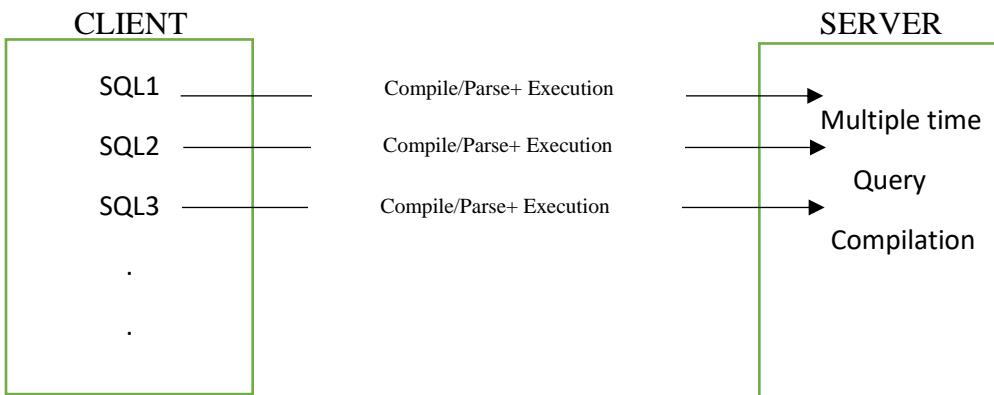
After  
update

```
4 rows in set (0.00 sec)
```

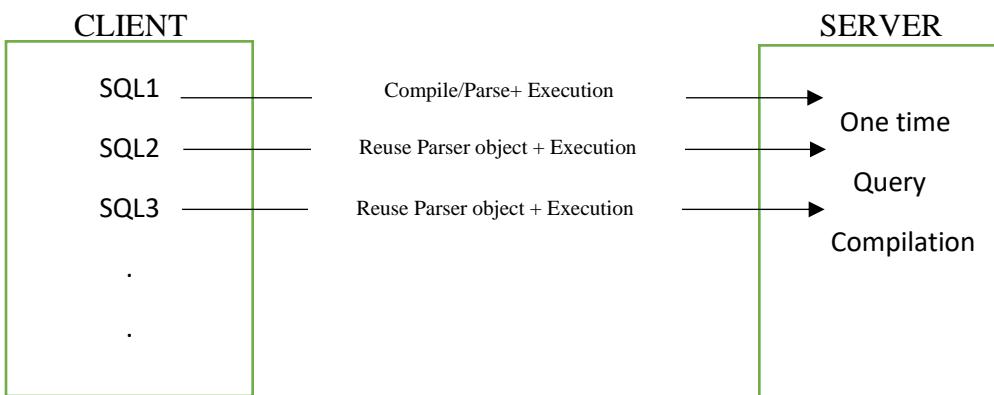
## PreparedStatement Interface

- The PreparedStatement interface is a sub-interface of Statement.
- It is used to execute a parameterized query (? Parameter).
- It handles SQL Injection Problems.
- Reuse parser object for similar queries
- Faster than Statement
- The application's performance will be faster using the PreparedStatement interface because the query is compiled only once.

Ex: String sql="insert into emp values(?, ?, ?);



**FIG: USING STATEMENT INTERFACE**



**FIG: USING PREPAREMENT STATEMENT INTERFACE**

If SQL1, SQL2, SQL3... are same queries with different data. So, it doesn't need to be parsed/compiled multiple times; it needs to be parsed/compiled only once with different data/datasets.

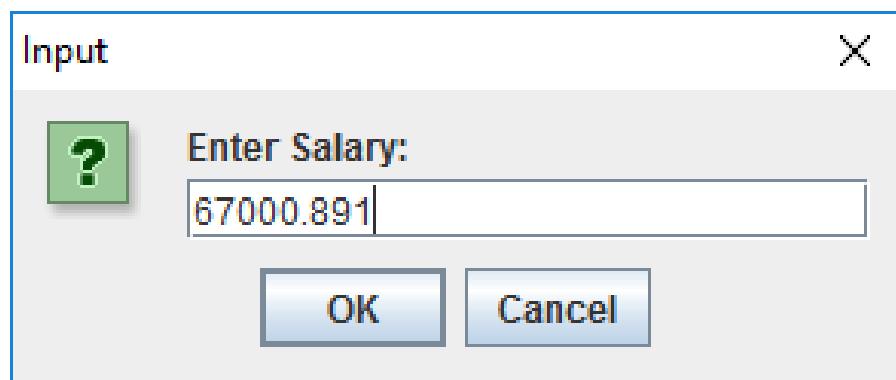
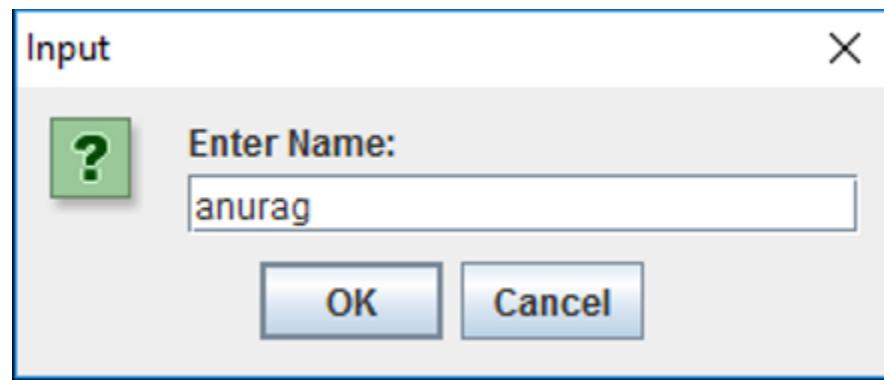
## DynamicInsertPST.java

This program takes input (name and salary) from the user at runtime using a dialog box (JOptionPane) and inserts it into a MySQL database table named emp using PreparedStatement.

```
import java.sql.*;
import javax.swing.JOptionPane;
class DynamicInsertPST {
    public static void main(String[] args) throws Exception {
        Class.forName("com.mysql.jdbc.Driver").newInstance();
        Connection con =
DriverManager.getConnection("jdbc:mysql://localhost:3306/jdbc?user=root&password=saurabh");
        String n=JOptionPane.showInputDialog("Enter Name:");
        String s=JOptionPane.showInputDialog("Enter Salary:");
        float fs=Float.parseFloat(s);

        // ? : place holder or parameter (parameterized query)
        String sql = "INSERT INTO emp(name,salary) VALUE(?,?)";
        PreparedStatement st = con.prepareStatement(sql);
        //bind data in PST
        st.setString(1, n);// 1 specifies the first parameter in the query
        st.setFloat(2, fs); // 2 specifies the second parameter in the query
        st.executeUpdate(); //no arguments
        st.close();
        con.close();
        System.out.println("---SQL executed successfully---");
    }
}
```

```
D:\1 Java\Programs>java DynamicInsertPST
---SQL executed successfully---
```



```
mysql> select * from emp;
+-----+-----+-----+
| empId | name  | salary   |
+-----+-----+-----+
|    2  | Raj    | 50000.000 |
|    3  | aditi  | 87000.344 |
|    4  | Aman   | 256856.047 |
|    5  | Sumit  | 32568.051  |
|    6  | anurag | 67000.891  |
+-----+-----+-----+
5 rows in set (0.06 sec)
```

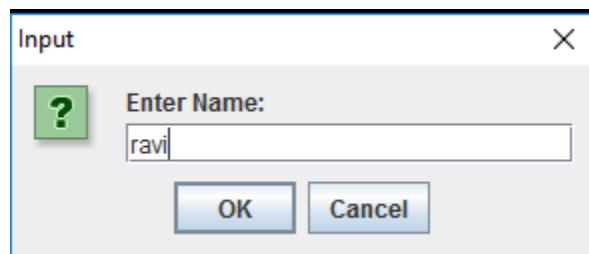
## UpdatePST.java

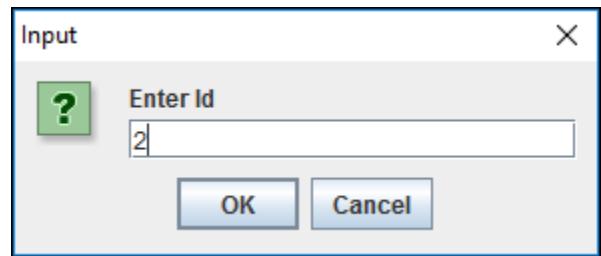
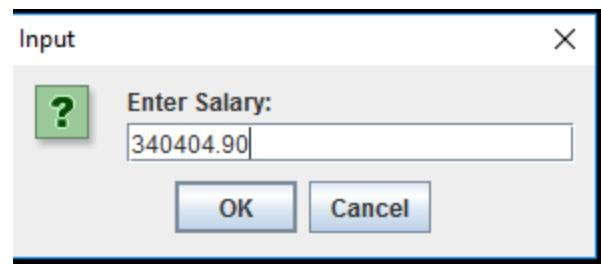
```
import java.sql.*;
import javax.swing.JOptionPane;
class UpdatePST {
    public static void main(String[] args) throws Exception {
        Class.forName("com.mysql.jdbc.Driver").newInstance();
        Connection con =
DriverManager.getConnection("jdbc:mysql://localhost:3306/jdbc?user=root&password=saurabh");
        String n=JOptionPane.showInputDialog("Enter Name:");
        String s=JOptionPane.showInputDialog("Enter Salary:");
        float fs=Float.parseFloat(s);
        String sid=JOptionPane.showInputDialog("Enter Id");
        int id =Integer.parseInt(sid);
        String sql = "UPDATE emp SET name=? , salary=? WHERE empId=?";
        PreparedStatement st = con.prepareStatement(sql);
        st.setString(1, n);
        st.setFloat(2, fs);
        st.setInt(3, id); //PK
        st.executeUpdate();
        st.close();
        con.close();
        System.out.println("---SQL executed successfully---");
    }
}
```

---

```
C:\Windows\System32\cmd.exe - java UpdatePST
D:\1 Java\Programs>javac UpdatePST.java

D:\1 Java\Programs>SET CLASSPATH=D:\1 Java\Programs\mysql-connector-java-5.1.46.jar;
D:\1 Java\Programs>java UpdatePST
```





```
mysql> select * from emp;
+-----+-----+-----+
| empId | name   | salary |
+-----+-----+-----+
|      2 | Raj    | 50000.000 |
|      3 | aditi  | 87000.344 |
|      4 | Aman   | 256856.047 |
|      5 | Sumit  | 32568.051  |
|      6 | anurag | 67000.891  |
+-----+-----+-----+
5 rows in set (0.00 sec)
```

```
mysql> select * from emp;
+-----+-----+-----+
| empId | name   | salary |
+-----+-----+-----+
|      2 | ravi   | 340404.906 |
|      3 | aditi  | 87000.344 |
|      4 | Aman   | 256856.047 |
|      5 | Sumit  | 32568.051  |
|      6 | anurag | 67000.891  |
+-----+-----+-----+
5 rows in set (0.00 sec)
```

## ResultSet Interface

In Java, the ResultSet is the Object that holds the result of a database query, typically the SQL select statement. It is also part of the JDBC API used to interact with relational databases. The ResultSet allows us to traverse the rows of tables returned by the SQL query and extract a specific column from the SQL query result.

The ResultSet is essentially a table of data where each row represents a record, and each column represents a field in the database. The ResultSet has a cursor that points to the current row in the ResultSet and we can be able to navigate in ResultSet by using the **next()**, **previous()**, **first()**, and **last()** methods. We can retrieve data by using different methods like **getString()**, **getInt()**, **getDouble()** and other methods.

The object of ResultSet maintains a cursor pointing to a table row. Initially, the cursor points to before the first row.

**The program fetches and displays employee records (empId, name, salary) from the emp table, where the employee ID (empId) is less than or equal to a given value (in this case, 6).**

```
import java.sql.*;
class RS {
    public static void main(String[] args) throws Exception {
        Class.forName("com.mysql.jdbc.Driver").newInstance();
        Connection con =
DriverManager.getConnection("jdbc:mysql://localhost:3306/jdbc?user=root&password=saurabh");
        String sql = "SELECT empId, name ,salary FROM emp WHERE empId <= ?";
        PreparedStatement st = con.prepareStatement(sql);
        // st.setInt(1, 3);//prints data where empId <= 3
        st.setInt(1, 6);/// prints data where empId <= 6
    }
}
```

ResultSet rs = st.executeQuery();//Executes the query and stores the **result set** (records) in rs.

```
while (rs.next()) {
    System.out.println(rs.getInt("empId")+" "+rs.getString("name")+
"+rs.getFloat("salary"));
}
rs.close();
st.close();
con.close();
System.out.println("---SQL executed successfully---");
}
}
```

```
D:\1 Java\Programs>javac RS.java

D:\1 Java\Programs>java RS
2 ravi 340404.9
3 aditi 87000.34
4 Aman 256856.05
5 Sumit 32568.05
6 anurag 67000.89
---SQL executed successfully---

D:\1 Java\Programs>javac RS.java

D:\1 Java\Programs>java RS
2 ravi 340404.9
3 aditi 87000.34
---SQL executed successfully---
```

A list of popular *interfaces* of JDBC API is given below:

- Driver interface
- Connection interface
- Statement interface
- PreparedStatement interface
- CallableStatement interface
- ResultSet interface
- ResultSetMetaData interface
- DatabaseMetaData interface
- RowSet interface

**A list of popular classes of JDBC API is given below:**

- DriverManager class
- Blob class
- Clob class
- Types class

**Assignment Topics:**

- CallableStatement interface
- RowSet interface

## **Wrapper Classes in Java**

In Java, wrapper classes allow primitive data types (like int, char, boolean) to be used as objects. They provide a way to wrap primitive values into object types and unwrap them back into primitives.

Since Java 5 (J2SE 5.0), the introduction of autoboxing and unboxing automates this conversion process:

- **Autoboxing:** Automatically converts a primitive to its wrapper object.
- **Unboxing:** Automatically converts a wrapper object back to its primitive.

## **Advantages of Wrapper Classes**

Wrapper classes offer several benefits over primitives:

- Used in Collections (which only support objects).
- Enable method usage like compareTo(), equals(), toString(), etc.
- Support cloning, serialization, and null values.
- Thread-safe and immutable, ideal for concurrent programming.
- Facilitate type conversion and utility operations.

## **Use of Wrapper Classes in Java**

Java is an object-oriented language, meaning many features, such as collections, serialization, and synchronization, require working with objects rather than primitive types. Wrapper classes help bridge this gap by converting primitives into their corresponding objects.

## **1. Changing the Value in a Method**

Java uses call-by-value, so when a primitive is passed to a method, the original value does not change. However, we can mimic pass-by-reference behavior using wrapper objects (which are passed as references).

## **2. Serialization**

Serialization converts objects into byte streams. Since primitives cannot be directly serialized, wrapper classes are used to wrap them as objects.

## **3. Synchronization**

Java synchronization works only with objects. When using thread synchronization, primitive types must be converted to wrapper objects.

## **4. Use in `java.util` Package**

The `java.util` package provides utility classes like `ArrayList`, `HashMap`, and others that work only with objects. Hence, wrapper classes are required when storing primitives.

## **5. Java Collection Framework**

All data structures in the Java Collections Framework (like `ArrayList`, `HashSet`, `TreeSet`, etc.) can store only objects, not primitives. Wrapper classes are essential for primitives to be used in these collections.

## **Wrapper Classes List**

The eight classes of the `java.lang` package are known as wrapper classes in Java. The list of eight wrapper classes is given below:

| <b>Primitive Type</b> | <b>Wrapper Class</b>   |
|-----------------------|------------------------|
| <code>boolean</code>  | <code>Boolean</code>   |
| <code>char</code>     | <code>Character</code> |
| <code>byte</code>     | <code>Byte</code>      |
| <code>short</code>    | <code>Short</code>     |
| <code>int</code>      | <code>Integer</code>   |
| <code>long</code>     | <code>Long</code>      |
| <code>float</code>    | <code>Float</code>     |
| <code>double</code>   | <code>Double</code>    |

## **Autoboxing**

Autoboxing is the automatic conversion of a primitive type to its corresponding wrapper class.

## **Example: Primitive to Wrapper**

```
public class WrapperExample1 {  
    public static void main(String[] args) {  
        int a = 20;
```

```
Integer i = Integer.valueOf(a); // Manual boxing
Integer j = a; // Autoboxing

System.out.println(a + " " + i + " " + j);

}
```

### **Output:**

20 20 20

## **Unboxing**

Unboxing is the automatic conversion of a wrapper object to its primitive type.

### **Example: Wrapper to Primitive**

```
public class WrapperExample2 {
    public static void main(String[] args) {
        Integer a = new Integer(3);
        int i = a.intValue(); // Manual unboxing
        int j = a; // Unboxing
        System.out.println(a + " " + i + " " + j);
    }
}
```

### **Output:**

3 3 3

## Complete Example: All Types

```
public class WrapperExample3 {  
    public static void main(String[] args) {  
        byte b = 10;  
        short s = 20;  
        int i = 30;  
        long l = 40L;  
        float f = 50.0f;  
        double d = 60.0;  
        char c = 'a';  
        boolean b2 = true;  
  
        // Autoboxing  
        Byte byteObj = b;  
        Short shortObj = s;  
        Integer intObj = i;  
        Long longObj = l;  
        Float floatObj = f;  
        Double doubleObj = d;  
        Character charObj = c;  
        Boolean boolObj = b2;  
  
        System.out.println("--- Printing Object Values ---");
```

```
System.out.println("Byte: " + byteObj);
System.out.println("Short: " + shortObj);
System.out.println("Integer: " + intObj);
System.out.println("Long: " + longObj);
System.out.println("Float: " + floatObj);
System.out.println("Double: " + doubleObj);
System.out.println("Character: " + charObj);
System.out.println("Boolean: " + boolObj);
```

```
// Unboxing
byte byteVal = byteObj;
short shortVal = shortObj;
int intVal = intObj;
long longVal = longObj;
float floatVal = floatObj;
double doubleVal = doubleObj;
char charVal = charObj;
boolean boolVal = boolObj;
```

```
System.out.println("--- Printing Primitive Values ---");
System.out.println("byte: " + byteVal);
System.out.println("short: " + shortVal);
System.out.println("int: " + intVal);
```

```
        System.out.println("long: " + longVal);
        System.out.println("float: " + floatVal);
        System.out.println("double: " + doubleVal);
        System.out.println("char: " + charVal);
        System.out.println("boolean: " + boolVal);
    }
}
```

### **Output:**

--- Printing Object Values ---

Byte: 10

Short: 20

Integer: 30

Long: 40

Float: 50.0

Double: 60.0

Character: a

Boolean: true

--- Printing Primitive Values ---

byte: 10

short: 20

int: 30

long: 40

float: 50.0

double: 60.0

char: a

boolean: true

## Java Swing — Introduction

Swing is a part of Java Foundation Classes (JFC).

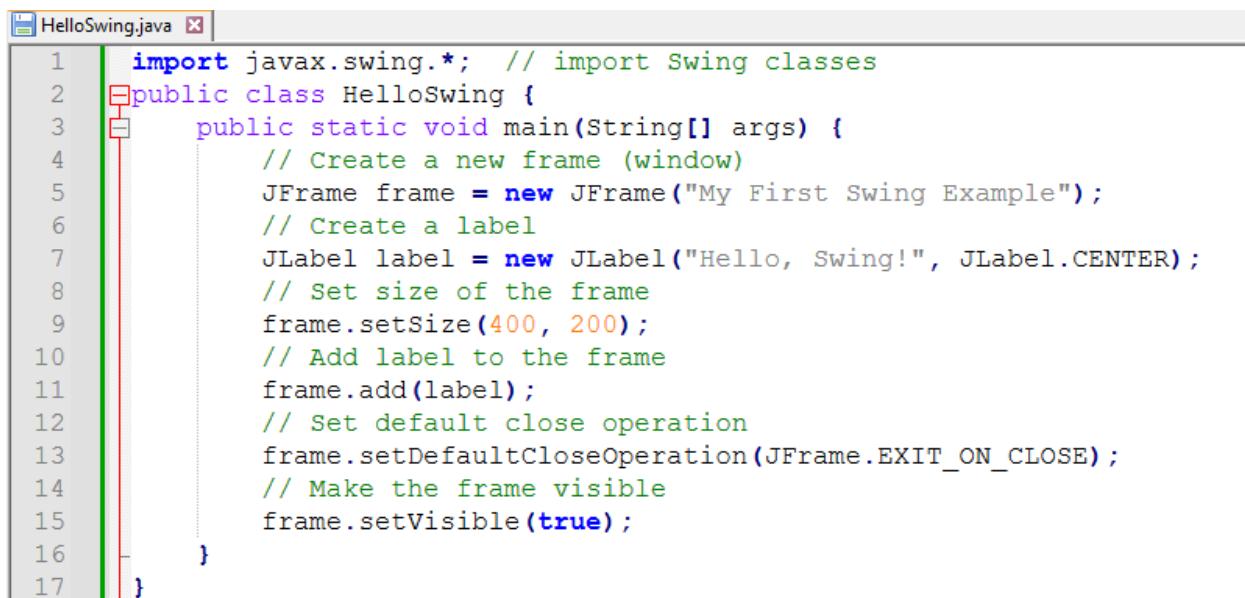
It is used to create Graphical User Interface (GUI) applications in Java.

- Lightweight components.
- Pluggable Look and Feel
- Platform-independent.
- More powerful than the old AWT (Abstract Window Toolkit).

## Basic Concepts

- JFrame → Main window (like a blank page).
- JButton → A clickable button.
- JLabel → Display text.
- JTextField → Single-line text box.
- JTextArea → Multi-line text area.

## First Simple Example: "Hello Swing"

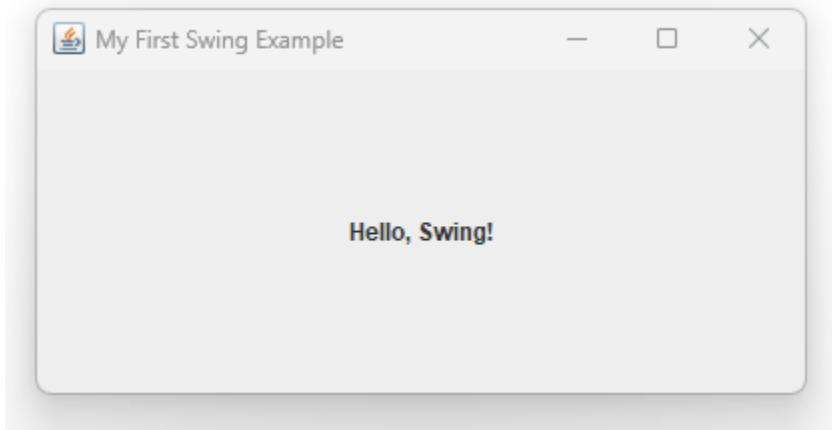


```
1 import javax.swing.*; // import Swing classes
2 public class HelloSwing {
3     public static void main(String[] args) {
4         // Create a new frame (window)
5         JFrame frame = new JFrame("My First Swing Example");
6         // Create a label
7         JLabel label = new JLabel("Hello, Swing!", JLabel.CENTER);
8         // Set size of the frame
9         frame.setSize(400, 200);
10        // Add label to the frame
11        frame.add(label);
12        // Set default close operation
13        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
14        // Make the frame visible
15        frame.setVisible(true);
16    }
17 }
```

**Output:**

```
D:\Java Code\swing>javac HelloSwing.java
```

```
D:\Java Code\swing>java HelloSwing
```



**Explanation**

- JFrame creates a window.
- setSize(width, height) sets the window size.
- add(component) adds a label (or button) inside the frame.
- setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE) ensures the program closes when you click X.
- setVisible(true) shows the window.

## **Adding Buttons and Action Listeners in Swing**

In Swing, when you click a button, you can make something happen, like showing a message!

### **Important things to use:**

- JButton → To create a button.
- ActionListener → To listen when the button is clicked.
- addActionListener() → To connect the button to an action.

### **setBounds() method**

In Java Swing, the setBounds() method is used to set the position and size of a component (like a button, label, text field, etc.) on a container (like a JFrame or JPanel).

```
component.setBounds(int x, int y, int width, int height);
```

Parameters it takes:

- x – the x-coordinate (horizontal position) of the component's top-left corner.
- y – the y-coordinate (vertical position) of the component's top-left corner.
- width – the width of the component.
- height – the height of the component.

### **Example:**

```
JButton button = new JButton("Click Me");
```

```
button.setBounds(100, 50, 120, 30);
```

```
frame.add(button);
```

- Here, the button will be placed 100 pixels from the left edge and 50 pixels from the top edge of the container.
- The button's size will be 120 pixels wide and 30 pixels tall.

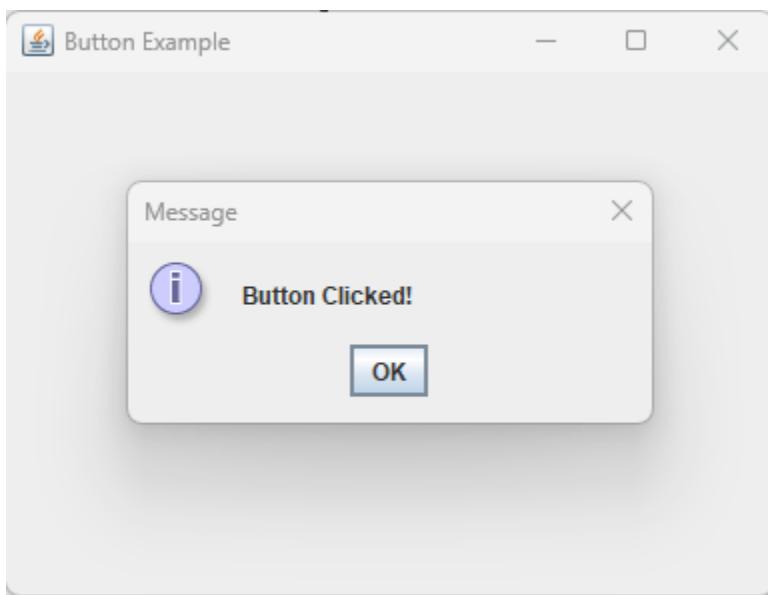
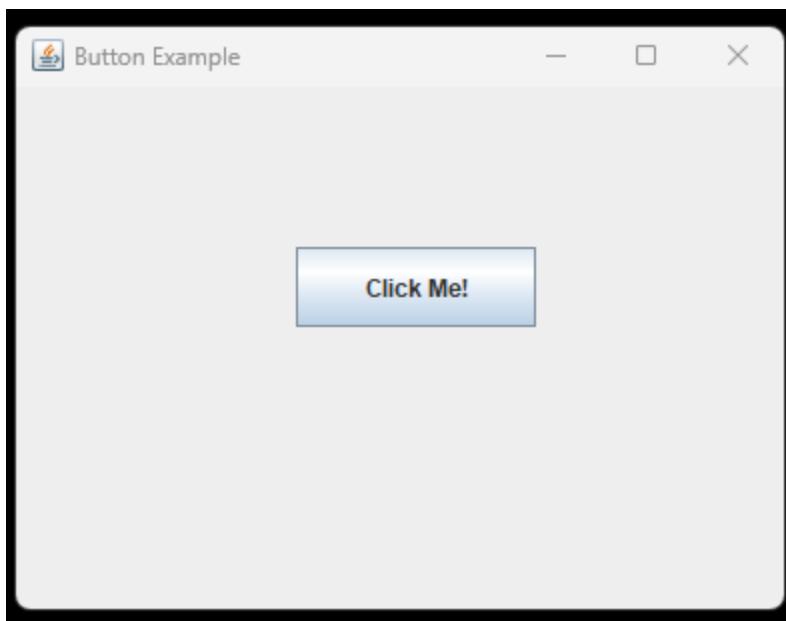
## Example: Button with ActionListener

```
1 import javax.swing.*;
2 import java.awt.event.*; // Needed for ActionListener
3 public class ButtonExample {
4     public static void main(String[] args) {
5         // Create a new frame
6         JFrame frame = new JFrame("Button Example");
7         // Create a new button
8         JButton button = new JButton("Click Me!");
9         // Set button position and size
10        button.setBounds(140, 80, 120, 40);
11        // Add ActionListener to the button
12        button.addActionListener(new ActionListener() {
13            public void actionPerformed(ActionEvent e) {
14                // Action performed when button is clicked
15                JOptionPane.showMessageDialog(frame, "Button Clicked!");
16            }
17        });
18        // Add button to the frame
19        frame.add(button);
20        // Set frame size
21        frame.setSize(400, 300);
22        // Set layout to null (absolute positioning)
23        frame.setLayout(null);
24        // Set default close operation
25        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
26        // Make frame visible
27        frame.setVisible(true);
    }}
```

## Output:

```
D:\Java Code\swing>javac ButtonExample.java
```

```
D:\Java Code\swing>java ButtonExample
```



## Explanation

### Thing

### Purpose

JButton

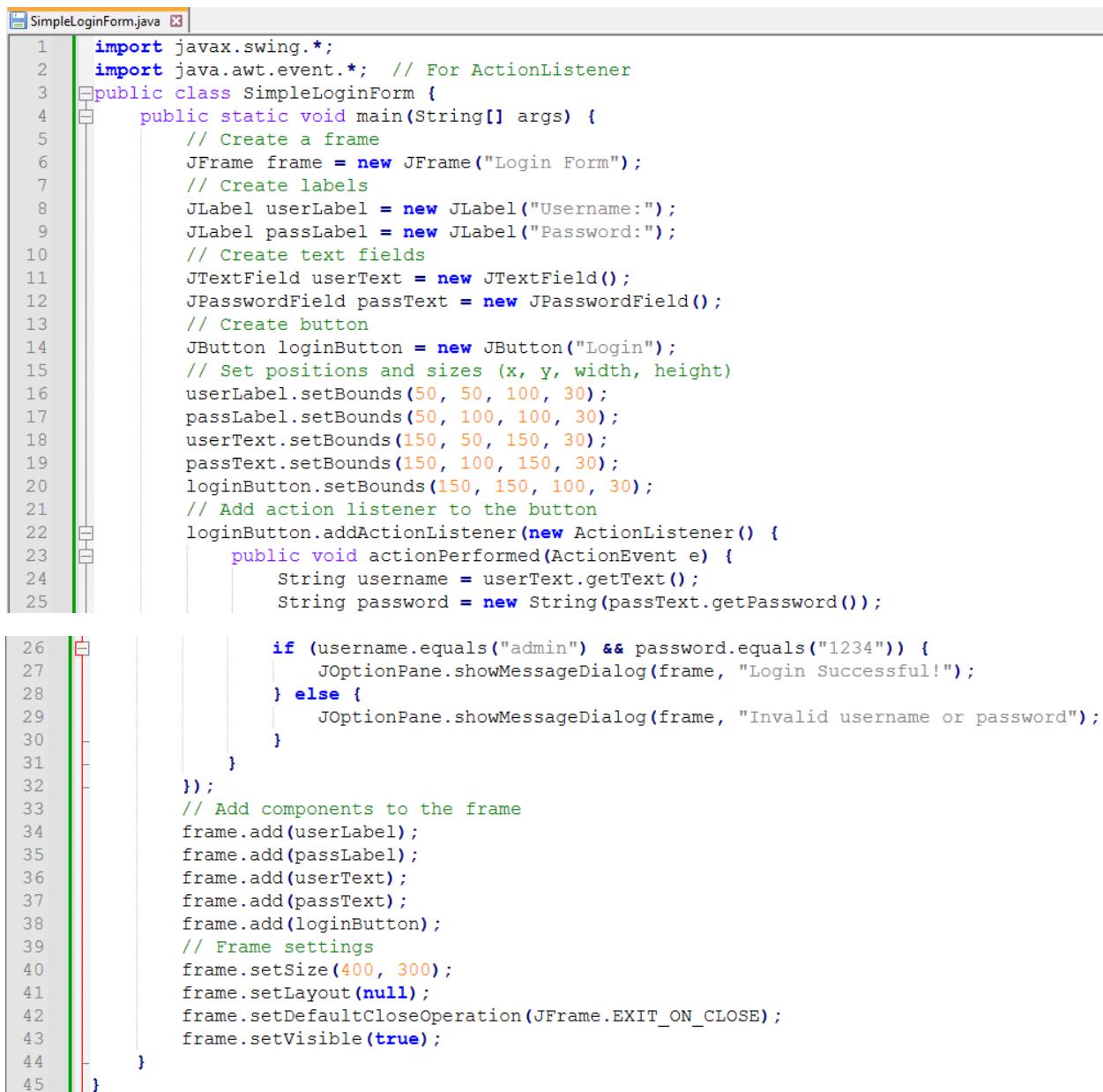
Creates the clickable button

setBounds(x, y, width, height)

Sets position and size manually

| Thing                           | Purpose                   |
|---------------------------------|---------------------------|
| addActionListener()             | Listens for clicks        |
| JOptionPane.showMessageDialog() | Shows a small message box |

### Mini Project: Simple Login Form (Swing + ActionListener)



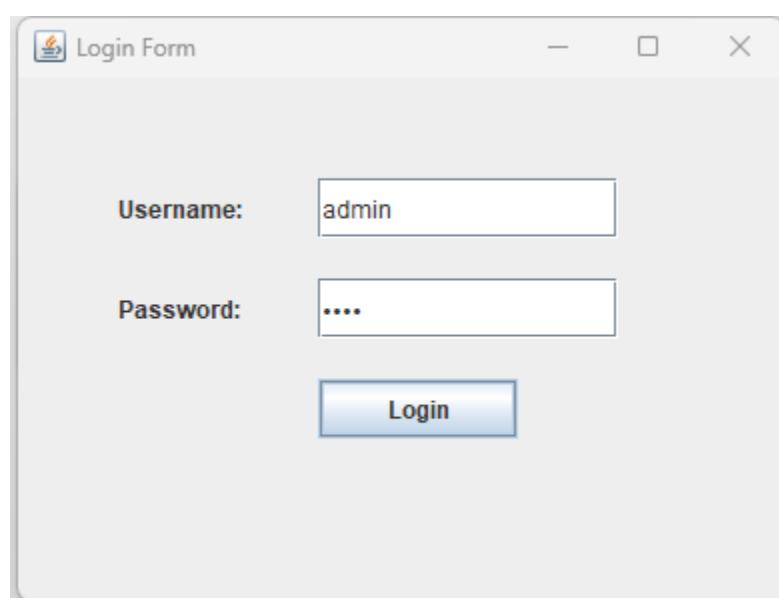
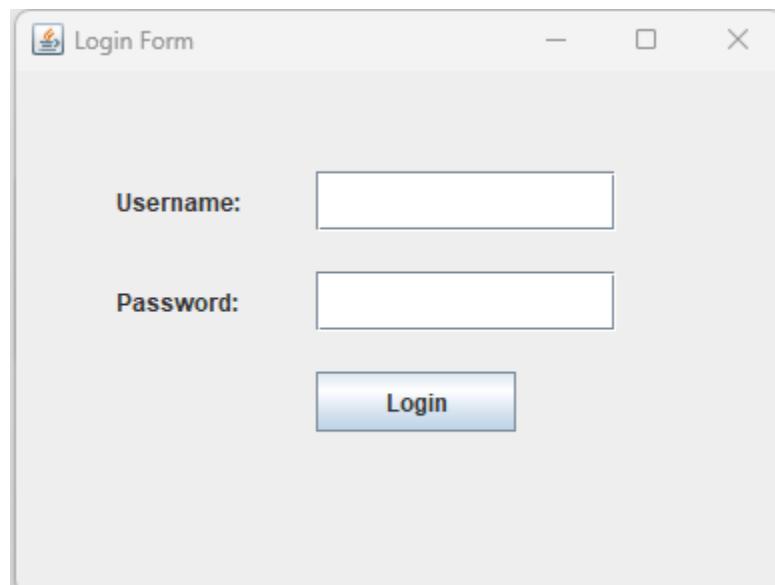
```

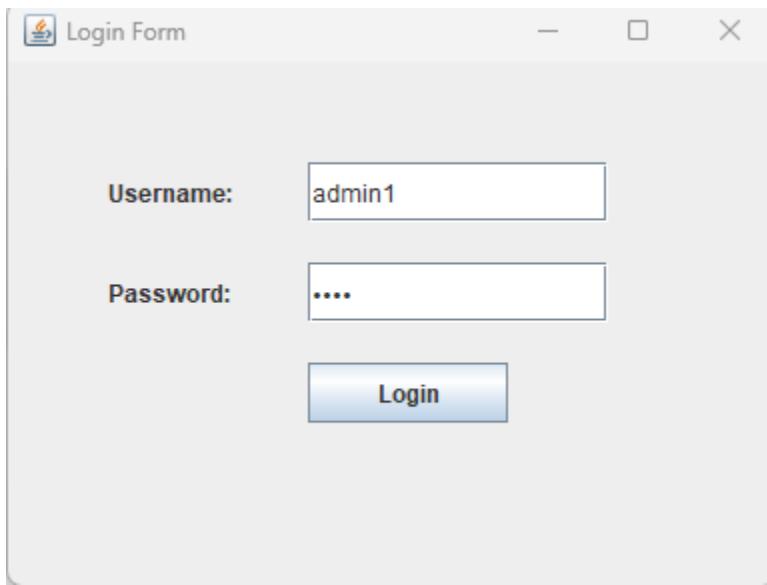
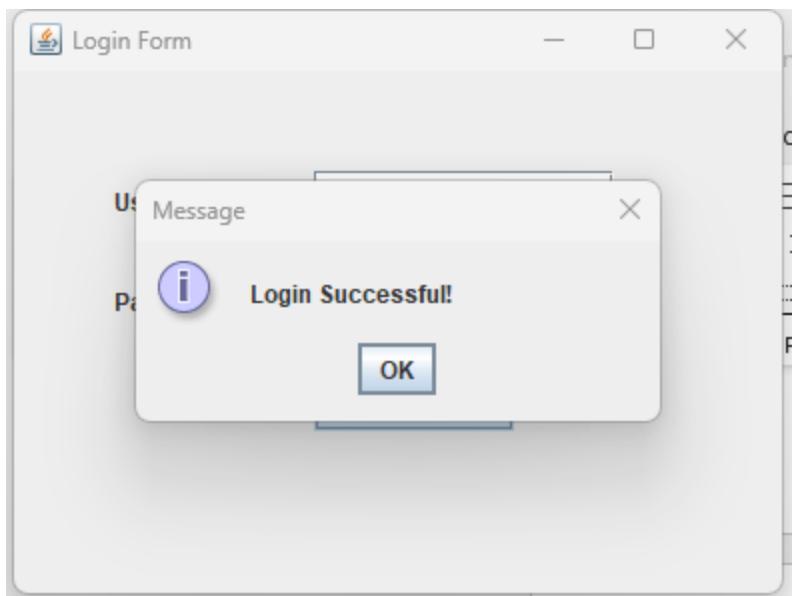
1 import javax.swing.*;
2 import java.awt.event.*; // For ActionListener
3 public class SimpleLoginForm {
4     public static void main(String[] args) {
5         // Create a frame
6         JFrame frame = new JFrame("Login Form");
7         // Create labels
8         JLabel userLabel = new JLabel("Username:");
9         JLabel passLabel = new JLabel("Password:");
10        // Create text fields
11        JTextField userText = new JTextField();
12        JPasswordField passText = new JPasswordField();
13        // Create button
14        JButton loginButton = new JButton("Login");
15        // Set positions and sizes (x, y, width, height)
16        userLabel.setBounds(50, 50, 100, 30);
17        passLabel.setBounds(50, 100, 100, 30);
18        userText.setBounds(150, 50, 150, 30);
19        passText.setBounds(150, 100, 150, 30);
20        loginButton.setBounds(150, 150, 100, 30);
21        // Add action listener to the button
22        loginButton.addActionListener(new ActionListener() {
23            public void actionPerformed(ActionEvent e) {
24                String username = userText.getText();
25                String password = new String(passText.getPassword());
26
27                if (username.equals("admin") && password.equals("1234")) {
28                    JOptionPane.showMessageDialog(frame, "Login Successful!");
29                } else {
30                    JOptionPane.showMessageDialog(frame, "Invalid username or password");
31                }
32            }
33        });
34        // Add components to the frame
35        frame.add(userLabel);
36        frame.add(passLabel);
37        frame.add(userText);
38        frame.add(passText);
39        frame.add(loginButton);
40        // Frame settings
41        frame.setSize(400, 300);
42        frame.setLayout(null);
43        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
44        frame.setVisible(true);
45    }
}

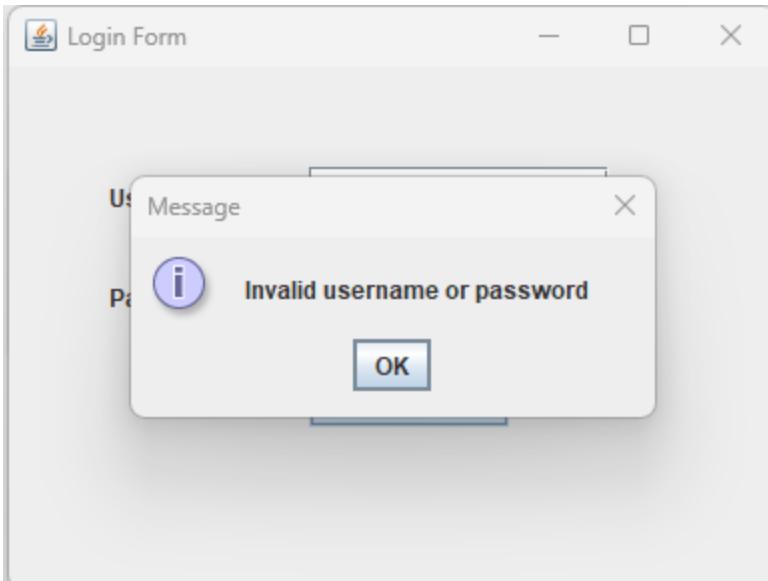
```

**Output:**

```
C:\WINDOWS\system32\cmd. X + | v  
D:\Java Code\swing>javac SimpleLoginForm.java  
D:\Java Code\swing>java SimpleLoginForm
```







## Explanation

| Component           | Purpose                                     |
|---------------------|---|
| JLabel              | To display "Username" and "Password" labels |
| JTextField          | To type username                            |
| JPasswordField      | To type password                            |
| JButton             | To click and check login                    |
| addActionListener() | To handle the button click                  |

### **JDBC Example:**

**Here's the Java program to connect to the database, insert employee records, and display them.**

```
import java.sql.*; // For JDBC classes
import javax.swing.*;
public class SimpleJDBCEExample {
    public static void main(String[] args) {
        // Database connection info
        String url = "jdbc:mysql://localhost:3306/company"; // Your MySQL database URL
        String username = "root"; // Your database username
        String password = "password"; // Your database password
        // JDBC Connection & Statement objects
        Connection conn = null;
        Statement stmt = null;
        try {
            // Connect to the database
            conn = DriverManager.getConnection(url, username, password);
            stmt = conn.createStatement();
            // SQL query to insert a new employee
            String insertSQL = "INSERT INTO emp (name, salary) VALUES ('Aman', 50000.00)";
            stmt.executeUpdate(insertSQL);
            // SQL query to select all employees from 'emp' table
            String selectSQL = "SELECT * FROM emp";
            ResultSet rs = stmt.executeQuery(selectSQL);
            // Display the result set (emp details)
            while (rs.next()) {
                int empId = rs.getInt("emp_id");
```

```
String name = rs.getString("name");
double salary = rs.getDouble("salary");
// Show employee data in a message dialog
JOptionPane.showMessageDialog(null, "Emp ID: " + empId + "\nName: " + name +
"\nSalary: " + salary);
}

// Close the ResultSet and Statement
rs.close();
stmt.close();

} catch (SQLException e) {
// Handle SQL exceptions
e.printStackTrace();
} finally {
try {
// Close the connection
if (conn != null) {
conn.close();
}
} catch (SQLException e) {
e.printStackTrace();
}
}
}
}
```

## Java Swing Event Handling - Step-by-Step Programs

### Step 1: Basic Frame with Button

```
import javax.swing.*;  
public class Step1_BasicFrame {  
    public static void main(String[] args) {  
        JFrame frame = new JFrame("Step 1: Basic Frame");  
        JButton button = new JButton("Click Me");  
        button.setBounds(100, 100, 120, 40);  
        frame.add(button);  
        frame.setSize(300, 300);  
        frame.setLayout(null);  
        frame.setVisible(true);  
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);  
    }  
}
```

### Step 2: Button Click Event Handling

```
import javax.swing.*;  
import java.awt.event.*;  
public class Step2_ButtonClickEvent {  
    public static void main(String[] args) {  
        JFrame frame = new JFrame("Step 2: Button Event");  
        JButton button = new JButton("Click Me");  
        button.setBounds(100, 100, 120, 40);  
        button.addActionListener(new ActionListener() {  
            public void actionPerformed(ActionEvent e) {  
                JOptionPane.showMessageDialog(frame, "Button Clicked!");  
            }  
        });  
        frame.add(button);  
        frame.setSize(300, 300);  
        frame.setLayout(null);  
    }  
}
```

```
frame.setVisible(true);
frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
}
}
```

### **Step 3: Text Input Event Handling**

```
import javax.swing.*;
import java.awt.event.*;
public class Step3_TextFieldEvent {
public static void main(String[] args) {
JFrame frame = new JFrame("Step 3: Text Input");
JTextField textField = new JTextField();
JButton button = new JButton("Submit");
textField.setBounds(50, 50, 200, 30);
button.setBounds(100, 100, 100, 30);
button.addActionListener(e -> {
String name = textField.getText();
 JOptionPane.showMessageDialog(frame, "Hello, " + name);
});
frame.add(textField);
frame.add(button);
frame.setSize(300, 250);
frame.setLayout(null);
frame.setVisible(true);
frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
}
}
```

### **Step 4: CheckBox and RadioButton**

```
import javax.swing.*;
import java.awt.event.*;
public class Step4_CheckBoxRadioButton {
public static void main(String[] args) {
```

```
JFrame frame = new JFrame("Step 4: CheckBox & Radio");
JCheckBox cbJava = new JCheckBox("Java");
JCheckBox cbPython = new JCheckBox("Python");
JRadioButton rMale = new JRadioButton("Male");
JRadioButton rFemale = new JRadioButton("Female");
ButtonGroup genderGroup = new ButtonGroup();
genderGroup.add(rMale);
genderGroup.add(rFemale);
JButton button = new JButton("Submit");
cbJava.setBounds(50, 30, 100, 30);
cbPython.setBounds(150, 30, 100, 30);
rMale.setBounds(50, 70, 100, 30);
rFemale.setBounds(150, 70, 100, 30);
button.setBounds(100, 120, 100, 30);
button.addActionListener(e -> {
    String skills = "Skills: ";
    if (cbJava.isSelected()) skills += "Java ";
    if (cbPython.isSelected()) skills += "Python ";
    String gender = rMale.isSelected() ? "Male" : (rFemale.isSelected() ?
        "Female" : "Not selected");
    JOptionPane.showMessageDialog(frame, skills + "\nGender: " + gender);
});
frame.add(cbJava); frame.add(cbPython);
frame.add(rMale); frame.add(rFemale);
frame.add(button);
frame.setSize(350, 230);
frame.setLayout(null);
frame.setVisible(true);
frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
}
}
```

### **Step 5: ComboBox and JList**

```
import javax.swing.*;
import java.awt.event.*;
public class Step5_ComboBoxList {
    public static void main(String[] args) {
        JFrame frame = new JFrame("Step 5: ComboBox & List");
        String[] countries = {"India", "USA", "UK", "Germany"};
        String[] colors = {"Red", "Green", "Blue", "Yellow"};
        JComboBox<String> countryBox = new JComboBox<>(countries);
        JList<String> colorList = new JList<>(colors);
        JButton button = new JButton("Show Selection");
        countryBox.setBounds(50, 30, 150, 30);
        colorList.setBounds(50, 70, 100, 80);
        button.setBounds(50, 170, 150, 30);
        button.addActionListener(e -> {
            String country = (String) countryBox.getSelectedItem();
            String color = colorList.getSelectedValue();
            JOptionPane.showMessageDialog(frame, "Country: " + country + "\nColor: " +
                color);
        });
        frame.add(countryBox);
        frame.add(colorList);
        frame.add(button);
        frame.setSize(300, 300);
        frame.setLayout(null);
        frame.setVisible(true);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }
}
```

### **Step 6: Mouse Events**

```
import javax.swing.*;
```

```
import java.awt.event.*;
public class Step6_MouseEvents {
    public static void main(String[] args) {
        JFrame frame = new JFrame("Step 6: Mouse Events");
        JLabel label = new JLabel("Click anywhere!");
        label.setBounds(80, 50, 200, 30);
        frame.addMouseListener(new MouseAdapter() {
            public void mouseClicked(MouseEvent e) {
                label.setText("Clicked at: " + e.getX() + ", " + e.getY());
            }
        });
        frame.add(label);
        frame.setSize(300, 200);
        frame.setLayout(null);
        frame.setVisible(true);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }
}
```

### Step 7: Window Events

```
import javax.swing.*;
import java.awt.event.*;
public class Step7_WindowEvents {
    public static void main(String[] args) {
        JFrame frame = new JFrame("Step 7: Window Events");
        JLabel label = new JLabel("Try minimizing or closing this window.");
        label.setBounds(30, 50, 300, 30);
        frame.addWindowListener(new WindowAdapter() {
            public void windowIconified(WindowEvent e) {
                System.out.println("Window Minimized");
            }
            public void windowDeiconified(WindowEvent e) {
```

```

System.out.println("Window Restored");
}

public void windowClosing(WindowEvent e) {
    System.out.println("Window is closing...");
    JOptionPane.showMessageDialog(frame, "Closing App...");
    System.exit(0);
}
});

frame.add(label);
frame.setSize(350, 200);
frame.setLayout(null);
frame.setVisible(true);
}
}

```

## **Mini-Project: Swing-Based Student Feedback Form**

### **Description:**

This project demonstrates a Swing-based GUI for student feedback collection. It integrates JTextField, JRadioButton, JComboBox, JList, JCheckBox, and event handling for buttons to dynamically display the user's input.

### **Complete Code:**

```

import javax.swing.*;
import java.awt.event.*;

public class StudentFeedbackForm {
    public static void main(String[] args) {
        JFrame frame = new JFrame("Student Feedback Form");
        JLabel nameLabel = new JLabel("Name:");
        JTextField nameField = new JTextField();
        JLabel genderLabel = new JLabel("Gender:");
        JRadioButton male = new JRadioButton("Male");

```

```
JRadioButton female = new JRadioButton("Female");
ButtonGroup genderGroup = new ButtonGroup();
JLabel courseLabel = new JLabel("Course:");
String[] courses = {"Java", "Python", "C++"};
JComboBox<String> courseBox = new JComboBox<>(courses);
JLabel ratingLabel = new JLabel("Rate the course:");
JList<String> ratingList = new JList<>(new String[]{"Excellent", "Good", "Average",
"Poor"});
JCheckBox practical = new JCheckBox("Practical");
JCheckBox theory = new JCheckBox("Theory");
JButton submitButton = new JButton("Submit");
JLabel outputLabel = new JLabel("");
// Setting Bounds
nameLabel.setBounds(30, 20, 100, 25);
nameField.setBounds(140, 20, 150, 25);
genderLabel.setBounds(30, 60, 100, 25);
male.setBounds(140, 60, 70, 25);
female.setBounds(210, 60, 80, 25);
courseLabel.setBounds(30, 100, 100, 25);
courseBox.setBounds(140, 100, 150, 25);
ratingLabel.setBounds(30, 140, 120, 25);
ratingList.setBounds(140, 140, 150, 60);
practical.setBounds(30, 210, 100, 25);
theory.setBounds(140, 210, 100, 25);
submitButton.setBounds(100, 250, 120, 30);
outputLabel.setBounds(30, 290, 300, 40);
// Adding components
genderGroup.add(male);
genderGroup.add(female);
frame.add(nameLabel);
frame.add(nameField);
```

```
frame.add(genderLabel);
frame.add(male);
frame.add(female);
frame.add(courseLabel);
frame.add(courseBox);
frame.add(ratingLabel);
frame.add(ratingList);
frame.add(practical);
frame.add(theory);
frame.add(submitButton);
frame.add(outputLabel);
// Action Listener
submitButton.addActionListener(e -> {
String name = nameField.getText();
String gender = male.isSelected() ? "Male" : (female.isSelected() ? "Female" :
"Unspecified");
String course = (String) courseBox.getSelectedItem();
String rating = ratingList.getSelectedValue();
String modules = "";
if (practical.isSelected()) modules += "Practical ";
if (theory.isSelected()) modules += "Theory";
outputLabel.setText("<html>Name: " + name + "<br>Gender: " + gender +
"<br>Course: " + course + "<br>Rating: " + rating +
"<br>Modules: " + modules + "</html>");
});
frame.setSize(350, 400);
frame.setLayout(null);
frame.setVisible(true);
frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
}
}
```