

# Java Tutorial

Java Tutorial or Core Java Tutorial or Java Programming Tutorial is a widely used robust technology. Let's start learning of java from basic questions like what is java tutorial, core java, where it is used, what type of applications are created in java and why use java.

## What is Java

Java is a **programming language** and a **platform**.

Java is a high level, robust, secured and object-oriented programming language.

**Platform:** Any hardware or software environment in which a program runs, is known as a platform. Since Java has its own runtime environment (JRE) and API, it is called platform.

## Java Example

Let's have a quick look at java programming example. A detailed description of hello java example is given in next page.

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

**Test it Now**

## Where it is 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, javatpoint.com etc.
3. Enterprise Applications such as banking applications.
4. Mobile
5. Embedded System
6. Smart Card
7. Robotics
8. Games etc.

## Types of Java Applications

There are mainly 4 type of applications that can be created using java programming:

### 1) Standalone Application

It is also known as desktop application or window-based application. An application that we need to install on every machine such as media player, antivirus etc. AWT and Swing are used in java for creating standalone applications.

### 2) Web Application

An application that runs on the server side and creates dynamic page, is called web application. Currently, servlet, jsp, struts, jsf etc. technologies are used for creating 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 for creating enterprise applications.

### 4) Mobile Application

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

# History of Java

**Java history** is interesting to know. The history of java starts from Green Team. Java team members (also known as **Green Team**), initiated a revolutionary task to develop a language for digital devices such as set-top boxes, televisions etc.

For the green team members, it was an advance concept at that time. But, it was suited for internet programming. Later, Java technology as incorporated by Netscape.

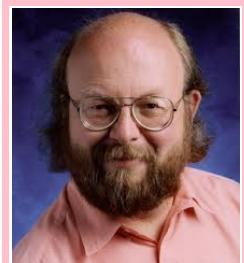
Currently, Java is used in internet programming, mobile devices, games, e-business solutions etc. There are given the major points that describes the history of java.

1) **James Gosling, Mike Sheridan, and Patrick Naughton** initiated the Java language project in June 1991. The small team of sun engineers called **Green Team**.

2) Originally designed for small, embedded systems in electronic appliances like set-top boxes.

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

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



**James Gosling**

## Why Oak name for java language?

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



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

## Why Java name for java language?

7) **Why they choosed java name for java language?** The team gathered to choose a new name. The suggested words were "dynamic", "revolutionary", "Silk", "jolt", "DNA" etc. They wanted something that reflected the essence of the technology: revolutionary, dynamic, lively, cool, unique, and easy to spell and fun to say.

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

8) Java is an island of Indonesia where first coffee was produced (called java coffee).

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

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

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

12) JDK 1.0 released in(January 23, 1996).

## Java Version History

There are many java versions that has been released. Current stable release of Java is Java SE 8.

1. JDK Alpha and Beta (1995)
2. JDK 1.0 (23rd Jan, 1996)
3. JDK 1.1 (19th Feb, 1997)
4. J2SE 1.2 (8th Dec, 1998)
5. J2SE 1.3 (8th May, 2000)
6. J2SE 1.4 (6th Feb, 2002)
7. J2SE 5.0 (30th Sep, 2004)

8. Java SE 6 (11th Dec, 2006)
9. Java SE 7 (28th July, 2011)
10. Java SE 8 (18th March, 2014)

## Features of Java

There are many features of Java. They are also known as Java buzzwords. The Java Features given below are simple and easy to understand.

1. Simple
  2. Object-Oriented
  3. Platform independent
  4. Secured
  5. Robust
  6. Architecture neutral
  7. Portable
  8. Dynamic
  9. Interpreted
  10. High Performance
  11. Multithreaded
  12. Distributed
- 

### Simple

According to Sun, Java language is simple because:

Java syntax is based on C++ (so easier for programmers to learn it after C++).

Java removed many confusing and/or rarely-used features e.g., explicit pointers, operator overloading etc.

No need to remove unreferenced objects because there is Automatic Garbage Collection in Java.

---

### Object-oriented

Object-oriented means we organize our software as a combination of different types of objects that incorporates both data and behaviour.

Object-oriented programming(OOPS) is a methodology that simplifies software development and maintenance by providing some rules.

Basic concepts of OOPS are:

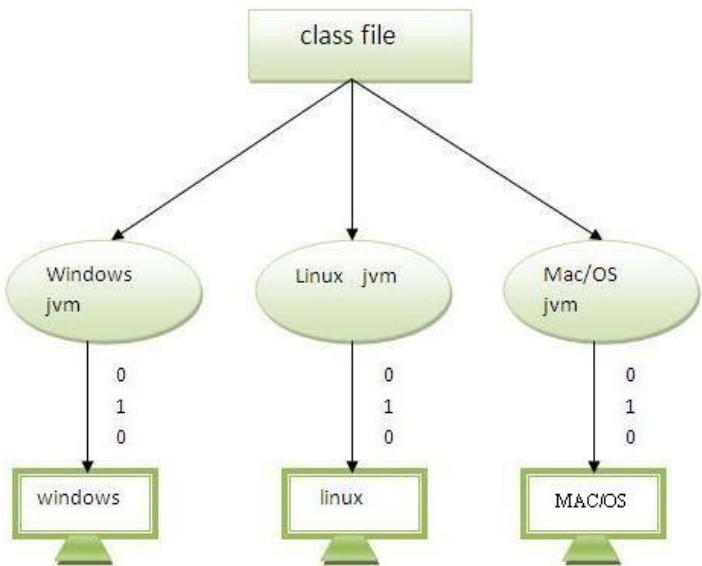
1. Object
  2. Class
  3. Inheritance
  4. Polymorphism
  5. Abstraction
  6. Encapsulation
- 

### Platform Independent

A platform is the hardware or software environment in which a program runs. There are two types of platforms software-based and hardware-based. Java provides software-based platform. The Java platform differs from most other platforms in the sense that it's a software-based platform that runs on top of other hardware-based platforms. It has two components:

1. Runtime Environment
2. API(Application Programming Interface)

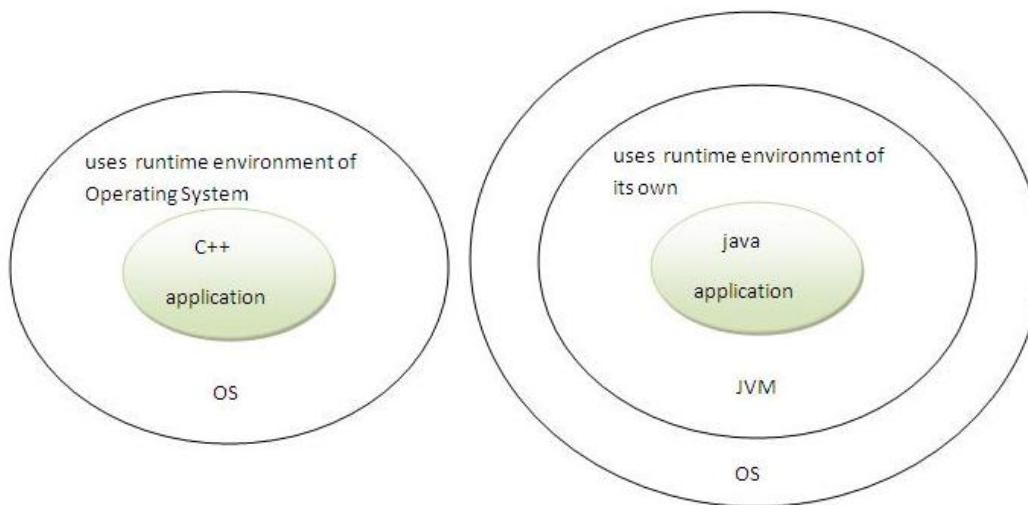
Java code can be run on multiple platforms e.g. Windows, Linux, Sun Solaris, Mac/OS etc. Java code is compiled by the compiler and converted into bytecode. This bytecode is a platform independent code because it can be run on multiple platforms i.e. Write Once and Run Anywhere (WORA).



## Secured

Java is secured because:

- No explicit pointer
- Programs run inside virtual machine sandbox.



- **Classloader**- adds security by separating the package for the classes of the local file system from those that are imported from network sources.
- **Bytecode Verifier**- checks the code fragments for illegal code that can violate access right to objects.
- **Security Manager**- determines what resources a class can access such as reading and writing to the local disk.

These security features are provided by the Java language. Some security can also be provided by application developer through SSL, JAAS, cryptography etc.

## Robust

Robust simply means strong. Java uses strong memory management. There are no pointers that avoid security problems. There is automatic garbage collection in Java. There is exception handling and type checking mechanism in Java. All these points make Java robust.

## Architecture-neutral

There are no implementation-dependent features e.g. size of primitive types is set.

## Portable

We may carry the java bytecode to any platform.

---

## High-performance

Java is faster than traditional interpretation since byte code is "close" to native code still somewhat slower than a compiled language (e.g., C++)

---

## Distributed

We can create distributed applications in java. RMI and EJB are used for creating distributed applications. We may access files by calling the methods from any machine on the internet.

---

## Multi-threaded

A thread is like a separate program, executing concurrently. We can write Java programs that deal with many tasks at once by defining multiple threads. The main advantage of multi-threading is that it shares the same memory. Threads are important for multi-media, Web applications etc.

# Simple Program of Java

In this page, we will learn how to write the simple program of java. We can write a simple hello java program easily after installing the JDK.

To create a simple java program, you need to create a class that contains main method. Let's understand the requirement first.

## Requirement for Hello Java Example

For executing any java program, you need to

- install the JDK if you don't have installed it, [download the JDK](#) and install it.
  - set path of the jdk/bin directory. <http://www.javatpoint.com/how-to-set-path-in-java>
  - create the java program
  - compile and run the java program
- 

## Creating hello java example

Let's create the hello java program:

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

**Test it Now**

save this file as Simple.java

**To compile:** javac Simple.java

**To execute:** java Simple

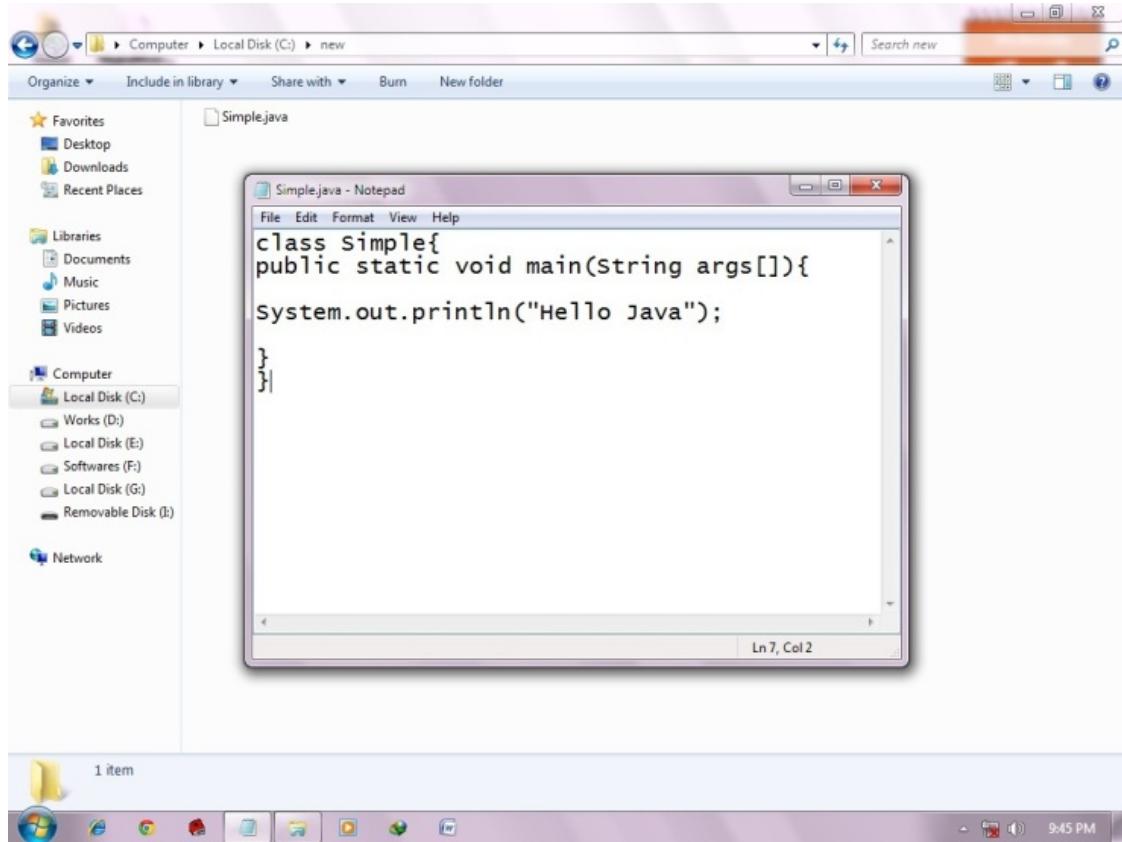
**Output:** Hello Java

## Understanding first java program

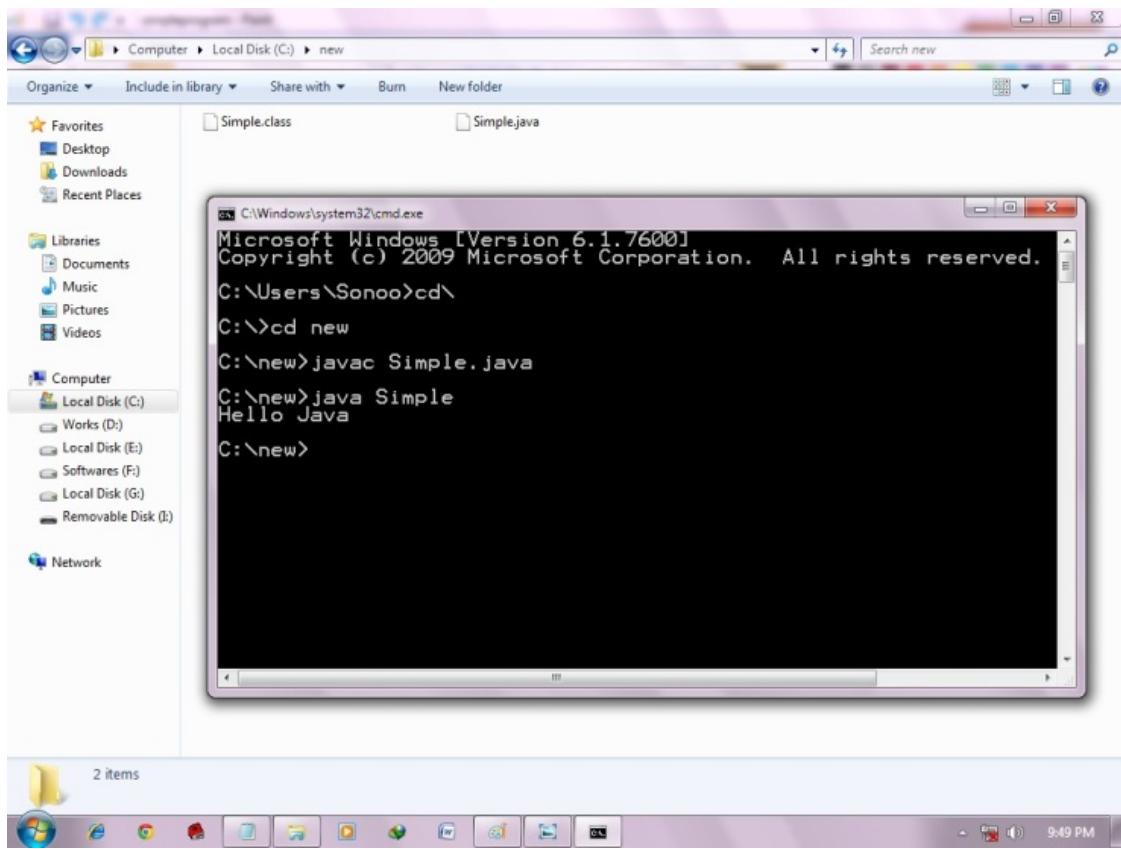
Let's see what is the meaning of class, public, static, void, main, String[], System.out.println().

- **class** keyword is used to declare a class in java.
- **public** keyword is an access modifier which represents visibility, it means it is visible to all.
- **static** is a keyword, if we declare any method as static, it is known as static method. The core advantage of static method is that there is no need to create object to invoke the static method. The main method is executed by the JVM, so it doesn't require to create object to invoke the main method. So it saves memory.
- **void** is the return type of the method, it means it doesn't return any value.
- **main** represents startup of the program.
- **String[] args** is used for command line argument. We will learn it later.
- **System.out.println()** is used print statement. We will learn about the internal working of System.out.println statement later.

To write the simple program, open notepad by **start menu -> All Programs -> Accessories -> notepad** and write simple program as displayed below:



As displayed in the above diagram, write the simple program of java in notepad and saved it as Simple.java. To compile and run this program, you need to open command prompt by **start menu -> All Programs -> Accessories -> command prompt**.



To compile and run the above program, go to your current directory first; my current directory is c:\new . Write here:

**To compile:** javac Simple.java

**To execute:** java Simple

## How many ways can we write a java program

There are many ways to write a java program. The modifications that can be done in a java program are given below:

### 1) By changing sequence of the modifiers, method prototype is not changed.

Let's see the simple code of main method.

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

### 2) subscript notation in java array can be used after type, before variable or after variable.

Let's see the different codes to write the main method.

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

### 3) You can provide var-args support to main method by passing 3 ellipses (dots)

Let's see the simple code of using var-args in main method. We will learn about var-args later in Java New Features chapter.

```
public static void main(String... args)
```

### 4) Having semicolon at the end of class in java is optional.

Let's see the simple code.

```
class A{
    static public void main(String... args){
        System.out.println("hello java4");
    }
};
```

## Valid java main method signature

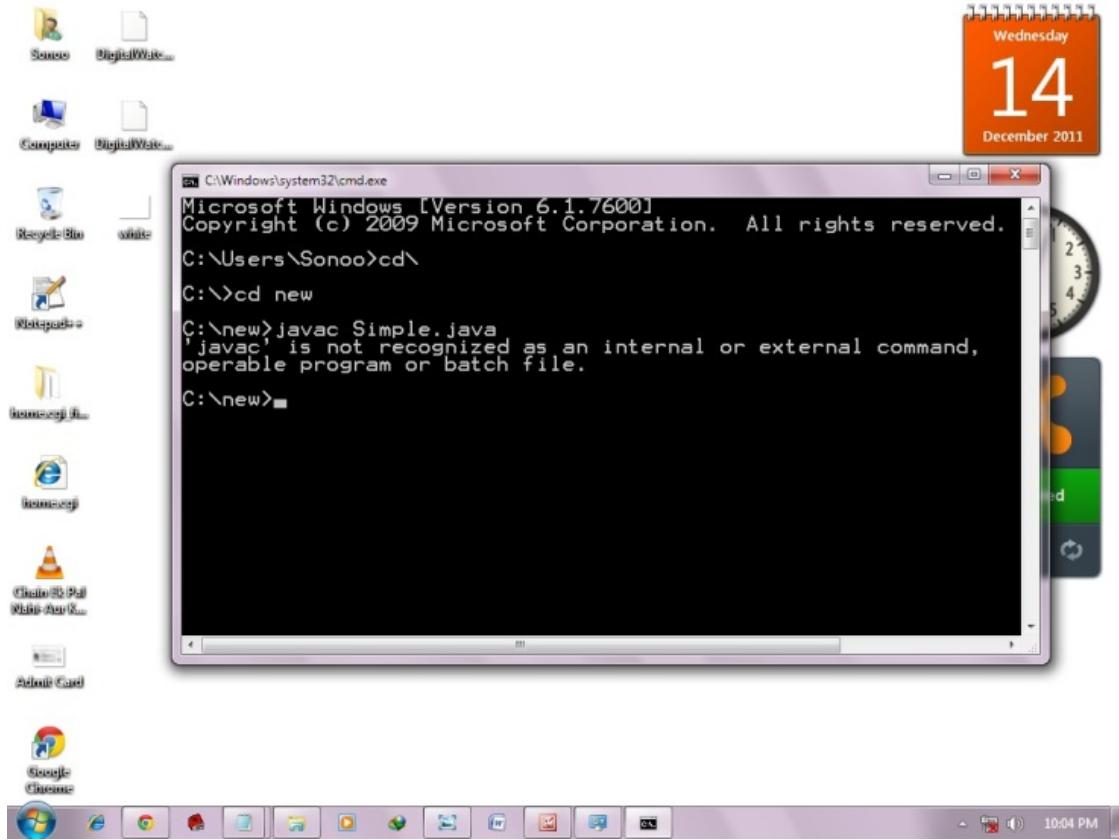
```
public static void main(String[] args)
public static void main(String []args)
public static void main(String args[])
public static void main(String... args)
static public void main(String[] args)
public static final void main(String[] args)
final public static void main(String[] args)
final strictfp public static void main(String[] args)
```

## Invalid java main method signature

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

### Resolving an error "javac is not recognized as an internal or external command" ?

If there occurs a problem like displayed in the below figure, you need to set path. Since DOS doesn't know javac or java, we need to set path. Path is not required in such a case if you save your program inside the jdk/bin folder. But its good approach to set path. Click here for [How to set path in java](#).

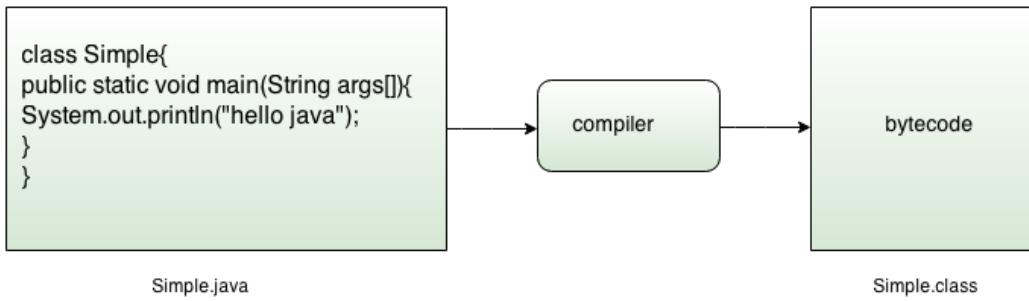


## Internal Details of Hello Java Program

In the previous page, we have learned about the first program, how to compile and how to run the first java program. Here, we are going to learn, what happens while compiling and running the java program. Moreover, we will see some question based on the first program.

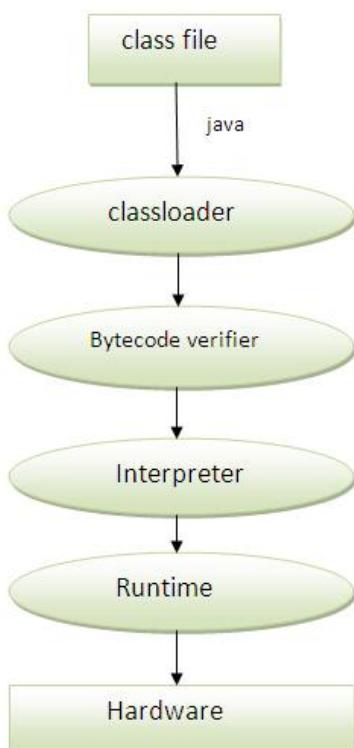
### What happens at compile time?

At compile time, java file is compiled by Java Compiler (It does not interact with OS) and converts the java code into bytecode.



## What happens at runtime?

At runtime, following steps are performed:



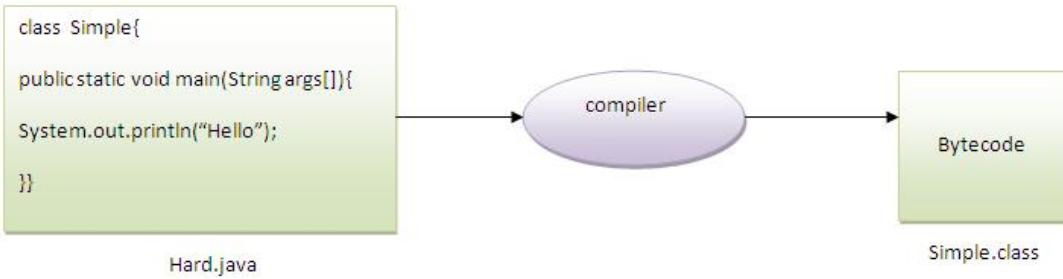
**Classloader:** is the subsystem of JVM that is used to load class files.

**Bytecode Verifier:** checks the code fragments for illegal code that can violate access right to objects.

**Interpreter:** read bytecode stream then execute the instructions.

**Q) Can you save a java source file by other name than the class name?**

Yes, if the class is not public. It is explained in the figure given below:



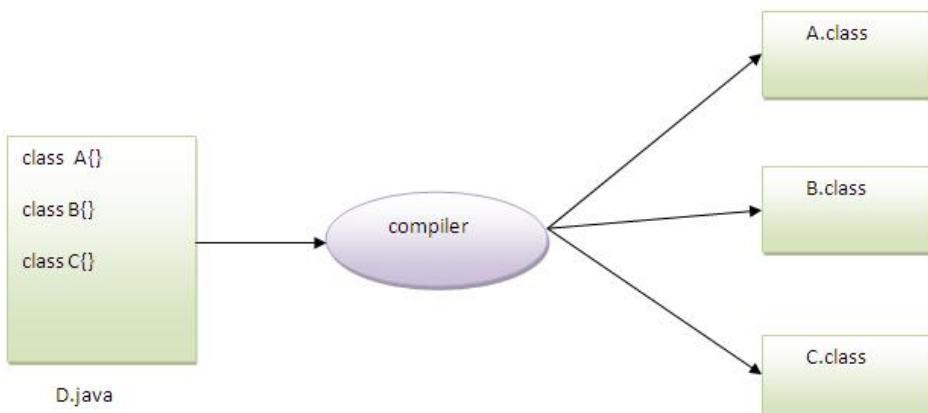
**To compile:** javac Hard.java

**To execute:** java Simple

---

**Q) Can you have multiple classes in a java source file?**

Yes, like the figure given below illustrates:



# How to set path in Java

The path is required to be set for using tools such as javac, java etc.

If you are saving the java source file inside the jdk/bin directory, path is not required to be set because all the tools will be available in the current directory.

But If you are having your java file outside the jdk/bin folder, it is necessary to set path of JDK.

There are 2 ways to set java path:

1. temporary
2. permanent

## 1) How to set Temporary Path of JDK in Windows

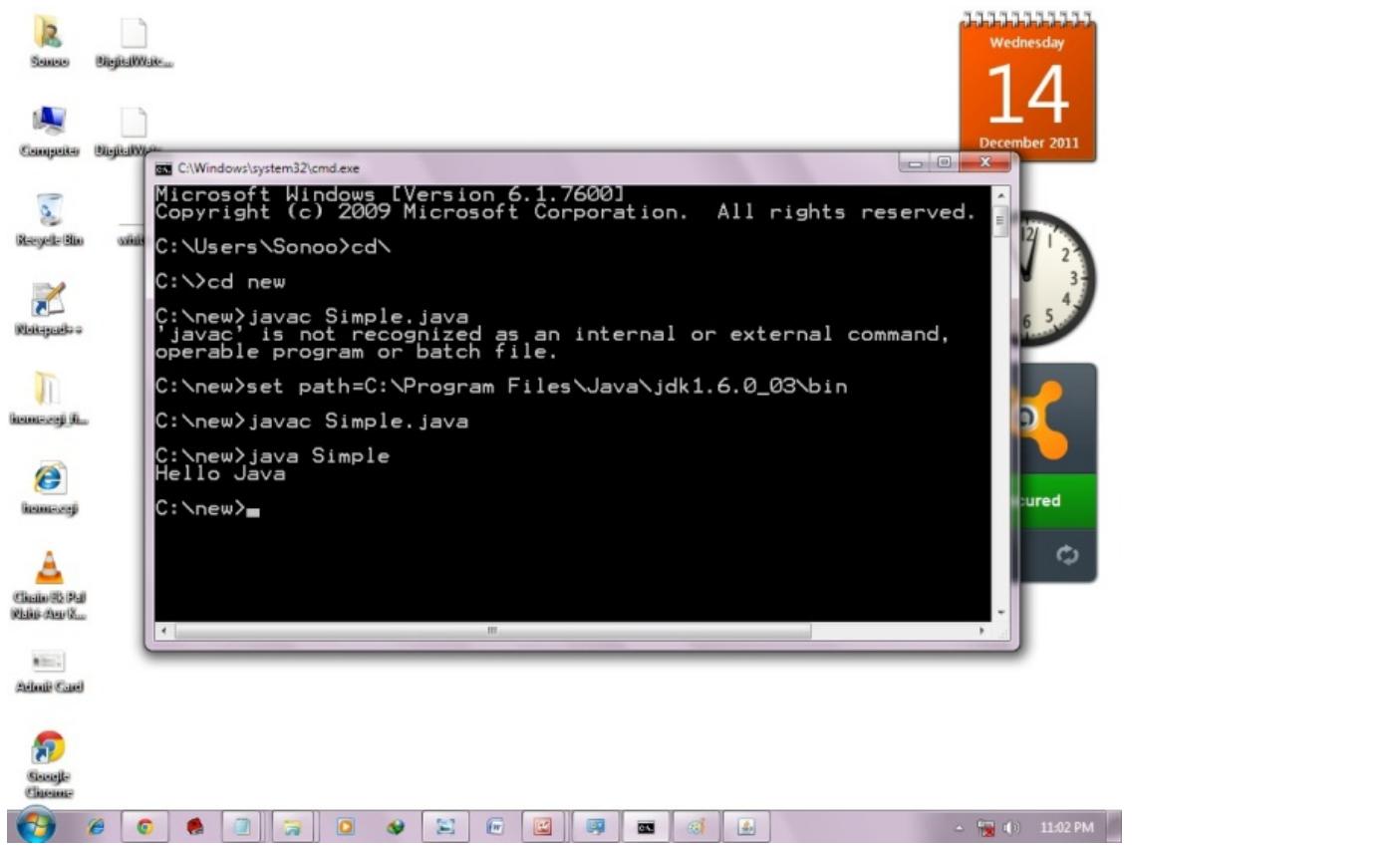
To set the temporary path of JDK, you need to follow following steps:

- Open command prompt
- copy the path of jdk/bin directory
- write in command prompt: set path=copied\_path

For Example:

```
set path=C:\Program Files\Java\jdk1.6.0_23\bin
```

Let's see it in the figure given below:



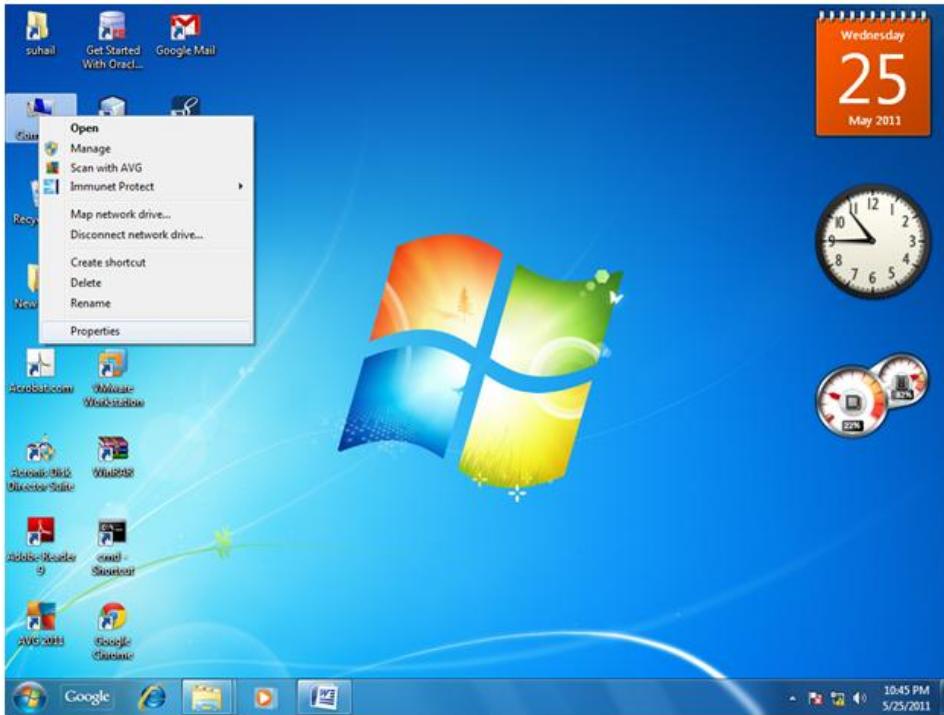
## 2) How to set Permanent Path of JDK in Windows

For setting the permanent path of JDK, you need to follow these steps:

- Go to MyComputer properties -> advanced tab -> environment variables -> new tab of user variable -> write path in variable name -> write path of bin folder in variable value -> ok -> ok -> ok

For Example:

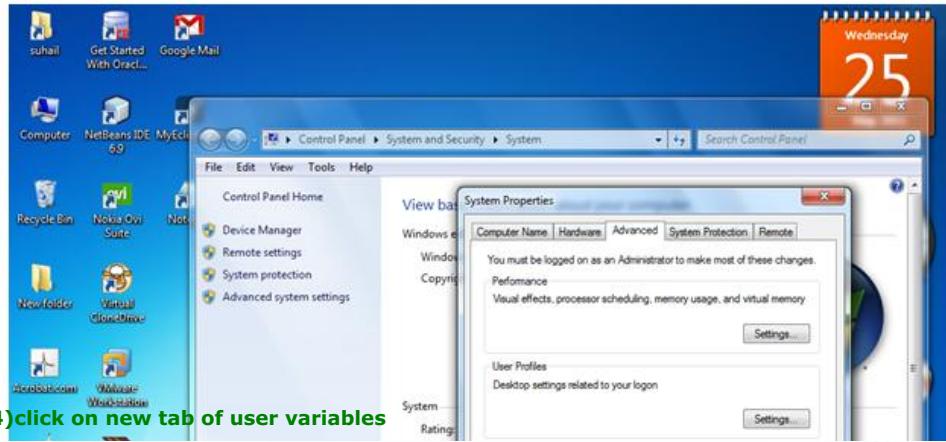
### 1) Go to MyComputer properties



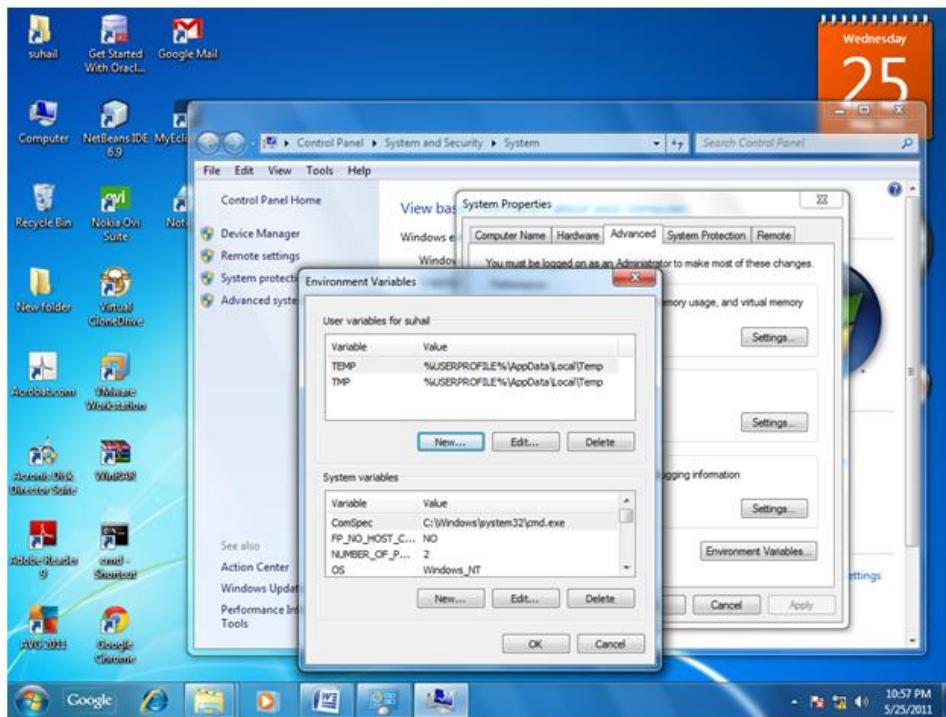
**2)click on advanced tab**



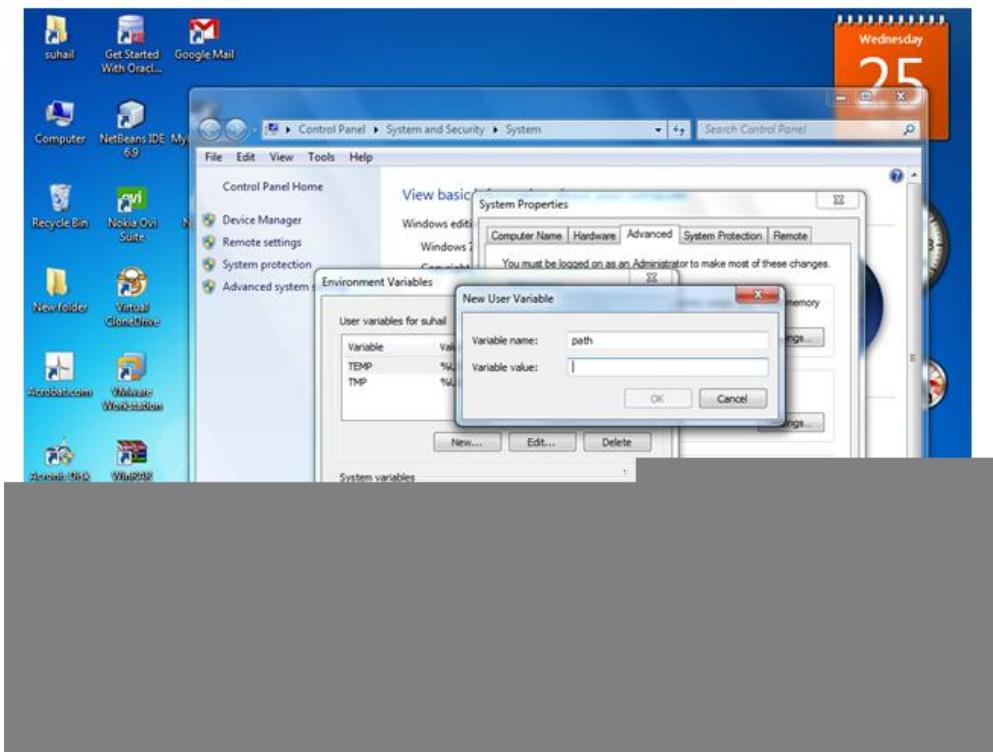
**3)click on environment variables**



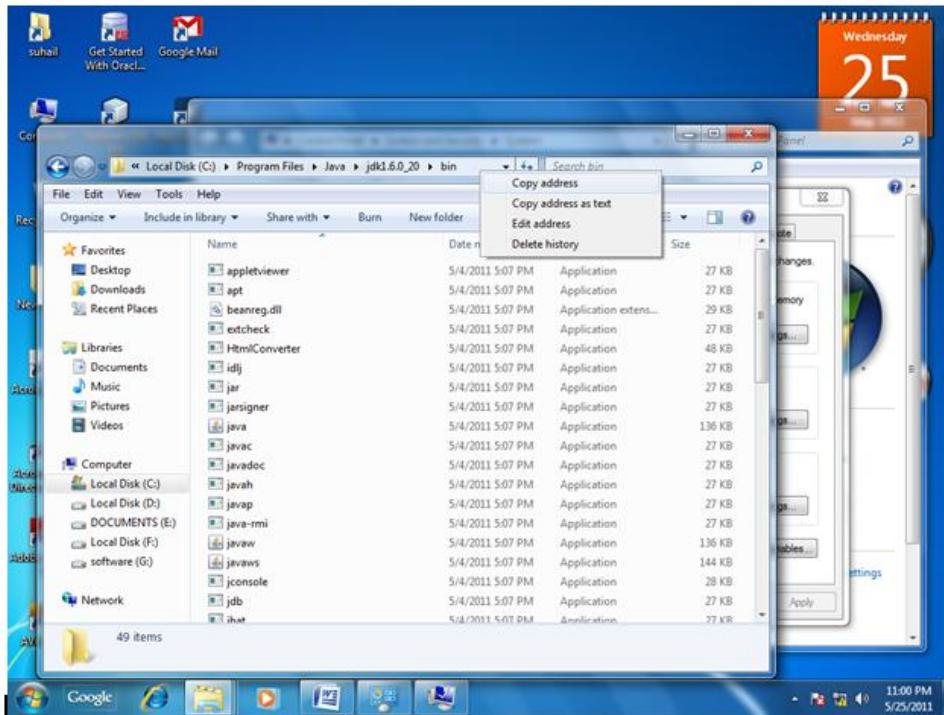
4)click on new tab of user variables



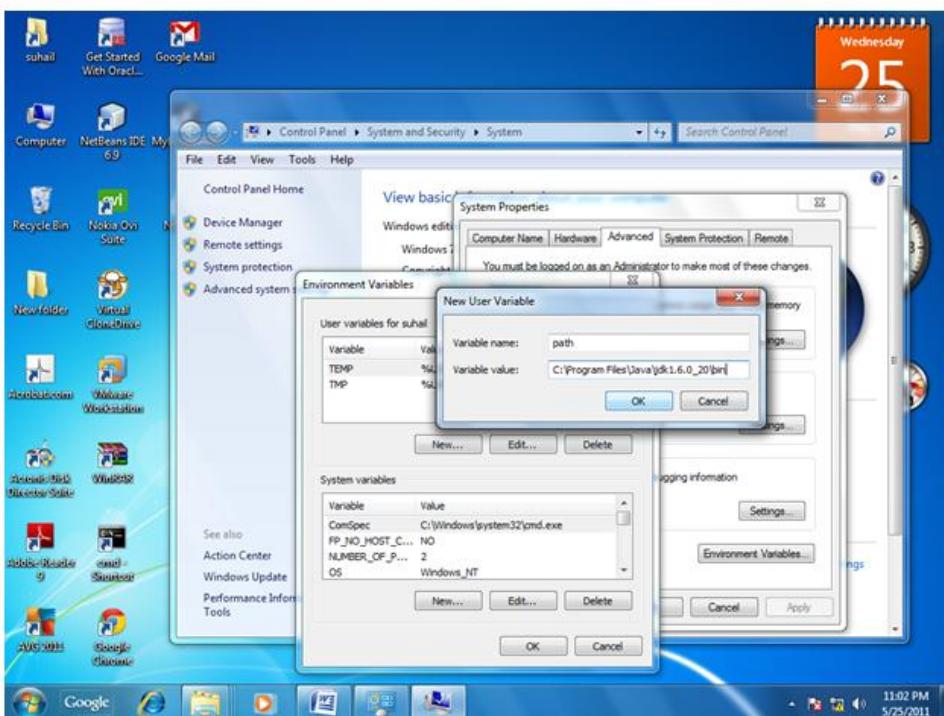
5)write path in variable name



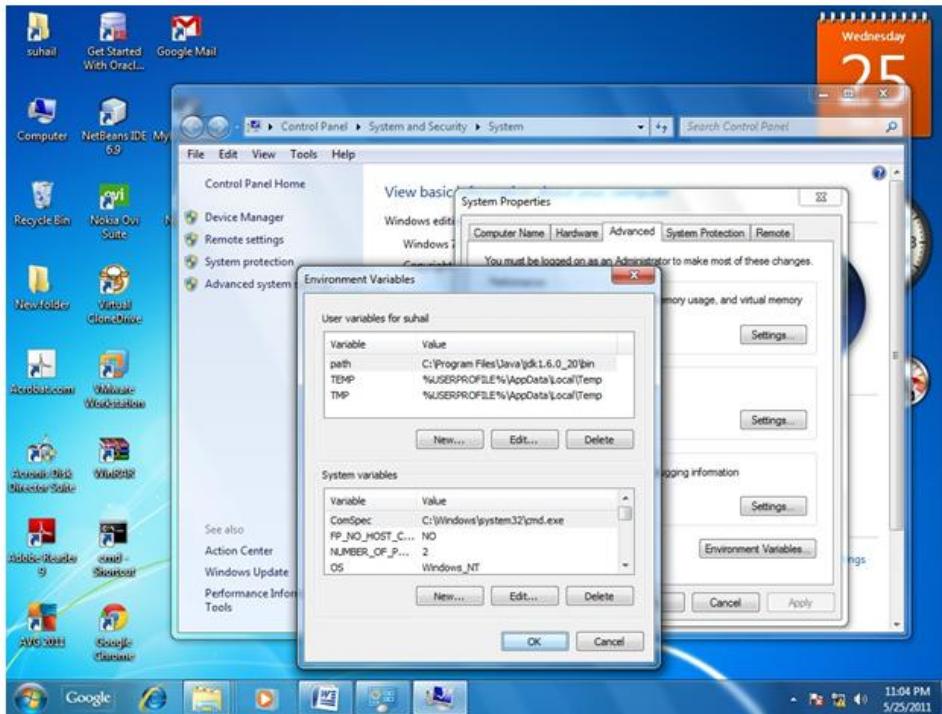
**6)Copy the path of bin folder**



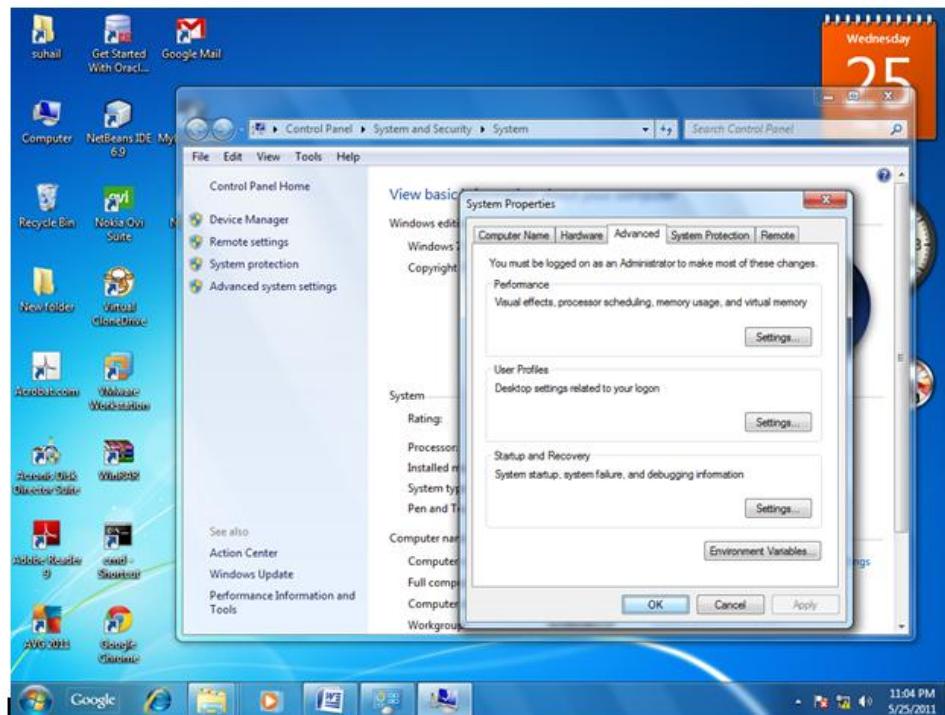
**7)paste path of bin folder in variable value**



**8)click on ok button**



**9)click on ok button**



Now your permanent path is set. You can now execute any program of java from any drive.

## Setting Java Path in Linux OS

Setting the path in Linux OS is same as setting the path in the Windows OS. But here we use export tool rather than set. Let's see how to set path in Linux OS:

```
export PATH=$PATH:/home/jdk1.6.01/bin/
```

Here, we have installed the JDK in the home directory under Root (/home).

---

## JVM

### Difference between JDK, JRE and JVM

Understanding the difference between JDK, JRE and JVM is important in Java. We are having brief overview of JVM here.

If you want to get the detailed knowledge of Java Virtual Machine, move to the next page. Firstly, let's see the basic differences between the JDK, JRE and JVM.

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

JVMs are available for many hardware and software platforms. JVM, JRE and JDK are platform dependent because configuration of each OS differs. But, Java is platform independent.

The JVM performs following main tasks:

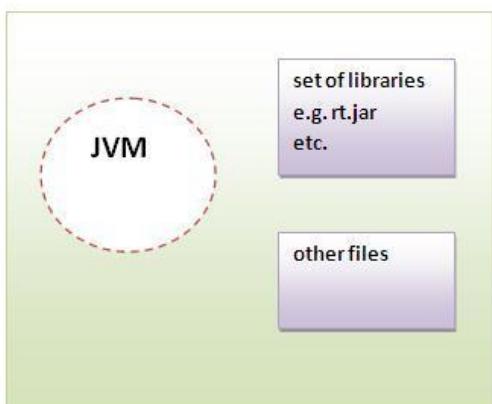
- Loads code
- Verifies code
- Executes code
- Provides runtime environment

---

## JRE

JRE is an acronym for Java Runtime Environment. It is used to provide runtime environment. It is the implementation of JVM. It physically exists. It contains set of libraries + other files that JVM uses at runtime.

Implementation of JVMs are also actively released by other companies besides Sun Micro Systems.

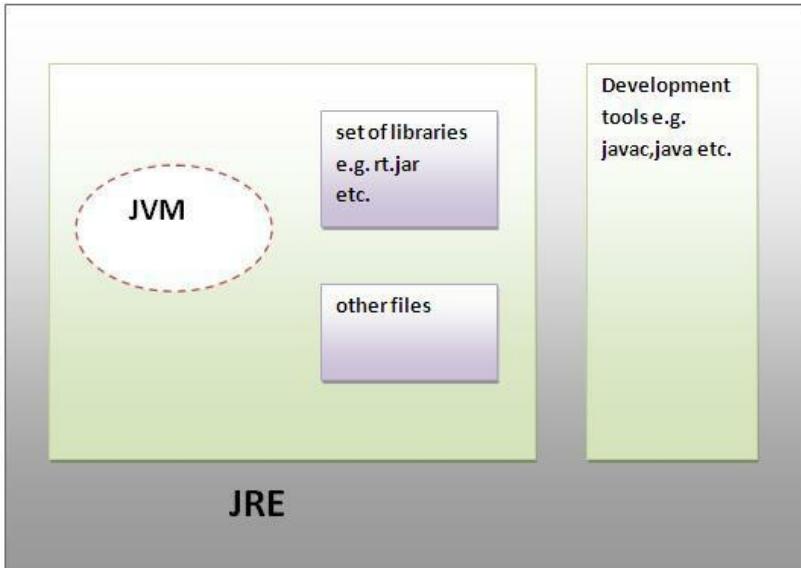


## JRE

---

## JDK

JDK is an acronym for Java Development Kit. It physically exists. It contains JRE + development tools.



## JDK

## JVM (Java Virtual Machine)

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

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

## What is JVM?

It is:

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

## What it does?

The JVM performs following operation:

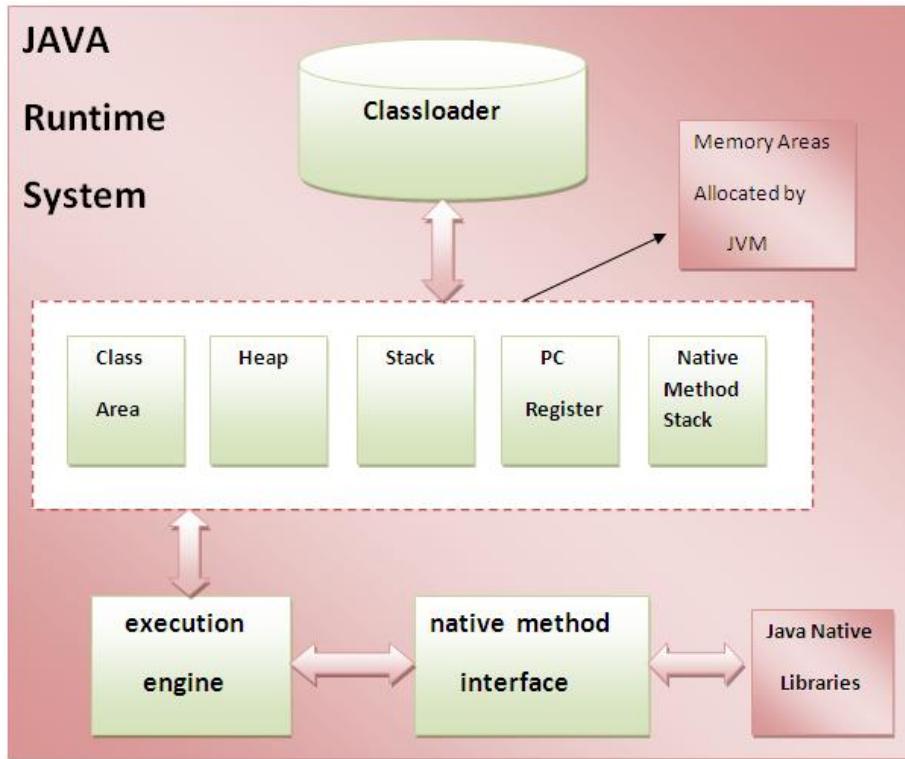
- Loads code
- Verifies code
- Executes code
- Provides runtime environment

JVM provides definitions for the:

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

## Internal Architecture of JVM

Let's understand the internal architecture of JVM. It contains classloader, memory area, execution engine etc.



### 1) Classloader:

Classloader is a subsystem of JVM that is used to load class files.

### 2) Class(Method) Area:

Class(Method) Area stores per-class structures such as the runtime constant pool, field and method data, the code for methods.

### 3) Heap:

It is the runtime data area in which objects are allocated.

### 4) Stack:

Java Stack stores frames. It holds local variables and partial results, and plays a part in method invocation and return.

Each thread has a private JVM stack, created at the same time as thread.

A new frame is created each time a method is invoked. A frame is destroyed when its method invocation completes.

### 5) Program Counter Register:

PC (program counter) register. It contains the address of the Java virtual machine instruction currently being executed.

### 6) Native Method Stack:

It contains all the native methods used in the application.

## 7) Execution Engine:

It contains:

**1) A virtual processor**

**2) Interpreter:** Read bytecode stream then execute the instructions.

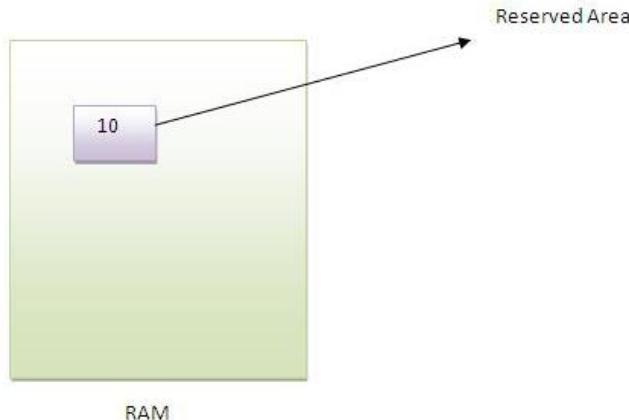
**3) Just-In-Time(JIT) compiler:** It is used to improve the performance. JIT compiles parts of the byte code that have similar functionality at the same time, and hence reduces the amount of time needed for compilation. Here the term ?compiler? refers to a translator from the instruction set of a Java virtual machine (JVM) to the instruction set of a specific CPU.

# Variable and Datatype in Java

In this page, we will learn about the variable and java data types. Variable is a name of memory location. There are three types of variables: local, instance and static. There are two types of datatypes in java, primitive and non-primitive.

## Variable

Variable is name of reserved area allocated in memory.

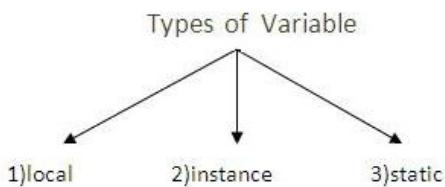


```
int data=50;//Here data is variable
```

## Types of Variable

There are three types of variables in java

- local variable
- instance variable
- static variable



## Local Variable

A variable that is declared inside the method is called local variable.

## Instance Variable

A variable that is declared inside the class but outside the method is called instance variable . It is not declared as static.

## Static variable

A variable that is declared as static is called static variable. It cannot be local.

We will have detailed learning of these variables in next chapters.

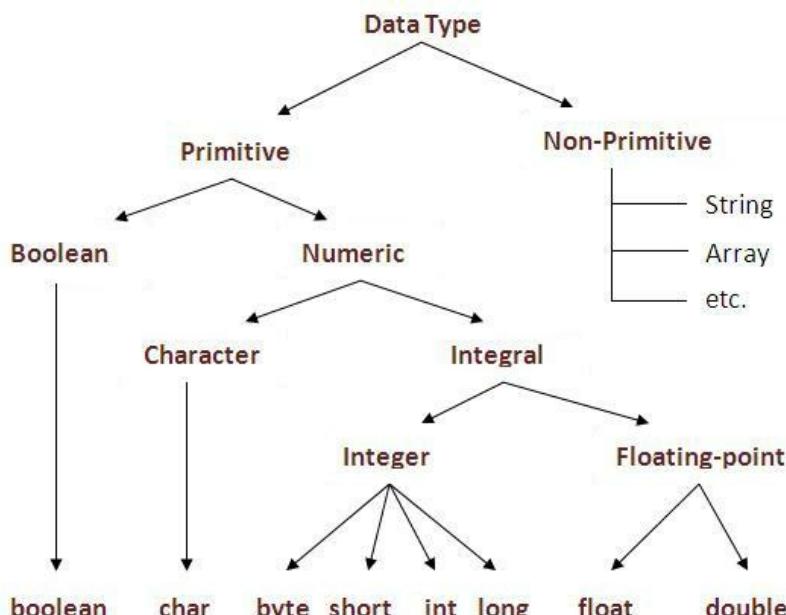
## Example to understand the types of variables

```
class A{  
    int data=50;//instance variable  
    static int m=100;//static variable  
    void method(){  
        int n=90;//local variable  
    }  
}//end of class
```

## Data Types in Java

In java, there are two types of data types

- primitive data types
- non-primitive data types



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

## Why char uses 2 byte in java and what is \u0000 ?

because java uses unicode system rather than ASCII code system. \u0000 is the lowest range of unicode system.To get detail about Unicode see below.

## Unicode System

Unicode is a universal international standard character encoding that is capable of representing most of the world's written languages.

### Why java uses Unicode System?

Before Unicode, there were many language standards:

- **ASCII** (American Standard Code for Information Interchange) for the United States.
- **ISO 8859-1** for Western European Language.
- **KOI-8** for Russian.
- **GB18030 and BIG-5** for chinese, and so on.

#### This caused two problems:

1. A particular code value corresponds to different letters in the various language standards.
2. The encodings for languages with large character sets have variable length. Some common characters are encoded as single bytes, others require two or more bytes.

To solve these problems, a new language standard was developed i.e. Unicode System.

In unicode, character holds 2 byte, so java also uses 2 byte for characters.

**lowest value:**\u0000

**highest value:**\uFFFF

## Operators in java

**Operator** in java is a symbol that is used to perform operations. There are many types of operators in java such as unary operator, arithmetic operator, relational operator, shift operator, bitwise operator, ternary operator and assignment operator.

Operators	Precedence
postfix	<i>expr++ expr--</i>
unary	<i>++expr --expr +expr -expr ~ !</i>
multiplicative	<i>* / %</i>
additive	<i>+ -</i>
shift	<i>&lt;&lt; &gt;&gt; &gt;&gt;&gt;</i>
relational	<i>&lt; &gt; &lt;= &gt;= instanceof</i>
equality	<i>== !=</i>
bitwise AND	<i>&amp;</i>
bitwise exclusive OR	<i>^</i>
bitwise inclusive OR	<i> </i>
logical AND	<i>&amp;&amp;</i>
logical OR	<i>  </i>
ternary	<i>? :</i>

# Java Programs

Java programs are frequently asked in the interview. These programs can be asked from control statements, array, string, oops etc. Let's see the list of java programs.

## 1) Fibonacci series

Write a java program to print fibonacci series without using recursion and using recursion.

**Input:** 10

**Output:** 0 1 1 2 3 5 8 13 21 34

## 2) Prime number

Write a java program to check prime number.

**Input:** 44

**Output:** not prime number

**Input:** 7

**Output:** prime number

## 3) Palindrome number

Write a java program to check palindrome number.

**Input:** 329

**Output:** not palindrome number

**Input:** 12321

**Output:** palindrome number

## 4) Factorial number

Write a java program to print factorial of a number.

**Input:** 5

**Output:** 120

**Input:** 6

**Output:** 720

## 5) Armstrong number

Write a java program to check Armstrong number.

**Input:** 153

**Output:** Armstrong number

**Input:** 22

**Output:** not Armstrong number

# Java OOPs Concepts

In this page, we will learn about basics of OOPs. Object Oriented Programming is a paradigm that provides many concepts such as **inheritance, data binding, polymorphism** etc.

**Simula** is considered as the first object-oriented programming language. The programming paradigm where everything is represented as an object, is known as truly object-oriented programming language.

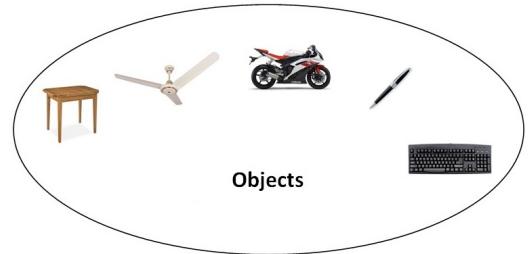
**Smalltalk** is considered as the first truly object-oriented programming language.

## OOPs (Object Oriented Programming System)

**Object** means a real word entity such as pen, chair, table etc. **Object-Oriented**

**Programming** is a methodology or paradigm to design a program using classes and objects. It simplifies the software development and maintenance by providing some concepts:

- Object
- Class
- Inheritance
- Polymorphism
- Abstraction
- Encapsulation



### Object

Any entity that has state and behavior is known as an object. For example: chair, pen, table, keyboard, bike etc. It can be physical and logical.

### Class

**Collection of objects** is called class. It is a logical entity.

### Inheritance

**When one object acquires all the properties and behaviours of parent object**i.e. known as inheritance. It provides code reusability. It is used to achieve runtime polymorphism.

### Polymorphism

When **one task is performed by different ways**i.e. known as polymorphism. For example: to converse the customer differently, to draw something e.g. shape or rectangle etc.

In java, we use method overloading and method overriding to achieve polymorphism.

Another example can be to speak something e.g. cat speaks meaw, dog barks woof etc.



### Abstraction

**Hiding internal details and showing functionality**is known as abstraction. For example: phone call, we don't know the internal processing.

In java, we use abstract class and interface to achieve abstraction.

### Encapsulation

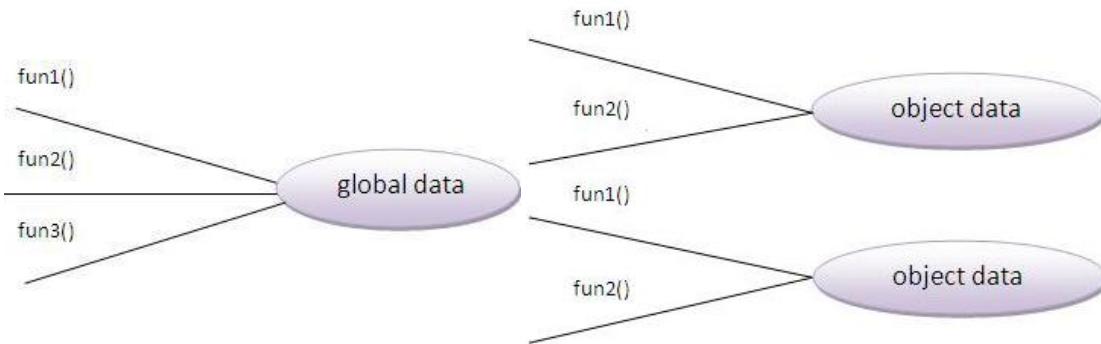
**Binding (or wrapping) code and data together into a single unit is known as encapsulation** For example: capsule, it is wrapped with different medicines.



A java class is the example of encapsulation. Java bean is the fully encapsulated class because all the data members are private here.

## Advantage of OOPs over Procedure-oriented programming language

- 1) OOPs makes development and maintenance easier where as in Procedure-oriented programming language it is not easy to manage if code grows as project size grows.
- 2) OOPs provides data hiding whereas in Procedure-oriented programming language a global data can be accessed from anywhere.
- 3) OOPs provides ability to simulate real-world event much more effectively. We can provide the solution of real word problem if we are using the Object-Oriented Programming language.



## What is difference between object-oriented programming language and object-based programming language?

Object based programming language follows all the features of OOPs except Inheritance. JavaScript and VBScript are examples of object based programming languages.

## Java Naming conventions

Java **naming convention** is a rule to follow as you decide what to name your identifiers such as class, package, variable, constant, method etc.

But, it is not forced to follow. So, it is known as convention not rule.

All the classes, interfaces, packages, methods and fields of java programming language are given according to java naming convention.

## Advantage of naming conventions in java

By using standard Java naming conventions, you make your code easier to read for yourself and for other programmers. Readability of Java program is very important. It indicates that **less time** is spent to figure out what the code does.

Name	Convention
class name	should start with uppercase letter and be a noun e.g. String, Color, Button, System, Thread etc.
interface name	should start with uppercase letter and be an adjective e.g. Runnable, Remote, ActionListener etc.
method name	should start with lowercase letter and be a verb e.g. actionPerformed(), main(), print(), println() etc.
variable name	should start with lowercase letter e.g. firstName, orderNumber etc.
package name	should be in lowercase letter e.g. java, lang, sql, util etc.
constants name	should be in uppercase letter. e.g. RED, YELLOW, MAX_PRIORITY etc.

## CamelCase in java naming conventions

Java follows camelcase syntax for naming the class, interface, method and variable.

If name is combined with two words, second word will start with uppercase letter always e.g. actionPerformed(), firstName, ActionEvent, ActionListener etc.

# Object and Class in Java

In this page, we will learn about java objects and classes. In object-oriented programming technique, we design a program using objects and classes.

Object is the physical as well as logical entity whereas class is the logical entity only.

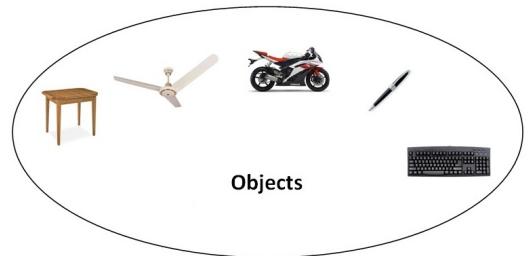
## Object in Java

An entity that has state and behavior is known as an object e.g. chair, bike, marker, pen, table, car etc. It can be physical or logical (tangible and intangible).

The example of tangible object is banking system.

An object has three characteristics:

- **state:** represents data (value) of an object.
- **behavior:** represents the behavior (functionality) of an object such as deposit, withdraw etc.
- **identity:** Object identity is typically implemented via a unique ID. The value of the ID is not visible to the external user. But, it is used internally by the JVM to identify each object uniquely.



For Example: Pen is an object. Its name is Reynolds, color is white etc. known as its state. It is used to write, so writing is its behavior.

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

## Class in Java

A class is a group of objects that has common properties. It is a template or blueprint from which objects are created.

A class in java can contain:

- **data member**
- **method**
- **constructor**
- **block**
- **class and interface**

Syntax to declare a class:

```
class <class_name>{
    data member;
    method;
}
```

## Simple Example of Object and Class

In this example, we have created a Student class that have two data members id and name. We are creating the object of the Student class by new keyword and printing the objects value.

```
class Student1{
    int id;//data member (also instance variable)
    String name;//data member(also instance variable)

    public static void main(String args[]){
        Student1 s1=new Student1();//creating an object of Student
        System.out.println(s1.id);
        System.out.println(s1.name);
    }
}
```

[Test it Now](#)

Output:0 null

---

## Instance variable in Java

A variable that is created inside the class but outside the method, is known as instance variable. Instance variable doesn't get memory at compile time. It gets memory at runtime when object(instance) is created. That is why, it is known as instance variable.

## Method in Java

In java, a method is like function i.e. used to expose behaviour of an object.

## Advantage of Method

- Code Reusability
- Code Optimization

## new keyword

The new keyword is used to allocate memory at runtime.

---

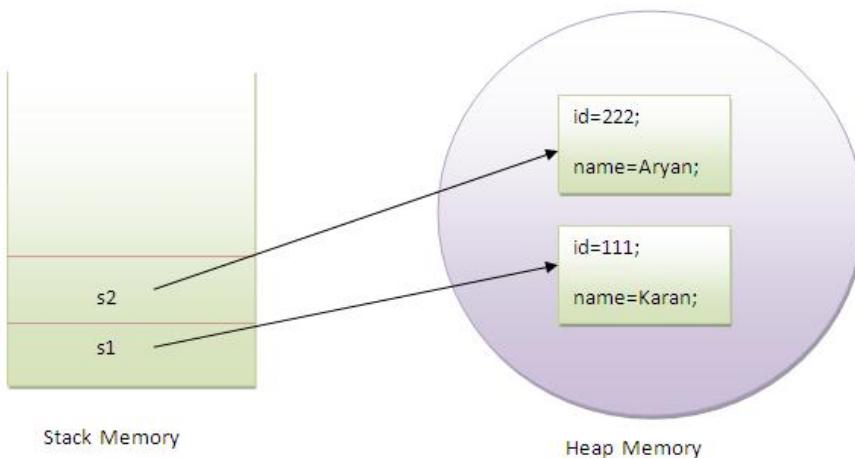
## Example of Object and class that maintains the records of students

In this example, we are creating the two objects of Student class and initializing the value to these objects by invoking the insertRecord method on it. Here, we are displaying the state (data) of the objects by invoking the displayInformation method.

```
class Student2{  
    int rollno;  
    String name;  
  
    void insertRecord(int r, String n){ //method  
        rollno=r;  
        name=n;  
    }  
  
    void displayInformation(){System.out.println(rollno+" "+name);}//method  
  
    public static void main(String args[]){  
        Student2 s1=new Student2();  
        Student2 s2=new Student2();  
  
        s1.insertRecord(111,"Karan");  
        s2.insertRecord(222,"Aryan");  
  
        s1.displayInformation();  
        s2.displayInformation();  
    }  
}
```

### Test it Now

```
111 Karan  
222 Aryan
```



As you see in the above figure, object gets the memory in Heap area and reference variable refers to the object allocated in the Heap memory area. Here, s1 and s2 both are reference variables that refer to the objects allocated in memory.

---

## Another Example of Object and Class

There is given another example that maintains the records of Rectangle class. Its explanation is same as in the above Student class example.

```
class Rectangle{
    int length;
    int width;

    void insert(int l,int w){
        length=l;
        width=w;
    }

    void calculateArea(){System.out.println(length*width);}

    public static void main(String args[]){
        Rectangle r1=new Rectangle();
        Rectangle r2=new Rectangle();

        r1.insert(11,5);
        r2.insert(3,15);

        r1.calculateArea();
        r2.calculateArea();
    }
}
```

Output:55

45

---

## What are the different ways to create an object in Java?

There are many ways to create an object in java. They are:

- By new keyword
- By newInstance() method
- By clone() method
- By factory method etc.

We will learn, these ways to create the object later.

---

## Anonymous object

Anonymous simply means nameless. An object that have no reference is known as anonymous object.  
If you have to use an object only once, anonymous object is a good approach.

```
class Calculation{  
  
    void fact(int n){  
        int fact=1;  
        for(int i=1;i<=n;i++){  
            fact=fact*i;  
        }  
        System.out.println("factorial is "+fact);  
    }  
  
    public static void main(String args[]){  
        new Calculation().fact(5);//calling method with anonymous object  
    }  
}
```

Output:Factorial is 120

---

## Creating multiple objects by one type only

We can create multiple objects by one type only as we do in case of primitives.

```
Rectangle r1=new Rectangle(),r2=new Rectangle();//creating two objects
```

Let's see the example:

```
class Rectangle{  
    int length;  
    int width;  
  
    void insert(int l,int w){  
        length=l;  
        width=w;  
    }  
  
    void calculateArea(){System.out.println(length*width);}  
  
    public static void main(String args[]){  
        Rectangle r1=new Rectangle(),r2=new Rectangle();//creating two objects  
  
        r1.insert(11,5);  
        r2.insert(3,15);  
  
        r1.calculateArea();  
        r2.calculateArea();  
    }  
}

```
Output:55
```


```

# Method Overloading in Java

If a class have multiple methods by same name but different parameters, it 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 behaviour of the method because its name differs. So, we perform method overloading to figure out the program quickly.

## Advantage of method overloading?

Method overloading **increases the readability of the program**

## Different ways to overload the method

There are two ways to overload the method in java



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

 **In java, Method Overloading is not possible by changing the return type of the method.**

## 1) Example of Method Overloading by changing the no. of arguments

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

```
class Calculation{  
    void sum(int a,int b){System.out.println(a+b);}  
    void sum(int a,int b,int c){System.out.println(a+b+c);}  
  
    public static void main(String args[]){  
        Calculation obj=new Calculation();  
        obj.sum(10,10,10);  
        obj.sum(20,20);  
  
    }  
}
```

**Test it Now**

Output:  
30  
40

## 2) Example of Method Overloading by changing data type of 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.

```
class Calculation2{  
    void sum(int a,int b){System.out.println(a+b);}  
    void sum(double a,double b){System.out.println(a+b);}
```

```
public static void main(String args[]){
Calculation2 obj=new Calculation2();
obj.sum(10.5,10.5);
obj.sum(20,20);

}
}
```

**Test it Now**

Output:21.0

40

---

### Que) Why Method Overloading is not possible by changing the return type of method?

In java, method overloading is not possible by changing the return type of the method because there may occur ambiguity. Let's see how ambiguity may occur:

because there was problem:

```
class Calculation3{
int sum(int a,int b){System.out.println(a+b);}
double sum(int a,int b){System.out.println(a+b);}

public static void main(String args[]){
Calculation3 obj=new Calculation3();
int result=obj.sum(20,20); //Compile Time Error

}
}
```

**Test it Now**

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

---

### Can we overload main() method?

Yes, by method overloading. You can have any number of main methods in a class by method overloading. Let's see the simple example:

```
class Overloading1{
public static void main(int a){
System.out.println(a);
}

public static void main(String args[]){
System.out.println("main() method invoked");
main(10);
}
}
```

**Test it Now**

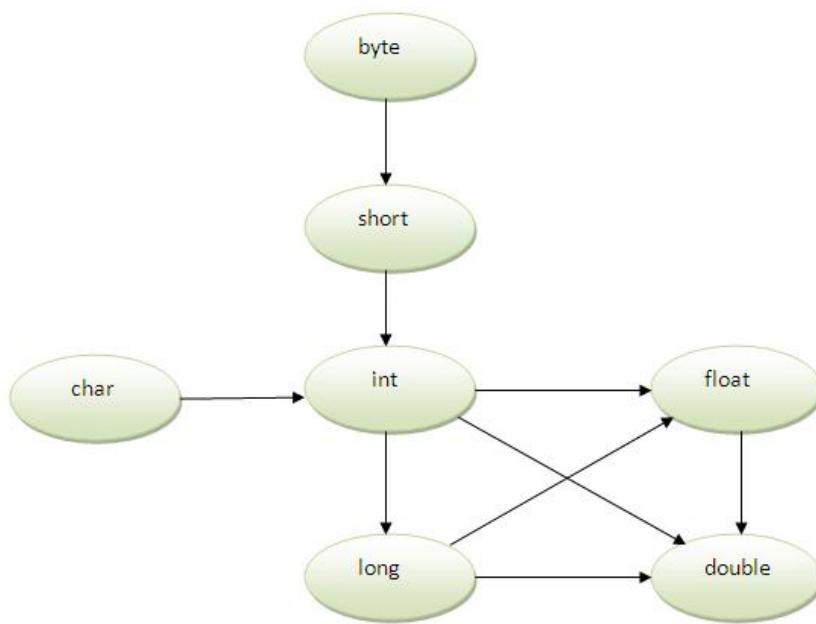
Output:main() method invoked

10

---

## Method Overloading and TypePromotion

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

```

class OverloadingCalculation1{
    void sum(int a,long b){System.out.println(a+b);}
    void sum(int a,int b,int c){System.out.println(a+b+c);}

    public static void main(String args[]){
        OverloadingCalculation1 obj=new OverloadingCalculation1();
        obj.sum(20,20); //now second int literal will be promoted to long
        obj.sum(20,20,20);
    }
}
  
```

**Test it Now**

Output:40  
60

## Example of Method Overloading with TypePromotion if matching found

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

```

class OverloadingCalculation2{
    void sum(int a,int b){System.out.println("int arg method invoked");}
    void sum(long a,long b){System.out.println("long arg method invoked");}

    public static void main(String args[]){
        OverloadingCalculation2 obj=new OverloadingCalculation2();
        obj.sum(20,20); //now int arg sum() method gets invoked
    }
}
  
```

**Test it Now**

Output:int arg method invoked

## Example of Method Overloading with TypePromotion in case ambiguity

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

```
class OverloadingCalculation3{  
    void sum(int a,long b){System.out.println("a method invoked");}  
    void sum(long a,int b){System.out.println("b method invoked");}  
  
    public static void main(String args[]){  
        OverloadingCalculation3 obj=new OverloadingCalculation3();  
        obj.sum(20,20); //now ambiguity  
    }  
}
```

**Test it Now**

Output:Compile Time Error

 **One type is not de-promoted implicitly for example double cannot be depromoted to any type implicitly.**

## Constructor in Java

**Constructor in java** is a *special type of method* that is used to initialize the object.

Java constructor is *invoked at the time of object creation*. It constructs the values i.e. provides data for the object that is why it is known as constructor.

### Rules for creating java constructor

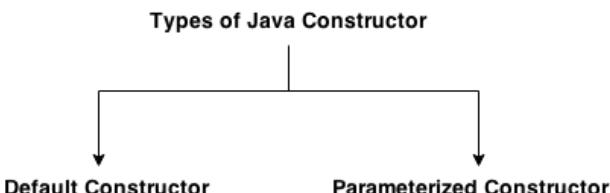
There are basically two rules defined for the constructor.

1. Constructor name must be same as its class name
2. Constructor must have no explicit return type

### Types of java constructors

There are two types of constructors:

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



## Java Default Constructor

A constructor that have no parameter is known as default constructor.

Syntax of default constructor:

```
<class_name>() {}
```

## Example of default constructor

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

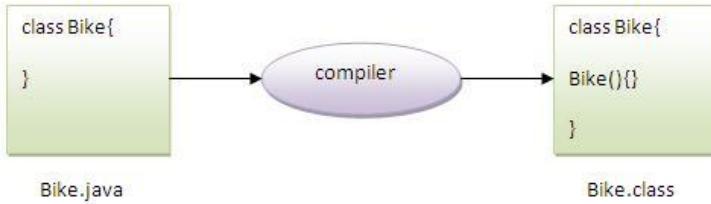
```
class Bike1{  
Bike1(){System.out.println("Bike is created");}  
public static void main(String args[]){  
Bike1 b=new Bike1();  
}  
}
```

**Test it Now**

Output:

```
Bike is created
```

 **Rule:** If there is no constructor in a class, compiler automatically creates a default constructor.



Q) What is the purpose of default constructor?

Default constructor provides the default values to the object like 0, null etc. depending on the type.

## Example of default constructor that displays the default values

```
class Student3{  
int id;  
String name;  
  
void display(){System.out.println(id+" "+name);}  
  
public static void main(String args[]){  
Student3 s1=new Student3();  
Student3 s2=new Student3();  
s1.display();  
s2.display();  
}
```

**Test it Now**

Output:

```
0 null  
0 null
```

**Explanation:** In the above class, you are not creating any constructor so compiler provides you a default constructor. Here 0 and null values are provided by default constructor.

## Java parameterized constructor

A constructor that have parameters is known as parameterized constructor.

### Why use 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 Student class that have two parameters. We can have any number of parameters in the constructor.

```
class Student4{  
    int id;  
    String name;  
  
    Student4(int i,String n){  
        id = i;  
        name = n;  
    }  
    void display(){System.out.println(id+" "+name);}  
  
    public static void main(String args[]){  
        Student4 s1 = new Student4(111,"Karan");  
        Student4 s2 = new Student4(222,"Aryan");  
        s1.display();  
        s2.display();  
    }  
}
```

**Test it Now**

Output:

```
111 Karan  
222 Aryan
```

## Constructor Overloading in Java

Constructor overloading is a technique in Java in which a class can have any number of constructors that differ in parameter lists. The compiler differentiates these constructors by taking into account the number of parameters in the list and their type.

### Example of Constructor Overloading

```
class Student5{  
    int id;  
    String name;  
    int age;  
    Student5(int i,String n){  
        id = i;  
        name = n;  
    }  
    Student5(int i,String n,int a){  
        id = i;  
        name = n;  
        age=a;  
    }
```

```

void display(){System.out.println(id+" "+name+" "+age);}

public static void main(String args[]){
Student5 s1 = new Student5(111,"Karan");
Student5 s2 = new Student5(222,"Aryan",25);
s1.display();
s2.display();
}
}

```

**Test it Now**

Output:

```

111 Karan 0
222 Aryan 25

```

## Difference between constructor and method in java

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

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

## Java Copy Constructor

There is no copy constructor in java. But, we can copy the values of one object to another like copy constructor in C++.

There are many ways to copy the values of one object into another in java. They are:

- By constructor
- By assigning the values of one object into another
- By clone() method of Object class

In this example, we are going to copy the values of one object into another using java constructor.

```

class Student6{
    int id;
    String name;
    Student6(int i,String n){
        id = i;
        name = n;
    }

    Student6(Student6 s){
        id = s.id;
        name = s.name;
    }
    void display(){System.out.println(id+" "+name);}

    public static void main(String args[]){
        Student6 s1 = new Student6(111,"Karan");
        Student6 s2 = new Student6(s1);
        s1.display();
        s2.display();
    }
}

```

**Test it Now**

Output:

```
111 Karan  
111 Karan
```

---

## Copying values without constructor

We can copy the values of one object into another by assigning the objects values to another object. In this case, there is no need to create the constructor.

```
class Student7{  
    int id;  
    String name;  
    Student7(int i, String n){  
        id = i;  
        name = n;  
    }  
    Student7(){  
    }  
    void display(){System.out.println(id+" "+name);}  
  
    public static void main(String args[]){  
        Student7 s1 = new Student7(111, "Karan");  
        Student7 s2 = new Student7();  
        s2.id=s1.id;  
        s2.name=s1.name;  
        s1.display();  
        s2.display();  
    }  
}
```

**Test it Now**

Output:

```
111 Karan  
111 Karan
```

---

## Q) Does constructor return any value?

**Ans:** yes, that is current class instance (You cannot use return type yet it returns a value).

---

## Can constructor perform other tasks instead of initialization?

Yes, like object creation, starting a thread, calling method etc. You can perform any operation in the constructor as you perform in the method.

## Java static keyword

The **static keyword** in java is used for memory management mainly. We can apply java static keyword with variables, methods, blocks and nested class. The static keyword belongs to the class than 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
- 

### 1) Java static variable

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

- The static variable can be used to refer 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 class area at the time of class loading.

## Advantage of static variable

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

## Understanding problem without static variable

```
class Student{  
    int rollno;  
    String name;  
    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.



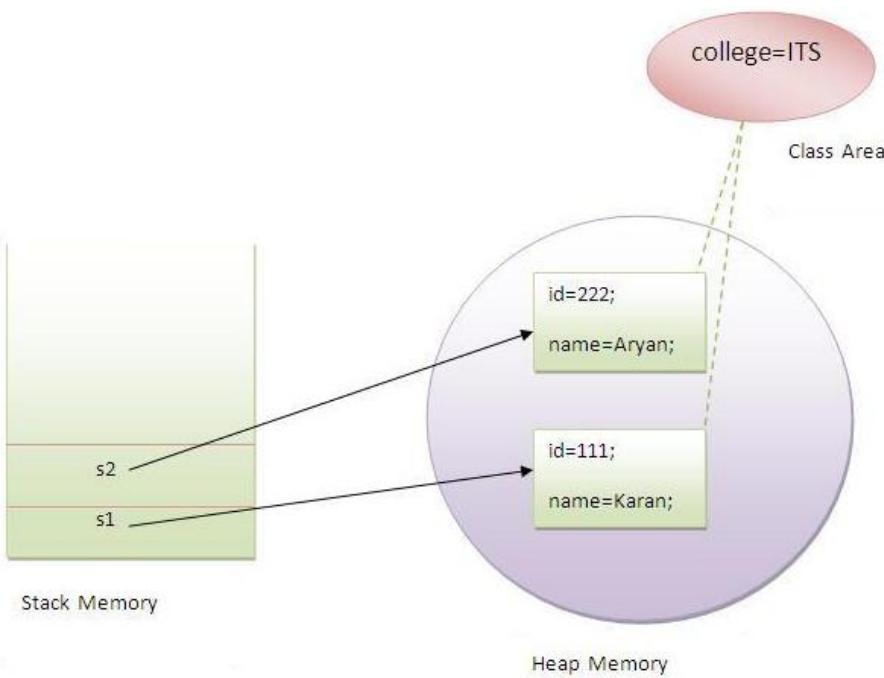
**Java static property is shared to all objects.**

## Example of static variable

```
//Program of static variable  
  
class Student8{  
    int rollno;  
    String name;  
    static String college ="ITS";  
  
    Student8(int r,String n){  
        rollno = r;  
        name = n;  
    }  
    void display (){System.out.println(rollno+" "+name+" "+college);}  
  
    public static void main(String args[]){  
        Student8 s1 = new Student8(111,"Karan");  
        Student8 s2 = new Student8(222,"Aryan");  
  
        s1.display();  
        s2.display();  
    }  
}
```

**Test it Now**

Output: 111 Karan ITS  
222 Aryan ITS



## Program of counter without static variable

In this example, we have created an instance variable named count which is incremented in the constructor. Since instance variable gets the memory at the time of object creation, each object will have the copy of the instance variable, if it is incremented, it won't reflect to other objects. So each objects will have the value 1 in the count variable.

```
class Counter{
int count=0;//will get memory when instance is created

Counter(){
count++;
System.out.println(count);
}

public static void main(String args[]){
Counter c1=new Counter();
Counter c2=new Counter();
Counter c3=new Counter();

}
}
```

**Test it Now**

Output:

```
1
1
```

## Program of counter by static variable

As we have mentioned above, static variable will get the memory only once, if any object changes the value of the static variable, it will retain its value.

```
class Counter2{
static int count=0;//will get memory only once and retain its value
```

```

Counter2(){
count++;
System.out.println(count);
}

public static void main(String args[]){
    Counter2 c1=new Counter2();
    Counter2 c2=new Counter2();
    Counter2 c3=new Counter2();

}
}

```

### Test it Now

Output:

1  
2  
3

## 2) Java static method

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

- A static method belongs to the class rather than 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 the value of it.

### Example of static method

```

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

class Student9{
    int rollno;
    String name;
    static String college = "ITS";

    static void change(){
        college = "BBDIT";
    }

    Student9(int r, String n){
        rollno = r;
        name = n;
    }

    void display (){System.out.println(rollno+" "+name+" "+college);}

    public static void main(String args[]){
        Student9.change();

        Student9 s1 = new Student9 (111,"Karan");
        Student9 s2 = new Student9 (222,"Aryan");
        Student9 s3 = new Student9 (333,"Sonoo");

        s1.display();
        s2.display();
        s3.display();
    }
}

```

### Test it Now

```
Output:111 Karan BBDIT
      222 Aryan BBDIT
      333 Sonoo BBDIT
```

## Another example of static method that performs normal calculation

```
//Program to get cube of a given number by static method
```

```
class Calculate{
    static int cube(int x){
        return x*x*x;
    }

    public static void main(String args[]){
        int result=Calculate.cube(5);
        System.out.println(result);
    }
}
```

### Test it Now

```
Output:125
```

## Restrictions for static method

There are two main restrictions for the static method. They are:

1. The static method can not use non static data member or call non-static method directly.
2. this and super cannot be used in static context.

```
class A{
    int a=40;//non static

    public static void main(String args[]){
        System.out.println(a);
    }
}
```

### Test it Now

```
Output:Compile Time Error
```

## Q) why java main method is static?

Ans) because object is not required to call static method if it were non-static method, jvm create object first then call main() method that will lead the problem of extra memory allocation.

## 3) Java static block

## this keyword in java

There can be a lot of usage of **java this keyword**. In java, this is a **reference variable** that refers to the current object.

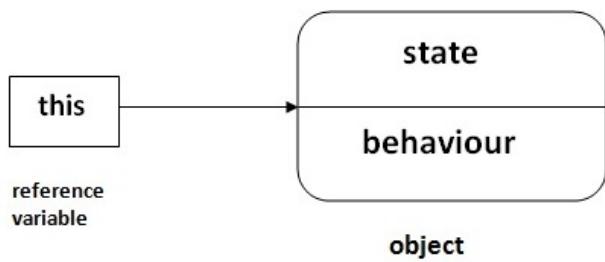
## Usage of java this keyword

Here is given the 6 usage of java this keyword.

1. this keyword can be used to refer current class instance variable.
2. this() can be used to invoke current class constructor.
3. this keyword can be used to invoke current class method (implicitly)

4. this can be passed as an argument in the method call.
5. this can be passed as argument in the constructor call.
6. this keyword can also be used to return the current class instance.

**Suggestion:** If you are beginner to java, lookup only two usage of this keyword.



## 1) The this keyword can be used to refer current class instance variable.

If there is ambiguity between the instance variable and parameter, this keyword resolves the problem of ambiguity.

### Understanding the problem without this keyword

Let's understand the problem if we don't use this keyword by the example given below:

```

class Student10{
    int id;
    String name;

    Student10(int id,String name){
        id = id;
        name = name;
    }
    void display(){System.out.println(id+" "+name);}

    public static void main(String args[]){
        Student10 s1 = new Student10(111,"Karan");
        Student10 s2 = new Student10(321,"Aryan");
        s1.display();
        s2.display();
    }
}

```

#### Test it Now

Output:0 null  
0 null

In the above example, parameter (formal arguments) and instance variables are same that is why we are using this keyword to distinguish between local variable and instance variable.

### Solution of the above problem by this keyword

```

//example of this keyword
class Student11{
    int id;
    String name;

    Student11(int id,String name){
        this.id = id;
        this.name = name;
    }
    void display(){System.out.println(id+" "+name);}
    public static void main(String args[]){
}

```

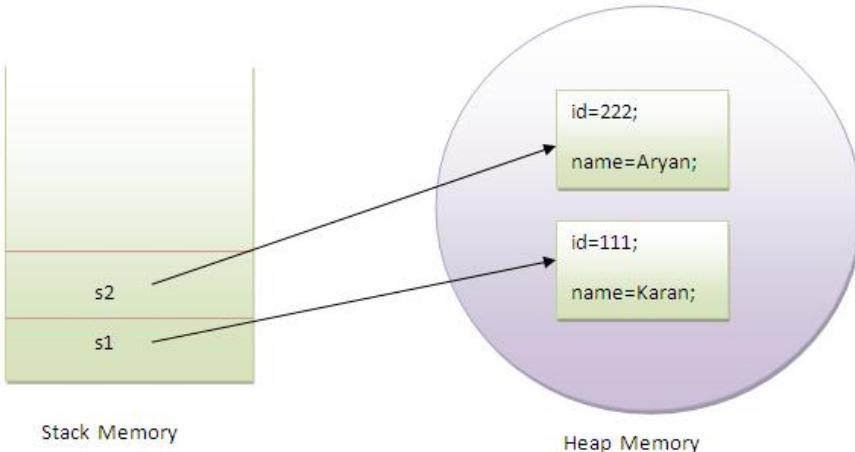
```

Student11 s1 = new Student11(111,"Karan");
Student11 s2 = new Student11(222,"Aryan");
s1.display();
s2.display();
}
}

```

**Test it Now**

Output  
111 Karan  
222 Aryan



If local variables(formal arguments) and instance variables are different, there is no need to use this keyword like in the following program:

**Program where this keyword is not required**

```

class Student12{
    int id;
    String name;

    Student12(int i,String n){
        id = i;
        name = n;
    }
    void display(){System.out.println(id+" "+name);}
    public static void main(String args[]){
        Student12 e1 = new Student12(111,"karan");
        Student12 e2 = new Student12(222,"Aryan");
        e1.display();
        e2.display();
    }
}

```

**Test it Now**

Output:  
111 Karan  
222 Aryan

## 2) this() can be used to invoked current class constructor.

The `this()` constructor call can be used to invoke the current class constructor (constructor chaining). This approach is better if you have many constructors in the class and want to reuse that constructor.

```
//Program of this() constructor call (constructor chaining)
```

```

class Student13{
    int id;
    String name;
    Student13(){System.out.println("default constructor is invoked");}

    Student13(int id,String name){
        this(); // it is used to invoke current class constructor.
        this.id = id;
        this.name = name;
    }
    void display(){System.out.println(id+" "+name);}

    public static void main(String args[]){
        Student13 e1 = new Student13(111,"karan");
        Student13 e2 = new Student13(222,"Aryan");
        e1.display();
        e2.display();
    }
}

```

### Test it Now

Output:

```

default constructor is invoked
default constructor is invoked
111 Karan
222 Aryan

```

## Where to use this() constructor call?

The this() constructor call should be used to reuse the constructor in the constructor. It maintains the chain between the constructors i.e. it is used for constructor chaining. Let's see the example given below that displays the actual use of this keyword.

```

class Student14{
    int id;
    String name;
    String city;

    Student14(int id,String name){
        this.id = id;
        this.name = name;
    }
    Student14(int id,String name,String city){
        this(id,name); // now no need to initialize id and name
        this.city=city;
    }
    void display(){System.out.println(id+" "+name+" "+city);}

    public static void main(String args[]){
        Student14 e1 = new Student14(111,"karan");
        Student14 e2 = new Student14(222,"Aryan","delhi");
        e1.display();
        e2.display();
    }
}

```

### Test it Now

Output:111 Karan null  
222 Aryan delhi



**Rule: Call to this() must be the first statement in constructor.**

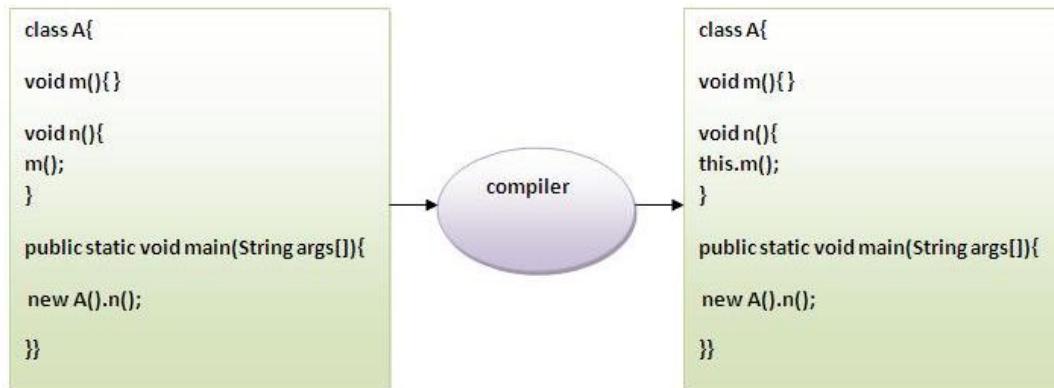
```
class Student15{  
    int id;  
    String name;  
    Student15(){System.out.println("default constructor is invoked");}  
  
    Student15(int id,String name){  
        id = id;  
        name = name;  
        this ();//must be the first statement  
    }  
    void display(){System.out.println(id+" "+name);}  
  
    public static void main(String args[]){  
        Student15 e1 = new Student15(111,"karan");  
        Student15 e2 = new Student15(222,"Aryan");  
        e1.display();  
        e2.display();  
    }  
}
```

#### Test it Now

Output:Compile Time Error

### 3)The this keyword can be used to invoke current class method (implicitly).

You may invoke the method of the current class by using the this keyword. If you don't use the this keyword, compiler automatically adds this keyword while invoking the method. Let's see the example



```
class S{  
    void m(){  
        System.out.println("method is invoked");  
    }  
    void n(){  
        this.m(); //no need because compiler does it for you.  
    }  
    void p(){  
        n(); //compiler will add this to invoke n() method as this.n()  
    }  
    public static void main(String args[]){  
        S s1 = new S();  
        s1.p();  
    }  
}
```

#### Test it Now

Output:method is invoked

---

#### 4) this keyword can be passed as an argument in the method.

The this keyword can also be passed as an argument in the method. It is mainly used in the event handling. Let's see the example:

```
class S2{  
    void m(S2 obj){  
        System.out.println("method is invoked");  
    }  
    void p(){  
        m(this);  
    }  
  
    public static void main(String args[]){  
        S2 s1 = new S2();  
        s1.p();  
    }  
}
```

**Test it Now**

Output:method is invoked

Application of this that can be passed as an argument:

In event handling (or) in a situation where we have to provide reference of a class to another one.

---

#### 5) The this keyword can be passed as argument in the constructor call.

We can pass the this keyword in the constructor also. It is useful if we have to use one object in multiple classes. Let's see the example:

```
class B{  
    A4 obj;  
    B(A4 obj){  
        this.obj=obj;  
    }  
    void display(){  
        System.out.println(obj.data);//using data member of A4 class  
    }  
}  
  
class A4{  
    int data=10;  
    A4(){  
        B b=new B(this);  
        b.display();  
    }  
    public static void main(String args[]){  
        A4 a=new A4();  
    }  
}
```

**Test it Now**

Output:10

---

#### 6) The this keyword can be used to return current class instance.

We can return the this keyword as an statement from the method. In such case, return type of the method must be the class type (non-primitive). Let's see the example:

## Syntax of this that can be returned as a statement

```
return_type method_name(){  
    return this;  
}
```

## Example of this keyword that you return as a statement from the method

```
class A{  
A getA(){  
    return this;  
}  
void msg(){System.out.println("Hello java");}  
}
```

```
class Test1{  
public static void main(String args[]){  
new A().getA().msg();  
}  
}
```

**Test it Now**

Output:Hello java

## Proving this keyword

Let's prove that this keyword refers to the current class instance variable. In this program, we are printing the reference variable and this, output of both variables are same.

```
class A5{  
void m(){  
System.out.println(this); //prints same reference ID  
}  
  
public static void main(String args[]){  
A5 obj=new A5();  
System.out.println(obj); //prints the reference ID  
  
obj.m();  
}  
}
```

**Test it Now**

Output:A5@22b3ea59  
A5@22b3ea59

## Inheritance in Java

**Inheritance in java** is a mechanism in which one object acquires all the properties and behaviors of parent object.

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

Inheritance represents the **IS-A relationship**, also known as *parent-child* relationship.

## Why use inheritance in java

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

## Syntax of Java Inheritance

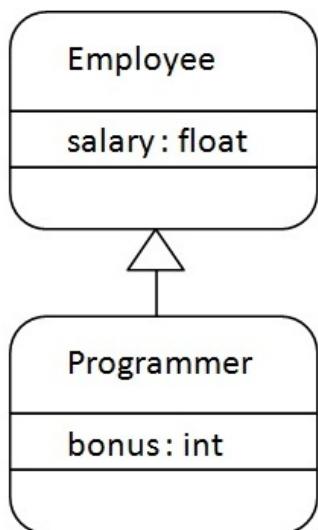
```
class Subclass-name extends Superclass-name
{
    //methods and fields
}
```

The **extends keyword** indicates that you are making a new class that derives from an existing class.

In the terminology of Java, a class that is inherited is called a super class. The new class is called a subclass.

---

## Understanding the simple example of inheritance



As displayed in the above figure, Programmer is the subclass and Employee is the superclass. Relationship between two classes is **Programmer IS-A Employee**. It means that Programmer is a type of Employee.

```
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);
    }
}
```

**Test it Now**

```
Programmer salary is:40000.0
Bonus of programmer is:10000
```

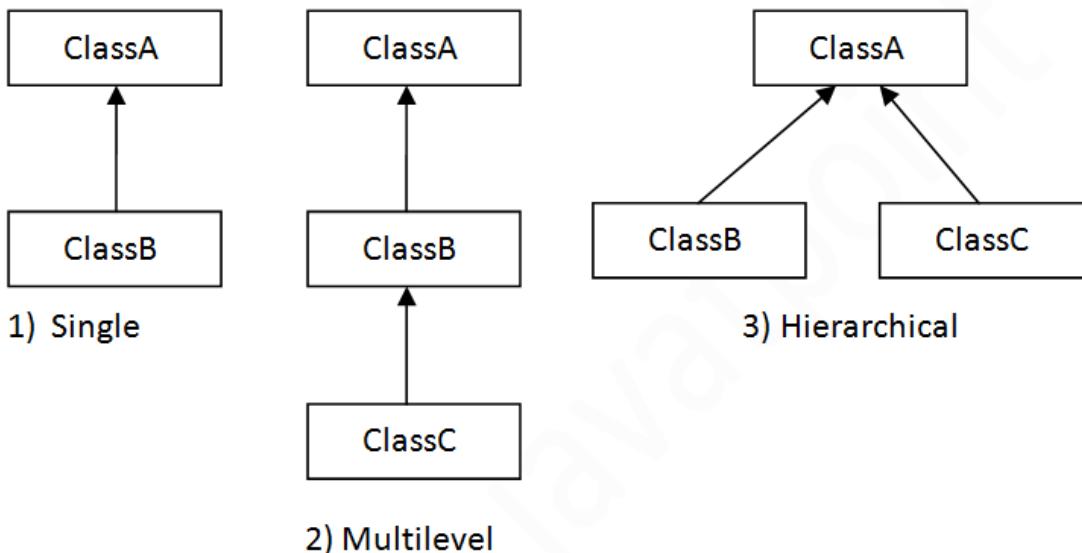
In the above example, Programmer object can access the field of own class as well as of Employee class i.e. code reusability.

---

## Types of inheritance in java

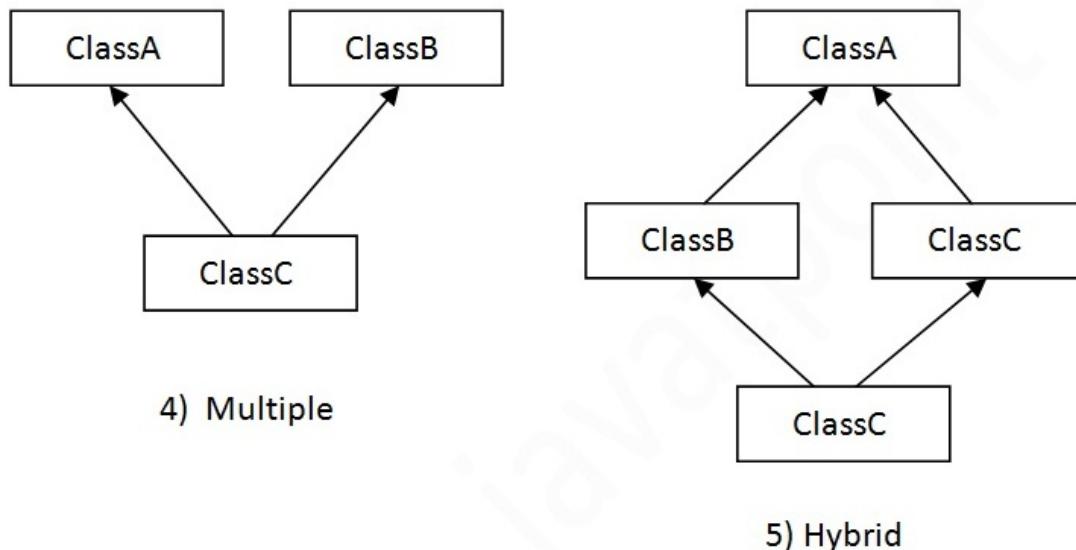
On the basis of class, there can be three types of inheritance in java: single, multilevel and hierarchical.

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



**Note:** *Multiple inheritance is not supported in java through class.*

When a class extends multiple classes i.e. known as multiple inheritance. For Example:



## Q) Why multiple inheritance is not supported in java?

To reduce the complexity and simplify the language, multiple inheritance is not supported in java.

Consider a scenario where A, B and C are three classes. The C class inherits A and B classes. If A and B classes have same method and you call it from child class object, there will be ambiguity to call method of A or B class.

Since compile time errors are better than runtime errors, java renders compile time error if you inherit 2 classes. So whether you have same method or different, there will be compile time error now.

```

class A{
void msg(){System.out.println("Hello");}
}
class B{
void msg(){System.out.println("Welcome");}
}
class C extends A,B{//suppose if it were

Public Static void main(String args[]){
C obj=new C();

```

```

    obj.msg(); //Now which msg() method would be invoked?
}
}

```

### Test it Now

Compile Time Error

## Aggregation in Java

If a class have an entity reference, it is known as Aggregation. Aggregation represents HAS-A relationship.

Consider a situation, Employee object contains many informations such as id, name, emailId etc. It contains one more object named address, which contains its own informations such as city, state, country, zipcode etc. as given below.

```

class Employee{
    int id;
    String name;
    Address address; //Address is a class
    ...
}

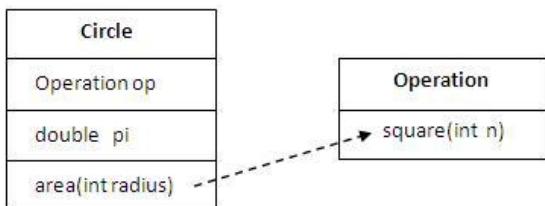
```

In such case, Employee has an entity reference address, so relationship is Employee HAS-A address.

### Why use Aggregation?

- For Code Reusability.

## Simple Example of Aggregation



In this example, we have created the reference of Operation class in the Circle class.

```

class Operation{
    int square(int n){
        return n*n;
    }
}

class Circle{
    Operation op; //aggregation
    double pi=3.14;

    double area(int radius){
        op=new Operation();
        int rsquare=op.square(radius); //code reusability (i.e. delegates the method call).
        return pi*rsquare;
    }
}

```

```

public static void main(String args[]){
    Circle c=new Circle();
    double result=c.area(5);
    System.out.println(result);
}
}

```

## Method Overriding in Java

If subclass (child class) has the same method as declared in the parent class, it is known as **method overriding in java**.

In other words, If subclass provides the specific implementation of the method that has been provided by one of its parent class, it is known as method overriding.

### Usage of Java Method Overriding

- Method overriding is used to provide specific implementation of a method that is already provided by its super class.
- Method overriding is used for runtime polymorphism

### Rules for Java Method Overriding

1. method must have same name as in the parent class
2. method must have same parameter as in the parent class.
3. must be 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.

```

class Vehicle{
    void run(){System.out.println("Vehicle is running");}
}

class Bike extends Vehicle{

    public static void main(String args[]){
        Bike obj = new Bike();
        obj.run();
    }
}

```

**Test it Now**

Output:Vehicle is running

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

### Example of 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 is same and there is IS-A relationship between the classes, so there is method overriding.

```

class Vehicle{
    void run(){System.out.println("Vehicle is running");}
}

class Bike2 extends Vehicle{
    void run(){System.out.println("Bike is running safely");}
}

public static void main(String args[]){
    Bike2 obj = new Bike2();
    obj.run();
}

```

```

    }
}

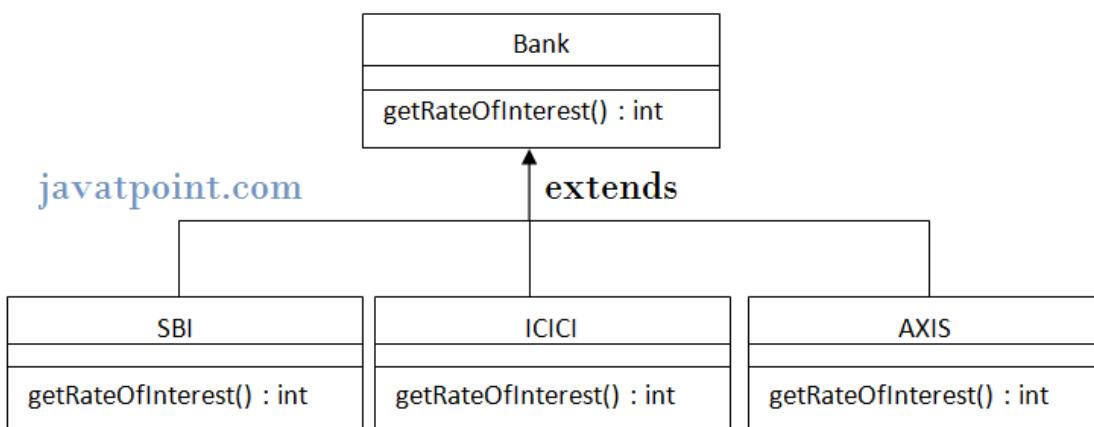
Test it Now

Output:Bike is running safely

```

## Real example of Java Method Overriding

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



```

class Bank{
int getRateOfInterest(){return 0;}
}

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

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

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

class Test2{
public static void main(String args[]){
SBI s=new SBI();
ICICI i=new ICICI();
AXIS a=new AXIS();
System.out.println("SBI Rate of Interest: "+s.getRateOfInterest());
System.out.println("ICICI Rate of Interest: "+i.getRateOfInterest());
System.out.println("AXIS Rate of Interest: "+a.getRateOfInterest());
}
}

```

**Test it Now**

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

## Can we override static method?

No, static method cannot be overridden. It can be proved by runtime polymorphism, so we will learn it later.

## Why we cannot override static method?

because static method is bound with class whereas instance method is bound with object. Static belongs to class area and instance belongs to heap area.

## Can we override java main method?

No, because main is a static method.

## Difference between method Overloading and Method Overriding in java

### Covariant Return Type

The covariant return type specifies that the return type may vary in the same direction as the subclass.

Before Java5, it was not possible to override any method by changing the return type. But now, since Java5, it is possible to override method by changing the return type if subclass overrides any method whose return type is Non-Primitive but it changes its return type to subclass type. Let's take a simple example:



**Note:** If you are beginner to java, skip this topic and return to it after OOPs concepts.

### Simple example of Covariant Return Type

```
class A{  
A get(){return this;}  
}  
  
class B1 extends A{  
B1 get(){return this;}  
void message(){System.out.println("welcome to covariant return type");}  
  
public static void main(String args[]){  
new B1().get().message();  
}  
}
```

**Test it Now**

Output:welcome to covariant return type

As you can see in the above example, the return type of the get() method of A class is A but the return type of the get() method of B class is B. Both methods have different return type but it is method overriding. This is known as covariant return type.

## super keyword in java

The **super** keyword in java is a reference variable that is used to refer immediate parent class object.

Whenever you create the instance of subclass, an instance of parent class is created implicitly i.e. referred by super reference variable.

### Usage of java super Keyword

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

#### 1) super is used to refer immediate parent class instance variable.

### **Problem without super keyword**

```
class Vehicle{
    int speed=50;
}

class Bike3 extends Vehicle{
    int speed=100;
    void display(){
        System.out.println(speed); //will print speed of Bike
    }
    public static void main(String args[]){
        Bike3 b=new Bike3();
        b.display();
    }
}
```

**Test it Now**

Output:100

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

### **Solution by super keyword**

```
//example of super keyword

class Vehicle{
    int speed=50;
}

class Bike4 extends Vehicle{
    int speed=100;

    void display(){
        System.out.println(super.speed); //will print speed of Vehicle now
    }
    public static void main(String args[]){
        Bike4 b=new Bike4();
        b.display();
    }
}
```

**Test it Now**

Output:50

## **2) super is used to invoke parent class constructor.**

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

```
class Vehicle{
    Vehicle(){System.out.println("Vehicle is created");}
}

class Bike5 extends Vehicle{
    Bike5(){
        super(); //will invoke parent class constructor
        System.out.println("Bike is created");
    }
}
```

```

public static void main(String args[]){
    Bike5 b=new Bike5();
}

}

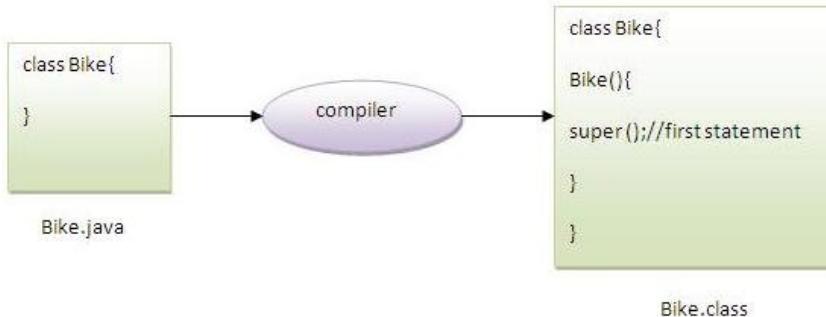
```

### Test it Now

Output:Vehicle is created  
Bike is created



**Note: super() is added in each class constructor automatically by compiler.**



As we know well that default constructor is provided by 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, compiler will provide super() as the first statement of the constructor.

Another example of super keyword where super() is provided by the compiler implicitly.

```

class Vehicle{
    Vehicle(){System.out.println("Vehicle is created");}
}

```

```

class Bike6 extends Vehicle{
    int speed;
    Bike6(int speed){
        this.speed=speed;
        System.out.println(speed);
    }
    public static void main(String args[]){
        Bike6 b=new Bike6(10);
    }
}

```

### Test it Now

Output:Vehicle is created  
10

## 3) super can be used to invoke parent class method

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

```

class Person{
    void message(){System.out.println("welcome");}
}

class Student16 extends Person{
    void message(){System.out.println("welcome to java");}

    void display(){
}

```

```

message();//will invoke current class message() method
super.message();//will invoke parent class message() method
}

public static void main(String args[]){
Student16 s=new Student16();
s.display();
}
}

```

### Test it Now

Output:welcome to java  
welcome

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

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

### Program in case super is not required

```

class Person{
void message(){System.out.println("welcome");}
}

class Student17 extends Person{

void display(){
message();//will invoke parent class message() method
}

public static void main(String args[]){
Student17 s=new Student17();
s.display();
}
}

```

### Test it Now

Output:welcome

## Instance initializer block:

**Instance Initializer block** is used to initialize the instance data member. It runs each time when object of the class is created.

The initialization of the instance variable can be directly but there can be performed extra operations while initializing the instance variable in the instance initializer block.

**Que) What is the use of instance initializer block while we can directly assign a value in instance data member? For example:**

```

class Bike{
    int speed=100;
}

```

## Why use instance initializer block?

Suppose I have to perform some operations while assigning value to instance data member e.g. a for loop to fill a complex array or error handling etc.

---

### Example of instance initializer block

Let's see the simple example of instance initializer block the performs initialization.

```
class Bike7{  
    int speed;  
  
    Bike7(){System.out.println("speed is "+speed);}  
  
    {speed=100;}  
  
    public static void main(String args[]){  
        Bike7 b1=new Bike7();  
        Bike7 b2=new Bike7();  
    }  
}
```

**Test it Now**

```
Output:speed is 100  
      speed is 100
```

There are three places in java where you can perform operations:

1. method
  2. constructor
  3. block
- 

### What is invoked firstly instance initializer block or constructor?

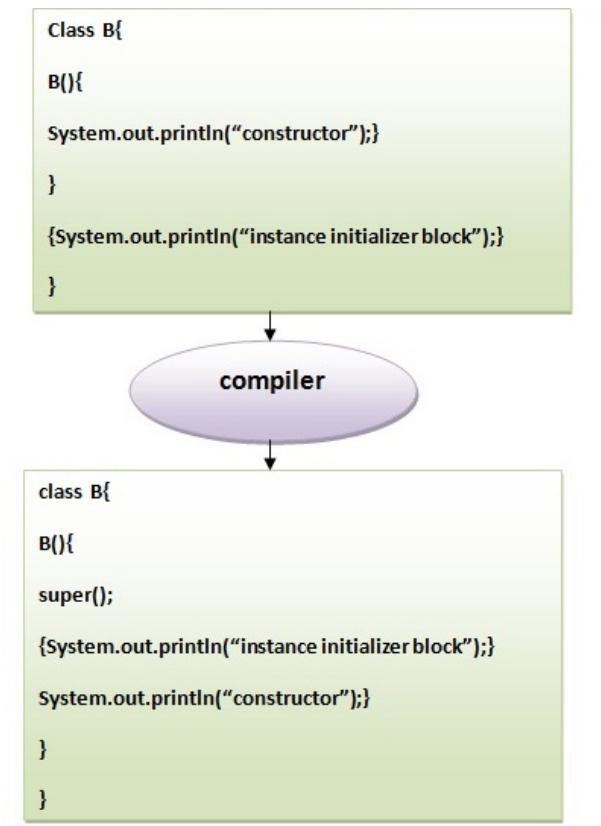
```
class Bike8{  
    int speed;  
  
    Bike8(){System.out.println("constructor is invoked");}  
  
    {System.out.println("instance initializer block invoked");}  
  
    public static void main(String args[]){  
        Bike8 b1=new Bike8();  
        Bike8 b2=new Bike8();  
    }  
}
```

**Test it Now**

```
Output:instance initializer block invoked  
      constructor is invoked  
      instance initializer block invoked  
      constructor is invoked
```

In the above example, it seems that instance initializer block is firstly invoked but NO. Instance initializer block is invoked at the time of object creation. The java compiler copies the instance initializer block in the constructor after the first statement super(). So firstly, constructor is invoked. Let's understand it by the figure given below:

**Note:** The java compiler copies the code of instance initializer block in every constructor.



## Rules for instance initializer block :

There are mainly three rules for the instance initializer block. They are as follows:

1. The instance initializer block is created when instance of the class is created.
2. The instance initializer block is invoked after the parent class constructor is invoked (i.e. after super() constructor call).
3. The instance initializer block comes in the order in which they appear.

## Program of instance initializer block that is invoked after super()

```

class A{
A(){
System.out.println("parent class constructor invoked");
}
}

class B2 extends A{
B2(){
super();
System.out.println("child class constructor invoked");
}

{System.out.println("instance initializer block is invoked");}

```

```

public static void main(String args[]){
B2 b=new B2();
}
}

```

**Test it Now**

```

Output:parent class constructor invoked
       instance initializer block is invoked
       child class constructor invoked

```

---

## Another example of instance block

```

class A{

```

```

A(){

System.out.println("parent class constructor invoked");
}

}

class B3 extends A{

B3(){

super();
System.out.println("child class constructor invoked");
}

B3(int a){

super();
System.out.println("child class constructor invoked "+a);
}

{System.out.println("instance initializer block is invoked");}

public static void main(String args[]){

B3 b1=new B3();
B3 b2=new B3(10);
}
}

```

#### Test it Now

Output:parent class constructor invoked  
 instance initializer block is invoked  
 child class constructor invoked  
 parent class constructor invoked  
 instance initializer block is invoked  
 child class constructor invoked 10

## Final Keyword In Java

The **final keyword** in java is used to restrict the user. The java final keyword can be used in many context. Final can be:

1. variable
2. method
3. class

The final keyword can be applied with the variables, a final variable that have no value it is called 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. We will have detailed learning of these. Let's first learn the basics of final keyword.

### 1) Java 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 speedlimit, 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 Bike9{

final int speedlimit=90;//final variable

void run(){

```

#### Java Final Keyword

- ⇒ Stop Value Change
- ⇒ Stop Method Overriding
- ⇒ Stop Inheritance

```
speedlimit=400;  
}  
public static void main(String args[]){  
Bike9 obj=new Bike9();  
obj.run();  
}  
}//end of class
```

**Test it Now**

Output:Compile Time Error

## 2) Java final method

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

### Example of final method

```
class Bike{  
final void run(){System.out.println("running");}  
}  
  
class Honda extends Bike{  
void run(){System.out.println("running safely with 100kmph");}  
  
public static void main(String args[]){  
Honda honda= new Honda();  
honda.run();  
}  
}
```

**Test it Now**

Output:Compile Time Error

## 3) Java final class

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

### Example of final class

```
final class Bike{  
  
class Hondal extends Bike{  
void run(){System.out.println("running safely with 100kmph");}  
  
public static void main(String args[]){  
Hondal honda= new Honda();  
honda.run();  
}  
}
```

**Test it Now**

Output:Compile Time Error

## Q) Is final method inherited?

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

```
class Bike{
```

```

final void run(){System.out.println("running...");}
}
class Honda2 extends Bike{
    public static void main(String args[]){
        new Honda2().run();
    }
}

```

**Test it Now**

Output:running...

---

## Q) What is blank or uninitialized final variable?

A final variable that is not initialized at the time of declaration is known as blank final variable.

If you want to create a variable that is initialized at the time of creating object and once initialized may not be changed, it is useful. For example PAN CARD number of an employee.

It can be initialized only in constructor.

## Example of blank final variable

```

class Student{
    int id;
    String name;
    final String PAN_CARD_NUMBER;
    ...
}

```

## Que) Can we initialize blank final variable?

Yes, but only in constructor. For example:

```

class Bike10{
    final int speedlimit;//blank final variable

    Bike10(){
        speedlimit=70;
        System.out.println(speedlimit);
    }

    public static void main(String args[]){
        new Bike10();
    }
}

```

**Test it Now**

Output:70

---

## static blank final variable

A static final variable that is not initialized at the time of declaration is known as static blank final variable. It can be initialized only in static block.

## Example of static blank final variable

```

class A{
    static final int data;//static blank final variable
    static{ data=50;}
    public static void main(String args[]){

```

```
        System.out.println(A.data);
    }
}
```

## Q) What is final parameter?

If you declare any parameter as final, you cannot change the value of it.

```
class Bike11{
    int cube(final int n){
        n=n+2;//can't be changed as n is final
        n*n*n;
    }
    public static void main(String args[]){
        Bike11 b=new Bike11();
        b.cube(5);
    }
}
```

**Test it Now**

Output:Compile Time Error

## Q) Can we declare a constructor final?

No, because constructor is never inherited.

# Polymorphism in Java

**Polymorphism in java** is a concept by which we can perform a *single action by different ways*. Polymorphism is derived from 2 greek words: poly and morphs. The word "poly" means many and "morphs" means forms. So polymorphism means many forms.

There are two types of polymorphism in java: compile time polymorphism and runtime polymorphism. We can perform polymorphism in java by method overloading and method overriding.

If you overload static method in java, it is the example of compile time polymorphism. Here, we will focus on runtime polymorphism in java.

## Runtime Polymorphism in Java

**Runtime polymorphism or Dynamic Method Dispatch** is a process in which a call to an overridden method is resolved at runtime rather than compile-time.

In this process, an overridden method is called through the reference variable of a superclass. The determination of the method to be called is based on the object being referred to by the reference variable.

Let's first understand the upcasting before Runtime Polymorphism.

### Upcasting

When reference variable of Parent class refers to the object of Child class, it is known as upcasting. For example:



```
class A{}
class B extends A{}
```

```
A a=new B(); //upcasting
```

## Example of Java Runtime Polymorphism

In this example, we are creating two classes Bike and Splender. Splender class extends Bike class and overrides its run() method. We are calling the run method by the reference variable of Parent class. Since it refers to the subclass object and subclass method overrides the Parent class method, subclass method is invoked at runtime.

Since method invocation is determined by the JVM not compiler, it is known as runtime polymorphism.

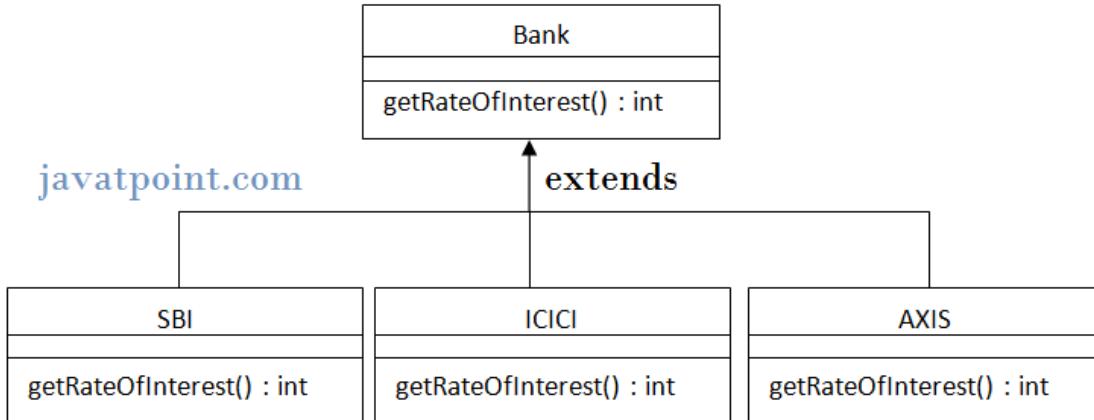
```
class Bike{  
    void run(){System.out.println("running");}  
}  
  
class Splender extends Bike{  
    void run(){System.out.println("running safely with 60km");}  
  
    public static void main(String args[]){  
        Bike b = new Splender(); //upcasting  
        b.run();  
    }  
}
```

**Test it Now**

Output:running safely with 60km.

## Real example of Java Runtime Polymorphism

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



Note: It is also given in method overriding but there was no upcasting.

```
class Bank{  
    int getRateOfInterest(){return 0;}  
}  
  
class SBI extends Bank{  
    int getRateOfInterest(){return 8;}  
}  
  
class ICICI extends Bank{  
    int getRateOfInterest(){return 7;}  
}  
  
class AXIS extends Bank{  
    int getRateOfInterest(){return 9;}  
}  
  
class Test3{
```

```

public static void main(String args[]){
    Bank b1=new SBI();
    Bank b2=new ICICI();
    Bank b3=new AXIS();
    System.out.println("SBI Rate of Interest: "+b1.getRateOfInterest());
    System.out.println("ICICI Rate of Interest: "+b2.getRateOfInterest());
    System.out.println("AXIS Rate of Interest: "+b3.getRateOfInterest());
}
}

```

**Test it Now**

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

## Java Runtime Polymorphism with data member

Method is overridden not the datamembers, so runtime polymorphism can't be achieved by data members.

In the example given below, both the classes have a datamember speedlimit, we are accessing the datamember by the reference variable of Parent class which refers to the subclass object. Since we are accessing the datamember which is not overridden, hence it will access the datamember of Parent class always.

 **Rule:** Runtime polymorphism can't be achieved by data members.

```

class Bike{
    int speedlimit=90;
}

class Honda3 extends Bike{
    int speedlimit=150;

    public static void main(String args[]){
        Bike obj=new Honda3();
        System.out.println(obj.speedlimit); //90
    }
}

```

**Test it Now**

Output:90

## Java Runtime Polymorphism with Multilevel Inheritance

Let's see the simple example of Runtime Polymorphism with multilevel inheritance.

```

class Animal{
    void eat(){System.out.println("eating");}
}

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

class BabyDog extends Dog{
    void eat(){System.out.println("drinking milk");}
}

public static void main(String args[]){
    Animal a1,a2,a3;
    a1=new Animal();
    a2=new Dog();
    a3=new BabyDog();
}

```

```
a3=new BabyDog();
```

```
a1.eat();
a2.eat();
a3.eat();
}
}
```

**Test it Now**

```
Output: eating
      eating fruits
      drinking Milk
```

## Try for Output

```
class Animal{
void eat(){System.out.println("animal is eating...");}
}

class Dog extends Animal{
void eat(){System.out.println("dog is eating...");}
}

class BabyDog1 extends Dog{
public static void main(String args[]){
Animal a=new BabyDog1();
a.eat();
}}}
```

**Test it Now**

```
Output: Dog is eating
```

Since, BabyDog is not overriding the eat() method, so eat() method of Dog class is invoked.

## Static Binding and Dynamic Binding

Connecting a method call to the method body is known as binding.

There are two types of binding

1. static binding (also known as early binding).
2. dynamic binding (also known as late binding).

## Understanding Type

Let's understand the type of instance.

### 1) variables have a type

Each variable has a type, it may be primitive and non-primitive.

```
int data=30;
```

Here data variable is a type of int.

### 2) References have a type



```
class Dog{
    public static void main(String args[]){
        Dog d1;//Here d1 is a type of Dog
    }
}
```

### 3) Objects have a type

An object is an instance of particular java class, but it is also an instance of its superclass.

```
class Animal{}

class Dog extends Animal{
    public static void main(String args[]){
        Dog d1=new Dog();
    }
}
```

Here d1 is an instance of Dog class, but it is also an instance of Animal.

---

### static binding

When type of the object is determined at compiled time (by the compiler), it is known as static binding.

If there is any private, final or static method in a class, there is static binding.

### Example of static binding

```
class Dog{
    private void eat(){System.out.println("dog is eating...");}

    public static void main(String args[]){
        Dog d1=new Dog();
        d1.eat();
    }
}
```

---

### Dynamic binding

When type of the object is determined at run-time, it is known as dynamic binding.

### Example of dynamic binding

```
class Animal{
    void eat(){System.out.println("animal is eating...");}
}

class Dog extends Animal{
    void eat(){System.out.println("dog is eating...");}

    public static void main(String args[]){
        Animal a=new Dog();
        a.eat();
    }
}
```

**Test it Now**

Output: dog is eating...

In the above example object type cannot be determined by the compiler, because the instance of Dog is also an instance of Animal. So

compiler doesn't know its type, only its base type.

## Java instanceof

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.

### Simple example of java instanceof

Let's see the simple example of instance operator where it tests the current class.

```
class Simple1{  
    public static void main(String args[]){  
        Simple1 s=new Simple1();  
        System.out.println(s instanceof Simple);//true  
    }  
}
```

**Test it Now**

Output:true

---

An object of subclass type is also a type of parent class. For example, if Dog extends Animal then object of Dog can be referred by either Dog or Animal class.

### Another example of java instanceof operator

```
class Animal{}  
class Dog1 extends Animal{//Dog inherits Animal  
  
    public static void main(String args[]){  
        Dog1 d=new Dog1();  
        System.out.println(d instanceof Animal);//true  
    }  
}
```

**Test it Now**

Output:true

---

### instanceof in java with a variable that have null value

If we apply instanceof operator with a variable that have null value, it returns false. Let's see the example given below where we apply instanceof operator with the variable that have null value.

```
class Dog2{  
    public static void main(String args[]){  
        Dog2 d=null;  
        System.out.println(d instanceof Dog2);//false  
    }  
}
```

**Test it Now**

Output:false

---

### Downcasting with java instanceof operator

When Subclass type refers to the object of Parent class, it is known as downcasting. If we perform it directly, compiler gives Compilation error. If you perform it by typecasting, ClassCastException is thrown at runtime. But if we use instanceof operator, downcasting is possible.

```
Dog d=new Animal(); //Compilation error
```

If we perform downcasting by typecasting, ClassCastException is thrown at runtime.

```
Dog d=(Dog)new Animal();
//Compiles successfully but ClassCastException is thrown at runtime
```

## Possibility of downcasting with instanceof

Let's see the example, where downcasting is possible by instanceof operator.

```
class Animal { }

class Dog3 extends Animal {
    static void method(Animal a) {
        if(a instanceof Dog3){
            Dog3 d=(Dog3)a; //downcasting
            System.out.println("ok downcasting performed");
        }
    }
}

public static void main (String [] args) {
    Animal a=new Dog3();
    Dog3.method(a);
}
```

**Test it Now**

Output:ok downcasting performed

## Downcasting without the use of java instanceof

Downcasting can also be performed without the use of instanceof operator as displayed in the following example:

```
class Animal { }

class Dog4 extends Animal {
    static void method(Animal a) {
        Dog4 d=(Dog4)a; //downcasting
        System.out.println("ok downcasting performed");
    }
}

public static void main (String [] args) {
    Animal a=new Dog4();
    Dog4.method(a);
}
```

**Test it Now**

Output:ok downcasting performed

Let's take closer look at this, actual object that is referred by a, is an object of Dog class. So if we downcast it, it is fine. But what will happen if we write:

```
Animal a=new Animal();
Dog.method(a);
//Now ClassCastException but not in case of instanceof operator
```

## Understanding Real use of instanceof in java

Let's see the real use of instanceof keyword by the example given below.

```
interface Printable{}  
class A implements Printable{  
public void a(){System.out.println("a method");}  
}  
class B implements Printable{  
public void b(){System.out.println("b method");}  
}  
  
class Call{  
void invoke(Printable p){//upcasting  
if(p instanceof A){  
A a=(A)p;//Downcasting  
a.a();  
}  
if(p instanceof B){  
B b=(B)p;//Downcasting  
b.b();  
}  
}  
}//end of Call class  
  
class Test4{  
public static void main(String args[]){  
Printable p=new B();  
Call c=new Call();  
c.invoke(p);  
}  
}
```

**Test it Now**

Output: b method

## Abstract class in Java

A class that is declared with abstract keyword, is known as abstract class in java. It can have abstract and non-abstract methods (method with body).

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

## Abstraction in Java

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

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

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

### Ways to achieve Abstraction

There are two ways to achieve abstraction in java

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

## Abstract class in Java

A class that is declared as abstract is known as **abstract class**. It needs to be extended and its method implemented. It cannot be instantiated.

### Example abstract class

```
abstract class A{}
```

### abstract method

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

### Example abstract method

```
abstract void printStatus(); //no body and abstract
```

### Example of abstract class that has abstract method

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

```
abstract class Bike{
    abstract void run();
}

class Honda4 extends Bike{
void run(){System.out.println("running safely..");}

public static void main(String args[]){
    Bike obj = new Honda4();
    obj.run();
}
}
```

**Test it Now**

```
running safely..
```

### Understanding the real scenario of abstract class

In this example, Shape is the abstract class, its implementation is provided by the Rectangle and Circle classes. Mostly, we don't know about the implementation class (i.e. hidden to the end user) and object of the implementation class is provided by the **factory method**.

A **factory method** is the method that returns the instance of the class. We will learn about the factory method later.

In this example, if you create the instance of Rectangle class, draw() method of Rectangle class will be invoked.

*File: TestAbstraction1.java*

```
abstract class Shape{
abstract void draw();
}

//In real scenario, implementation is provided by others i.e. unknown by end user
class Rectangle extends Shape{
void draw(){System.out.println("drawing rectangle");}
}

class Circle1 extends Shape{
void draw(){System.out.println("drawing circle");}
}

//In real scenario, method is called by programmer or user
class TestAbstraction1{
```

```
public static void main(String args[]){
Shape s=new Circle1(); //In real scenario, object is provided through method e.g. getShape() method
s.draw();
}
```

**Test it Now**

drawing circle

---

## Another example of abstract class in java

*File: TestBank.java*

```
abstract class Bank{
abstract int getRateOfInterest();
}

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

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

class TestBank{
public static void main(String args[]){
Bank b=new SBI(); //if object is PNB, method of PNB will be invoked
int interest=b.getRateOfInterest();
System.out.println("Rate of Interest is: "+interest+" %");
}}
```

**Test it Now**

Rate of Interest is: 7 %

---

## Abstract class having constructor, data member, methods etc.

An abstract class can have data member, abstract method, method body, constructor and even main() method.

*File: TestAbstraction2.java*

```
//example of abstract class that have method body
abstract class Bike{
Bike(){System.out.println("bike is created");}
abstract void run();
void changeGear(){System.out.println("gear changed");}
}

class Honda extends Bike{
void run(){System.out.println("running safely..");}
}

class TestAbstraction2{
public static void main(String args[]){
Bike obj = new Honda();
obj.run();
obj.changeGear();
}
}
```

**Test it Now**

bike is created

```
running safely..  
gear changed
```

 **Rule:** If there is any abstract method in a class, that class must be abstract.

```
class Bike12{  
abstract void run();  
}
```

**Test it Now**

compile time error

 **Rule:** If you are extending any abstract class that have abstract method, you must either provide the implementation of the method or make this class abstract.

## Another real scenario of abstract class

The abstract class can also be used to provide some implementation of the interface. In such case, the end user may not be forced to override all the methods of the interface.

 **Note:** If you are beginner to java, learn interface first and skip this example.

```
interface A{  
void a();  
void b();  
void c();  
void d();  
}  
  
abstract class B implements A{  
public void c(){System.out.println("I am C");}  
}  
  
class M extends B{  
public void a(){System.out.println("I am a");}  
public void b(){System.out.println("I am b");}  
public void d(){System.out.println("I am d");}  
}  
  
class Test5{  
public static void main(String args[]){  
A a=new M();  
a.a();  
a.b();  
a.c();  
a.d();  
}}
```

**Test it Now**

Output:I am a  
I am b  
I am c  
I am d

# Interface in Java

An **interface in java** is a blueprint of a class. It has static constants and abstract methods only.

The interface in java is **a mechanism to achieve fully abstraction**. There can be only abstract methods in the java interface not method body. It is used to achieve fully abstraction and multiple inheritance in Java.

Java Interface also **represents IS-A relationship**.

It cannot be instantiated just like abstract class.

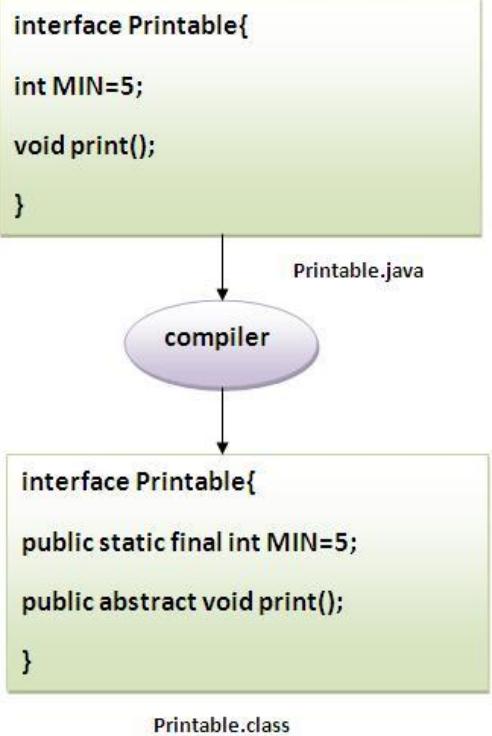
## Why use Java interface?

There are mainly three reasons to use interface. They are given below.

- It is used to achieve fully abstraction.
- By interface, we can support the functionality of multiple inheritance.
- It can be used to achieve loose coupling.

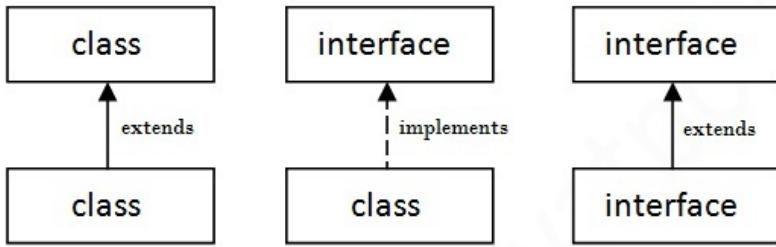
 **The java compiler adds public and abstract keywords before the interface method and public, static and final keywords before data members.**

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



## Understanding relationship between classes and interfaces

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



## Simple example of Java interface

In this example, Printable interface have only one method, its implementation is provided in the A class.

```
interface printable{
void print();
}

class A6 implements printable{
public void print(){System.out.println("Hello");}

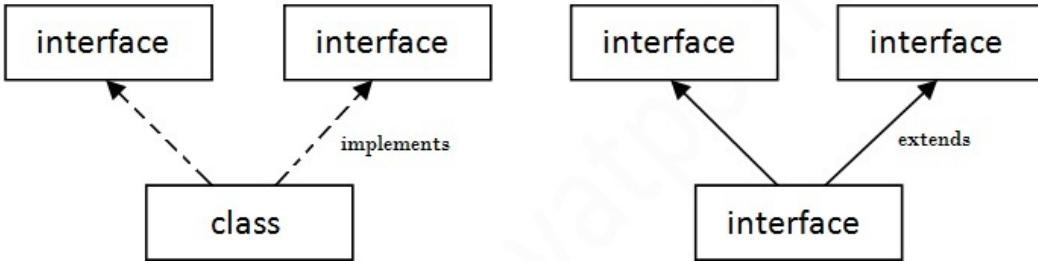
public static void main(String args[]){
A6 obj = new A6();
obj.print();
}
}
```

**Test it Now**

Output:Hello

## Multiple inheritance in Java by interface

If a class implements multiple interfaces, or an interface extends multiple interfaces i.e. known as multiple inheritance.



**Multiple Inheritance in Java**

```
interface Printable{
void print();
}

interface Showable{
void show();
}

class A7 implements Printable,Showable{
```

```

public void print(){System.out.println("Hello");}
public void show(){System.out.println("Welcome");}

public static void main(String args[]){
A7 obj = new A7();
obj.print();
obj.show();
}

```

**Test it Now**

Output:Hello  
Welcome

## Q) Multiple inheritance is not supported through class in java but it is possible by interface, why?

As we have explained in the inheritance chapter, multiple inheritance is not supported in case of class. But it is supported in case of interface because there is no ambiguity as implementation is provided by the implementation class. For example:

```

interface Printable{
void print();
}
interface Showable{
void print();
}

class TestTnterface1 implements Printable,Showable{
public void print(){System.out.println("Hello");}
public static void main(String args[]){
TestTnterface1 obj = new TestTnterface1();
obj.print();
}
}

```

**Test it Now**

Hello

As you can see in the above example, Printable and Showable interface have same methods but its implementation is provided by class TestTnterface1, so there is no ambiguity.

## Interface inheritance

A class implements interface but one interface extends another interface .

```

interface Printable{
void print();
}
interface Showable extends Printable{
void show();
}
class Testinterface2 implements Showable{

public void print(){System.out.println("Hello");}
public void show(){System.out.println("Welcome");}

public static void main(String args[]){
Testinterface2 obj = new Testinterface2();
obj.print();
obj.show();
}

```

```
}
```

### Test it Now

```
Hello  
Welcome
```

## Q) What is marker or tagged interface?

An interface that have no member is known as marker or tagged interface. For example: Serializable, Cloneable, Remote etc. They are used to provide some essential information to the JVM so that JVM may perform some useful operation.

```
//How Serializable interface is written?  
public interface Serializable{  
}
```

## Nested Interface in Java

Note: An interface can have another interface i.e. known as nested interface. We will learn it in detail in the nested classes chapter. For example:

```
interface printable{  
    void print();  
    interface MessagePrintable{  
        void msg();  
    }  
}
```

[More about Nested Interface](#)

## Difference between abstract class and interface

Abstract class and interface both are used to achieve abstraction where we can declare the abstract methods. Abstract class and interface both can't be instantiated.

But there are many differences between abstract class and interface that are given below.

Abstract class	Interface
1) Abstract class can <b>have abstract and non-abstract</b> methods.	Interface can have <b>only abstract</b> methods.
2) Abstract class <b>doesn't support multiple inheritance</b> .	Interface <b>supports multiple inheritance</b> .
3) Abstract class <b>can have final, non-final, static and non-static variables</b> .	Interface has <b>only static and final variables</b> .
4) Abstract class <b>can have static methods, main method and constructor</b> .	Interface <b>can't have static methods, main method or constructor</b> .
5) Abstract class <b>can provide the implementation of interface</b> .	Interface <b>can't provide the implementation of abstract class</b> .
6) The <b>abstract keyword</b> is used to declare abstract class.	The <b>interface keyword</b> is used to declare interface.
7) <b>Example:</b> <pre>public abstract class Shape{     public abstract void draw(); }</pre>	<b>Example:</b> <pre>public interface Drawable{     void draw(); }</pre>

Simply, abstract class achieves partial abstraction (0 to 100%) whereas interface achieves fully abstraction (100%).

## Example of abstract class and interface in Java

Let's see a simple example where we are using interface and abstract class both.

```
//Creating interface that has 4 methods
interface A{
void a();//bydefault, public and abstract
void b();
void c();
void d();
}

//Creating abstract class that provides the implementation of one method of A interface
abstract class B implements A{
public void c(){System.out.println("I am C");}
}

//Creating subclass of abstract class, now we need to provide the implementation of rest of the methods
class M extends B{
public void a(){System.out.println("I am a");}
public void b(){System.out.println("I am b");}
public void d(){System.out.println("I am d");}
}

//Creating a test class that calls the methods of A interface
class Test5{
public static void main(String args[]){
A a=new M();
a.a();
a.b();
a.c();
a.d();
}}
```

**Test it Now**

Output:

```
I am a
I am b
I am c
I am d
```

## Java Package

A **java package** is a group of similar types of classes, interfaces and sub-packages.

Package in java can be categorized in two form, built-in package and user-defined package.

There are many built-in packages such as java, lang, awt, javax, swing, net, io, util, sql etc.

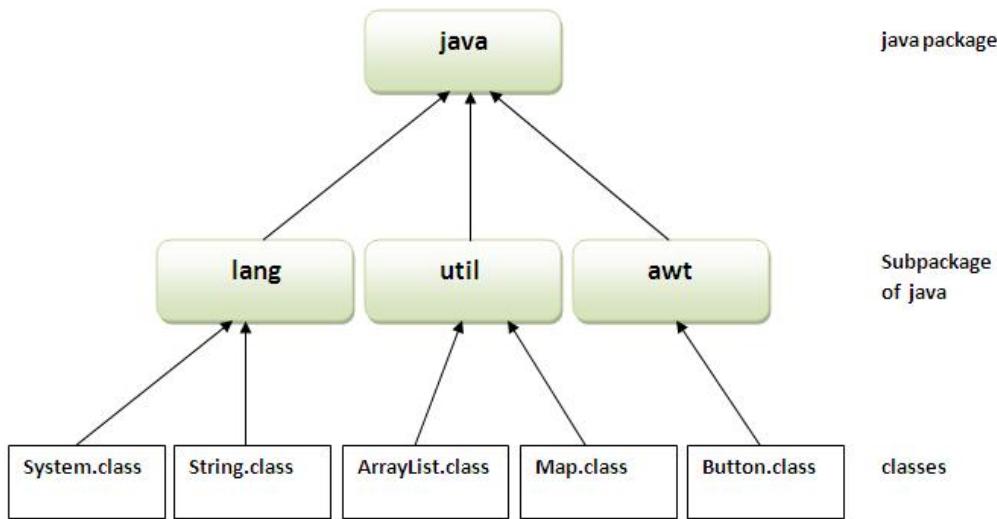
Here, we will have the detailed learning of creating and using user-defined packages.

### Advantage of Java Package

- 1) Java package is used to categorize the classes and interfaces so that they can be easily maintained.

2) Java package provides access protection.

3) Java package removes naming collision.



## Simple example of java package

The **package keyword** is used to create a package in java.

```
//save as Simple.java
package mypack;
public class Simple{
    public static void main(String args[]){
        System.out.println("Welcome to package");
    }
}
```

## How to compile java package

If you are not using any IDE, you need to follow the **syntax** given below:

```
javac -d directory javafilename
```

For **example**

```
javac -d . Simple.java
```

The -d switch specifies the destination where to put the generated class file. You can use any directory name like /home (in case of Linux), d:/abc (in case of windows) etc. If you want to keep the package within the same directory, you can use . (dot).

## How to run java package program

You need to use fully qualified name e.g. mypack.Simple etc to run the class.

**To Compile:** javac -d . Simple.java

**To Run:** java mypack.Simple

Output:Welcome to package

The -d is a switch that tells the compiler where to put the class file i.e. it represents destination. The . represents the current folder.

## How to access package from another package?

There are three ways to access the package from outside the package.

1. import package.\*;
2. import package.classname;
3. fully qualified name.

## 1) Using packagename.\*

If you use package.\* then all the classes and interfaces of this package will be accessible but not subpackages.

The import keyword is used to make the classes and interface of another package accessible to the current package.

### Example of package that import the packagename.\*

```
//save by A.java

package pack;
public class A{
    public void msg(){System.out.println("Hello");}
}

//save by B.java

package mypack;
import pack.*;

class B{
    public static void main(String args[]){
        A obj = new A();
        obj.msg();
    }
}

Output:Hello
```

---

## 2) Using packagename.classname

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

### Example of package by import package.classname

```
//save by A.java

package pack;
public class A{
    public void msg(){System.out.println("Hello");}
}

//save by B.java

package mypack;
import pack.A;

class B{
    public static void main(String args[]){
        A obj = new A();
        obj.msg();
    }
}

Output:Hello
```

---

### 3) Using fully qualified name

If you use fully qualified name then only declared class of this package will be accessible. Now there is no need to import. But you need to use fully qualified name every time when you are accessing the class or interface.

It is generally used when two packages have same class name e.g. java.util and java.sql packages contain Date class.

#### Example of package by import fully qualified name

```
//save by A.java

package pack;
public class A{
    public void msg(){System.out.println("Hello");}
}

//save by B.java

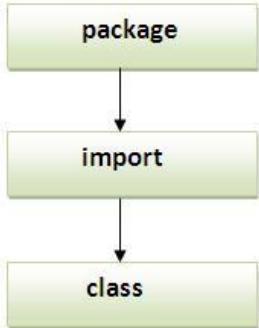
package mypack;
class B{
    public static void main(String args[]){
        pack.A obj = new pack.A(); //using fully qualified name
        obj.msg();
    }
}

Output:Hello
```

 **Note:** If you import a package, subpackages will not be imported.

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

**Note:** Sequence of the program must be package then import then class.



### Subpackage in java

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 subpackages such as lang, net, io etc. and put the Input/Output related classes in io package, Server and ServerSocket classes in net packages and so on.

 **The standard of defining package is domain.company.package e.g. com.javatpoint.bean or org.sssit.dao.**

#### Example of Subpackage

```

package com.javatpoint.core;
class Simple{
    public static void main(String args[]){
        System.out.println("Hello subpackage");
    }
}

```

**To Compile:** javac -d . Simple.java

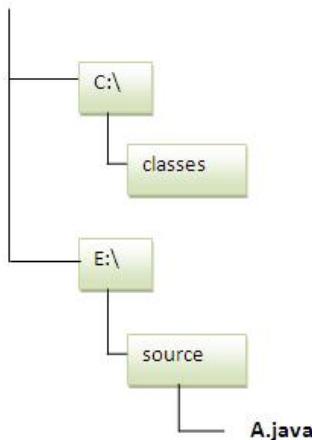
**To Run:** java com.javatpoint.core.Simple

Output:Hello subpackage

---

## How to send the class file to another directory or drive?

There is a scenario, I want to put the class file of A.java source file in classes folder of c: drive. For example:



//save as Simple.java

```

package mypack;
public class Simple{
    public static void main(String args[]){
        System.out.println("Welcome to package");
    }
}

```

**To Compile:**

```

e:\sources> javac -d c:\classes
Simple.java

```

**To Run:**

To run this program from e:\source directory, you need to set classpath of the directory where the class file resides.

```

e:\sources> set classpath=c:\classes;;
e:\sources> java mypack.Simple

```

## Another way to run this program by -classpath switch of java:

The -classpath switch can be used with javac and java tool.

To run this program from e:\source directory, you can use -classpath switch of java that tells where to look for class file. For example:

```

e:\sources> java -classpath c:\classes mypack.Simple

```

Output:Welcome to package

---

## Ways to load the class files or jar files

There are two ways to load the class files temporary and permanent.

- Temporary
  - By setting the classpath in the command prompt
  - By -classpath switch
- Permanent
  - By setting the classpath in the environment variables
  - By creating the jar file, that contains all the class files, and copying the jar file in the jre/lib/ext folder.

 **Rule:** *There can be only one public class in a java source file and it must be saved by the public class name.*

```
//save as C.java otherwise Compile Time Error

class A{}
class B{}
public class C{}
```

## How to put two public classes in a package?

If you want to put two public classes in a package, have two java source files containing one public class, but keep the package name same. For example:

```
//save as A.java

package javatpoint;
public class A{}

//save as B.java

package javatpoint;
public class B{}
```

## What is static import feature of Java5?

Click [Static Import](#) feature of Java5.

## What about package class?

Click for [Package class](#)

« prev

next »

# Access Modifiers in java

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

The access modifiers in java specifies accessibility (scope) of a data member, method, constructor or class.

There are 4 types of java access modifiers:

1. private
2. default
3. protected
4. public

There are many non-access modifiers such as static, abstract, synchronized, native, volatile, transient etc. Here, we will learn access modifiers.

---

## 1) private access modifier

The private access modifier is accessible only within class.

### Simple example of private access modifier

In this example, we have created two classes A and Simple. A class contains private data member and private method. We are accessing these private members from outside the class, so there is compile time error.

```
class A{  
    private int data=40;  
    private void msg(){System.out.println("Hello java");}  
}  
  
public class Simple{  
    public static void main(String args[]){  
        A obj=new A();  
        System.out.println(obj.data);//Compile Time Error  
        obj.msg();//Compile Time Error  
    }  
}
```

### Role of Private Constructor

If you make any class constructor private, you cannot create the instance of that class from outside the class. For example:

```
class A{  
    private A(){}//private constructor  
    void msg(){System.out.println("Hello java");}  
}  
  
public class Simple{  
    public static void main(String args[]){  
        A obj=new A();//Compile Time Error  
    }  
}
```



**Note:** A class cannot be private or protected except nested class.

## 2) default access modifier

If you don't use any modifier, it is treated as **default** by default. The default modifier is accessible only within package.

### Example of default access modifier

In this example, we have created two packages pack and mypack. We are accessing the A class from outside its package, since A class is not public, so it cannot be accessed from outside the package.

```
//save by A.java  
package pack;  
class A{  
    void msg(){System.out.println("Hello");}
```

```

}

//save by B.java
package mypack;
import pack.*;
class B{
    public static void main(String args[]){
        A obj = new A(); //Compile Time Error
        obj.msg(); //Compile Time Error
    }
}

```

In the above example, the scope of class A and its method msg() is default so it cannot be accessed from outside the package.

---

### 3) protected access modifier

The **protected access modifier** is accessible within package and outside the package but through inheritance only.

The protected access modifier can be applied on the data member, method and constructor. It can't be applied on the class.

#### Example of protected access modifier

In this example, we have created the two packages pack and mypack. The A class of pack package is public, so can be accessed from outside the package. But msg method of this package is declared as protected, so it can be accessed from outside the class only through inheritance.

```

//save by A.java
package pack;
public class A{
    protected void msg(){System.out.println("Hello");}
}

//save by B.java
package mypack;
import pack.*;

class B extends A{
    public static void main(String args[]){
        B obj = new B();
        obj.msg();
    }
}

Output:Hello

```

---

### 4) public access modifier

The **public access modifier** is accessible everywhere. It has the widest scope among all other modifiers.

#### Example of public access modifier

```

//save by A.java

package pack;
public class A{
    public void msg(){System.out.println("Hello");}
}

//save by B.java

package mypack;

```

```

import pack.*;

class B{
    public static void main(String args[]){
        A obj = new A();
        obj.msg();
    }
}

Output:Hello

```

## Understanding all java access modifiers

Let's understand the access modifiers by a simple table.

Access Modifier	within class	within package	outside package by subclass only	outside package
<b>Private</b>	Y	N	N	N
<b>Default</b>	Y	Y	N	N
<b>Protected</b>	Y	Y	Y	N
<b>Public</b>	Y	Y	Y	Y

## Java access modifiers with method overriding

If you are overriding any method, overridden method (i.e. declared in subclass) must not be more restrictive.

```

class A{
protected void msg(){System.out.println("Hello java");}
}

public class Simple extends A{
void msg(){System.out.println("Hello java");}//C.T.Error
public static void main(String args[]){
    Simple obj=new Simple();
    obj.msg();
}
}

```

The default modifier is more restrictive than protected. That is why there is compile time error.

Â« prev

next Â»

## Encapsulation in Java

**Encapsulation in java** is a *process of wrapping code and data together into a single unit*, for example capsule i.e. mixed of several medicines.

We can create a fully encapsulated class in java by making all the data members of the class private. Now we can use setter and getter methods to set and get the data in it.

The **Java Bean** class is the example of fully encapsulated class.



### Advantage of Encapsulation in java

By providing only setter or getter method, you can make the class **read-only or write-only**.

It provides you the **control over the data**. Suppose you want to set the value of id i.e. greater than 100 only, you can write the logic inside the setter method.

## Simple example of encapsulation in java

Let's see the simple example of encapsulation that has only one field with its setter and getter methods.

```
//save as Student.java
package com.javatpoint;
public class Student{
private String name;

public String getName(){
return name;
}
public void setName(String name){
this.name=name
}
}

//save as Test.java
package com.javatpoint;
class Test{
public static void main(String[] args){
Student s=new Student();
s.setname("vijay");
System.out.println(s.getName());
}
}

Compile By: javac -d . Test.java
Run By: java com.javatpoint.Test

Output: vijay
```

[Â« prev](#)

[next Â»](#)

## Object class in Java

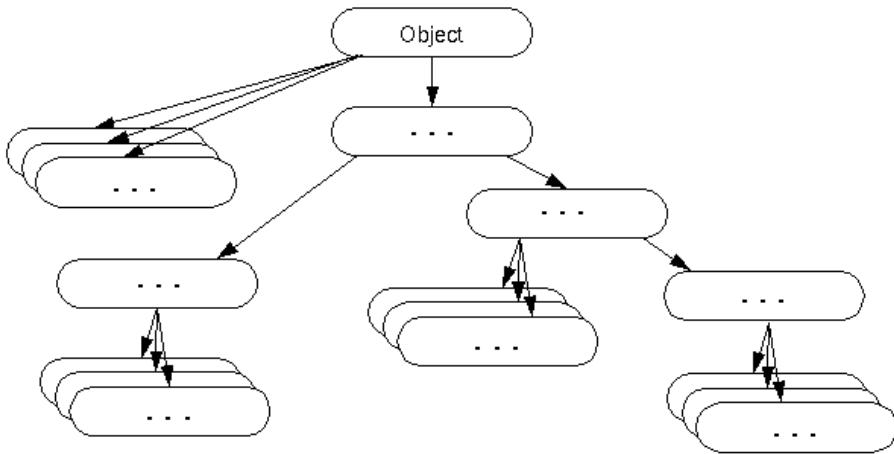
The **Object class** is the parent class of all the classes in java by default. In other words, it is the topmost class of java.

The Object class is beneficial if you want to refer any object whose type you don't know. Notice that parent class reference variable can refer the child class object, known as upcasting.

Let's take an example, there is getObject() method that returns an object but it can be of any type like Employee, Student etc, we can use Object class reference to refer that object. For example:

```
Object obj=getObject(); //we don't know what object would be returned from this method
```

The Object class provides some common behaviours to all the objects such as object can be compared, object can be cloned, object can be notified etc.



## Methods of Object class

The Object class provides many methods. They are as follows:

Method	Description
<b>public final Class getClass()</b>	returns the Class class object of this object. The Class class can further be used to get the metadata of this class.
<b>public int hashCode()</b>	returns the hashcode number for this object.
<b>public boolean equals(Object obj)</b>	compares the given object to this object.
<b>protected Object clone() throws CloneNotSupportedException</b>	creates and returns the exact copy (clone) of this object.
<b>public String toString()</b>	returns the string representation of this object.
<b>public final void notify()</b>	wakes up single thread, waiting on this object's monitor.
<b>public final void notifyAll()</b>	wakes up all the threads, waiting on this object's monitor.
<b>public final void wait(long timeout) throws InterruptedException</b>	causes the current thread to wait for the specified milliseconds, until another thread notifies (invokes notify() or notifyAll() method).
<b>public final void wait(long timeout, int nanos) throws InterruptedException</b>	causes the current thread to wait for the specified miliseconds and nanoseconds, until another thread notifies (invokes notify() or notifyAll() method).
<b>public final void wait() throws InterruptedException</b>	causes the current thread to wait, until another thread notifies (invokes notify() or notifyAll() method).
<b>protected void finalize() throws Throwable</b>	is invoked by the garbage collector before object is being garbage collected.

We will have the detailed learning of these methods in next chapters.

## Object Cloning in Java

The **object cloning** is a way to create exact copy of an object. For this purpose, clone() method of Object class is used to clone an object.

The **java.lang.Cloneable interface** must be implemented by the class whose object clone we want to create. If we don't implement Cloneable interface, clone() method generates **CloneNotSupportedException**.

The **clone() method** is defined in the Object class. Syntax of the clone() method is as follows:

```
protected Object clone() throws CloneNotSupportedException
```



## Why use clone() method ?

The **clone() method** saves the extra processing task for creating the exact copy of an object. If we perform it by using the new keyword, it will take a lot of processing to be performed that is why we use object cloning.

## Advantage of Object cloning

Less processing task.

## Example of clone() method (Object cloning)

Let's see the simple example of object cloning

```
class Student18 implements Cloneable{
int rollno;
String name;

Student18(int rollno,String name){
this.rollno=rollno;
this.name=name;
}

public Object clone()throws CloneNotSupportedException{
return super.clone();
}

public static void main(String args[]){
try{
Student18 s1=new Student18(101,"amit");

Student18 s2=(Student18)s1.clone();

System.out.println(s1.rollno+" "+s1.name);
System.out.println(s2.rollno+" "+s2.name);

}catch(CloneNotSupportedException c){}
}
}
```

**Test it Now**

```
Output:101 amit
      101 amit
```

**download the example of object cloning**

As you can see in the above example, both reference variables have the same value. Thus, the clone() copies the values of an object to another. So we don't need to write explicit code to copy the value of an object to another.

If we create another object by new keyword and assign the values of another object to this one, it will require a lot of processing on this object. So to save the extra processing task we use clone() method.

[Â« prev](#)

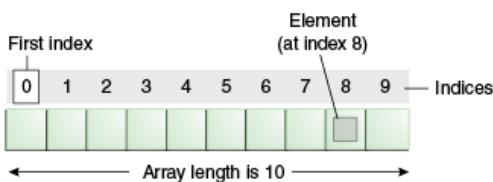
[next Â»](#)

## Java Array

Normally, array is a collection of similar type of elements that have contiguous memory location.

**Java array** is an object that contains elements of similar data type. It is a data structure where we store similar elements. We can store only fixed set of elements in a java array.

Array in java is index based, first element of the array is stored at 0 index.



### Advantage of Java Array

- **Code Optimization:** It makes the code optimized, we can retrieve or sort the data easily.
- **Random access:** We can get any data located at any index position.

---

### Disadvantage of Java Array

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

---

## Types of Array in java

There are two types of array.

- Single Dimensional Array
- Multidimensional Array

---

## Single Dimensional Array in java

### Syntax to Declare an Array in java

```
dataType[] arr; (or)  
dataType []arr; (or)  
dataType arr[];
```

### Instantiation of an Array in java

```
arrayRefVar=new datatype[size];
```

### Example of single dimensional java array

Let's see the simple example of java array, where we are going to declare, instantiate, initialize and traverse an array.

```
class Testarray{  
public static void main(String args[]){  
  
int a[]=new int[5];//declaration and instantiation  
a[0]=10;//initialization  
a[1]=20;
```

```
a[2]=70;  
a[3]=40;  
a[4]=50;  
  
//printing array  
for(int i=0;i<a.length;i++)//length is the property of array  
System.out.println(a[i]);  
  
}}
```

**Test it Now**

```
Output: 10  
20  
70  
40  
50
```

---

## Declaration, Instantiation and Initialization of Java Array

We can declare, instantiate and initialize the java array together by:

```
int a[]={33,3,4,5}//declaration, instantiation and initialization
```

Let's see the simple example to print this array.

```
class Testarray1{  
public static void main(String args[]){  
  
int a[]={33,3,4,5}//declaration, instantiation and initialization  
  
//printing array  
for(int i=0;i<a.length;i++)//length is the property of array  
System.out.println(a[i]);  
  
}}
```

**Test it Now**

```
Output:33  
3  
4  
5
```

---

## Passing Array to method in java

We can pass the java array to method so that we can reuse the same logic on any array.

Let's see the simple example to get minimum number of an array using method.

```
class Testarray2{  
static void min(int arr[]){  
int min=arr[0];  
for(int i=1;i<arr.length;i++)  
if(min>arr[i])  
min=arr[i];  
  
System.out.println(min);  
}
```

```
public static void main(String args[]){
    int a[]={33,3,4,5};
    min(a);//passing array to method
}
```

**Test it Now**

Output:3

## Multidimensional array in java

In such case, data is stored in row and column based index (also known as matrix form).

### Syntax to Declare Multidimensional Array in java

```
dataType[][] arrayRefVar; (or)
dataType [][]arrayRefVar; (or)
dataType arrayRefVar[][]; (or)
dataType []arrayRefVar[];
```

### Example to instantiate Multidimensional Array in java

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

### Example to initialize Multidimensional Array in java

```
arr[0][0]=1;
arr[0][1]=2;
arr[0][2]=3;
arr[1][0]=4;
arr[1][1]=5;
arr[1][2]=6;
arr[2][0]=7;
arr[2][1]=8;
arr[2][2]=9;
```

## Example of Multidimensional java array

Let's see the simple example to declare, instantiate, initialize and print the 2Dimensional array.

```
class Testarray3{
    public static void main(String args[]){
        //declaring and initializing 2D array
        int arr[][]={{1,2,3},{2,4,5},{4,4,5}};

        //printing 2D array
        for(int i=0;i<3;i++){
            for(int j=0;j<3;j++){
                System.out.print(arr[i][j]+" ");
            }
            System.out.println();
        }
    }
}
```

**Test it Now**

Output:1 2 3  
2 4 5

## What is the class name of java array?

In java, array is an object. For array object, an proxy class is created whose name can be obtained by `getClass().getName()` method on the object.

```
class Testarray4{
public static void main(String args[]){

int arr[]={4,4,5};

Class c=arr.getClass();
String name=c.getName();

System.out.println(name);

}}
```

**Test it Now**

Output:I

## Copying a java array

We can copy an array to another by the `arraycopy` method of `System` class.

### Syntax of `arraycopy` method

```
public static void arraycopy(
Object src, int srcPos, Object dest, int destPos, int length
)
```

### Example of `arraycopy` method

```
class TestArrayCopyDemo {
    public static void main(String[] args) {
        char[] copyFrom = { 'd', 'e', 'c', 'a', 'f', 'f', 'e',
            'i', 'n', 'a', 't', 'e', 'd' };
        char[] copyTo = new char[7];

        System.arraycopy(copyFrom, 2, copyTo, 0, 7);
        System.out.println(new String(copyTo));
    }
}
```

**Test it Now**

Output:caffein

## Addition of 2 matrices in java

Let's see a simple example that adds two matrices.

```
class Testarray5{
public static void main(String args[]){
//creating two matrices
int a[][]={{1,3,4},{3,4,5}};
int b[][]={{1,3,4},{3,4,5}};

//creating another matrix to store the sum of two matrices
```

```

int c[][]=new int[2][3];

//adding and printing addition of 2 matrices
for(int i=0;i<2;i++){
for(int j=0;j<3;j++){
c[i][j]=a[i][j]+b[i][j];
System.out.print(c[i][j]+" ");
}
System.out.println();//new line
}

}}
```

**Test it Now**

Output:2 6 8  
6 8 10

## Wrapper class in Java

**Wrapper class in java** provides the mechanism to convert primitive into object and object into primitive .

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

One of the eight classes of *java.lang* package are known as wrapper class in java. The list of eight wrapper classes are given below:

Primitive Type	Wrapper class
boolean	Boolean
char	Character
byte	Byte
short	Short
int	Integer
long	Long
float	Float
double	Double

## Wrapper class Example: Primitive to Wrapper

```

public class WrapperExample1{
public static void main(String args[]){
//Converting int into Integer
int a=20;
Integer i=Integer.valueOf(a);//converting int into Integer
Integer j=a;//autoboxing, now compiler will write Integer.valueOf(a) internally

System.out.println(a+" "+i+" "+j);
}}
```

Output:

20 20 20

## Wrapper class Example: Wrapper to Primitive

```

public class WrapperExample2{
public static void main(String args[]){
//Converting Integer to int
```

```

Integer a=new Integer(3);
int i=a.intValue(); //converting Integer to int
int j=a; //unboxing, now compiler will write a.intValue() internally

System.out.println(a+" "+i+" "+j);
}

```

Output:

3 3 3

## Call by Value and Call by Reference in Java

There is only call by value in java, not call by reference. If we call a method passing a value, it is known as call by value. The changes being done in the called method, is not affected in the calling method.

### Example of call by value in java

In case of call by value original value is not changed. Let's take a simple example:

```

class Operation{
    int data=50;

    void change(int data){
        data=data+100; //changes will be in the local variable only
    }

    public static void main(String args[]){
        Operation op=new Operation();

        System.out.println("before change "+op.data);
        op.change(500);
        System.out.println("after change "+op.data);

    }
}

```

[download this example](#)

Output:  
before change 50  
after change 50

---

### Another Example of call by value in java

In case of call by reference original value is changed if we made changes in the called method. If we pass object in place of any primitive value, original value will be changed. In this example we are passing object as a value. Let's take a simple example:

```

class Operation2{
    int data=50;

    void change(Operation2 op){
        op.data=op.data+100; //changes will be in the instance variable
    }

    public static void main(String args[]){

```

```

Operation2 op=new Operation2();

System.out.println("before change "+op.data);
op.change(op);//passing object
System.out.println("after change "+op.data);

}
}

```

[download this example](#)

```
Output:before change 50
           after change 150
```

[\*\*<<prev\*\*](#)

[\*\*next>>\*\*](#)

## Java Strictfp Keyword

Java strictfp keyword ensures that you will get the same result on every platform if you perform operations in the floating-point variable. The precision may differ from platform to platform that is why java programming language have provided the strictfp keyword, so that you get same result on every platform. So, now you have better control over the floating-point arithmetic.

### Legal code for strictfp keyword

The strictfp keyword can be applied on methods, classes and interfaces.

```

strictfp class A{}//strictfp applied on class

strictfp interface M{}//strictfp applied on interface

class A{
strictfp void m(){}//strictfp applied on method
}
```

### Illegal code for strictfp keyword

The strictfp keyword **cannot** be applied on abstract methods, variables or constructors.

```

class B{
strictfp abstract void m();//Illegal combination of modifiers
}

class B{
strictfp int data=10;//modifier strictfp not allowed here
}

class B{
strictfp B(){}//modifier strictfp not allowed here
}
```

[\*\*â†’ prev\*\*](#)

[\*\*next â†’\*\*](#)

## Creating API Document I javadoc tool

We can create document api in java by the help of **javadoc** tool. In the java file, we must use the documentation comment `/**... */` to post information for the class, method, constructor, fields etc.

Let's see the simple class that contains documentation comment.

```

package com.abc;
/** This class is a user-defined class that contains one methods cube.*/
public class M{

/** The cube method prints cube of the given number */
public static void cube(int n){System.out.println(n*n*n);}
}

```

To create the document API, you need to use the javadoc tool followed by java file name. There is no need to compile the javafie.

On the command prompt, you need to write:

```
javadoc M.java
```

to generate the document api. Now, there will be created a lot of html files. Open the index.html file to get the information about the classes.

## Java 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.

### Simple example of command-line argument in java

In this example, we are receiving only one argument and printing it. To run this java program, you must pass at least one argument from the command prompt.

```

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

compile by > javac CommandLineExample.java
run by > java CommandLineExample sonoo

Output: Your first argument is: sonoo

```

### Example of command-line argument that prints all the values

In this example, we are printing all the arguments passed from the command-line. For this purpose, we have traversed the array using for loop.

```

class A{
public static void main(String args){

for(int i=0;i<args.length;i++)
System.out.println(args[i]);

}
}

compile by > javac A.java

```

```
run by > java A sonoo jaiswal 1 3 abc
```

```
Output: sonoo
        jaiswal
        1
        3
        abc
```

## Difference between object and class

There are many differences between object and class. A list of differences between object and class are given below:

No. Object	Class
1) Object is an <b>instance</b> of a class.	Class is a <b>blueprint or template</b> from which objects are created.
2) Object is a <b>real world entity</b> such as pen, laptop, mobile, bed, keyboard, mouse, chair etc.	Class is a <b>group of similar objects</b> .
3) Object is a <b>physical</b> entity.	Class is a <b>logical</b> entity.
4) Object is created through <b>new keyword</b> mainly e.g. Student s1=new Student();	Class is declared using <b>class keyword</b> e.g. class Student{}
5) Object is created <b>many times</b> as per requirement.	Class is declared <b>once</b> .
6) Object <b>allocates memory when it is created</b>	Class <b>doesn't allocated memory when it is created</b> .
7) There are <b>many ways to create object</b> in java such as new keyword, newInstance() method, clone() method, factory method and deserialization.	There is only <b>one way to define class</b> in java using class keyword.

## Difference between method overloading and method overriding in java

There are many differences between method overloading and method overriding in java. A list of differences between method overloading and method overriding are given below:

No. Method Overloading	Method Overriding
1) Method overloading is used <i>to increase the readability</i> of the program.	Method overriding is used <i>to provide the specific implementation</i> of the method that is already provided by its super class.
2) Method overloading is performed <i>within class</i> .	Method overriding occurs <i>in two classes</i> that have IS-A (inheritance) relationship.
3) In case of method overloading, <i>parameter must be different</i> .	In case of method overriding, <i>parameter must be same</i> .
4) Method overloading is the example of <i>compile time polymorphism</i> .	Method overriding is the example of <i>run time polymorphism</i> .
5) In java, method overloading can't be performed by changing return type of the method only. <i>Return type can be same or different</i> in method overloading. But you must have to change the parameter.	<i>Return type must be same or covariant</i> in method overriding.

## Java Method Overloading example

```
class OverloadingExample{  
    static int add(int a,int b){return a+b;}  
    static int add(int a,int b,int c){return a+b+c;}  
}
```

## Java Method Overriding example

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

## Java String

**Java String** provides a lot of concepts that can be performed on a string such as compare, concat, equals, split, length, replace, compareTo, intern, substring etc.

In java, string is basically an object that represents sequence of char values.

An array of characters works same as java string. For example:

```
char[] ch={'j','a','v','a','t','p','o','i','n','t'};  
String s=new String(ch);
```

is same as:

```
String s="javatpoint";
```

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

The java String is immutable i.e. it cannot be changed but a new instance is created. For mutable class, you can use `StringBuffer` and `StringBuilder` class.

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. 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

---

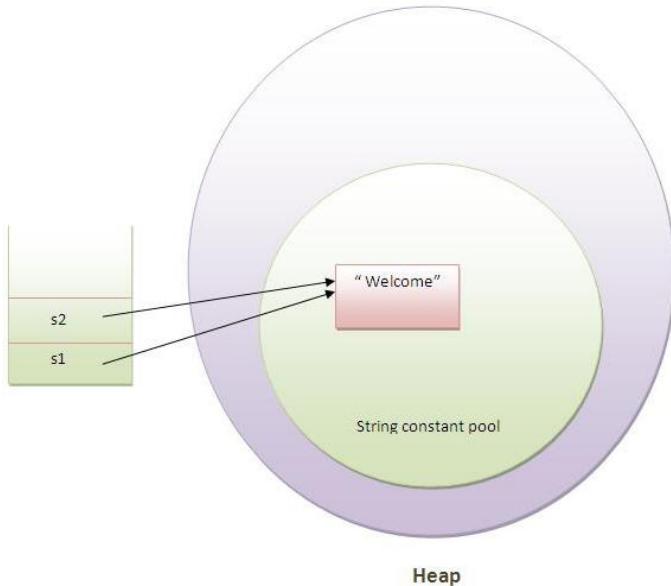
### 1) String Literal

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

```
String s="welcome";
```

Each time you create a string literal, the JVM checks the string constant pool 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:

```
String s1="Welcome";
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

```
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

```
public class StringExample{
public static void main(String args[]){
String s1="java";//creating string by java string literal

char ch[]={ 's','t','r','i','n','g','s'};
String s2=new String(ch);//converting char array to string

String s3=new String("example");//creating java string by new keyword

System.out.println(s1);
System.out.println(s2);
System.out.println(s3);
}}
```

**Test it Now**

```
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	<code>char charAt(int index)</code>	returns char value for the particular index
2	<code>int length()</code>	returns string length
3	<code>static String format(String format, Object... args)</code>	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>String trim()</code>	returns trimmed string omitting leading and trailing spaces
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	<code>int indexOf(String substring)</code>	returns specified substring index
22	<code>int indexOf(String substring, int fromIndex)</code>	returns specified substring index starting with given index
23	<code>String toLowerCase()</code>	returns string in lowercase.
24	<code>String toLowerCase(Locale l)</code>	returns string in lowercase using specified locale.
25	<code>String toUpperCase()</code>	returns string in uppercase.
26	<code>String toUpperCase(Locale l)</code>	returns string in uppercase using specified locale.

# 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.

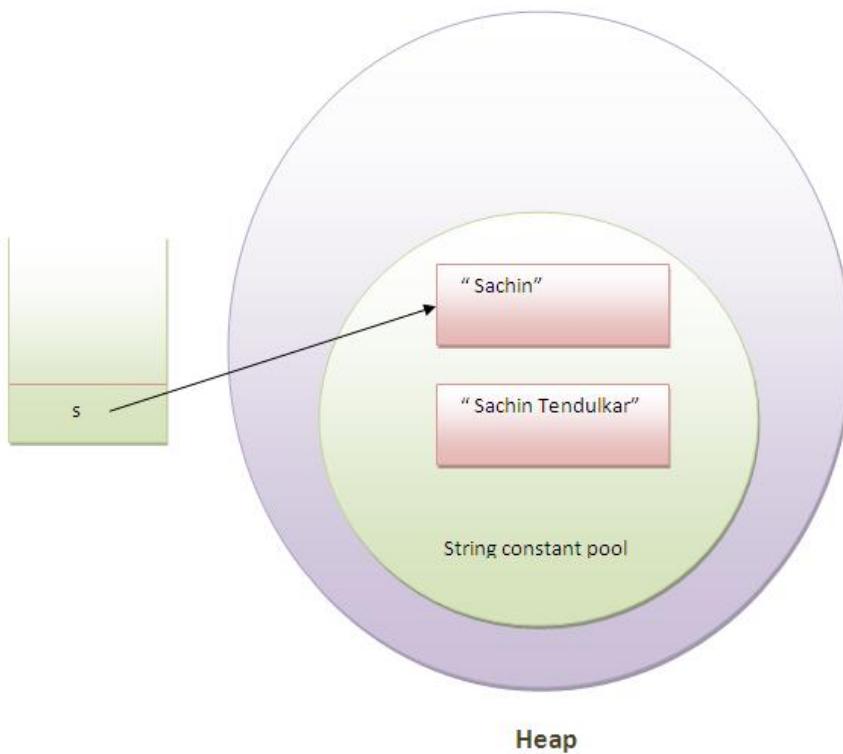
Let's try to understand the immutability concept by the example given below:

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

**Test it Now**

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:

```
class Testimmutablestring1{  
    public static void main(String args[]){  
        String s="Sachin";  
        s=s.concat(" Tendulkar");  
        System.out.println(s);  
    }  
}
```

**Test it Now**

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.

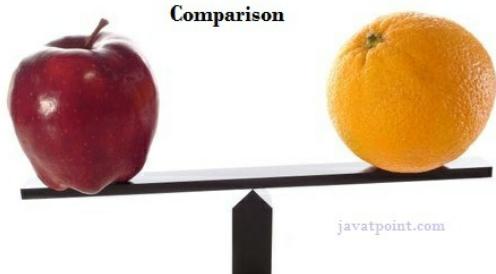
## Java String compare

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:

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

```
class Teststringcomparison1{
    public static void main(String args[]){
        String s1="Sachin";
        String s2="Sachin";
        String s3=new String("Sachin");
        String s4="Saurav";
        System.out.println(s1.equals(s2));//true
        System.out.println(s1.equals(s3));//true
        System.out.println(s1.equals(s4));//false
    }
}
```

**Test it Now**

```
Output:true
      true
      false

class Teststringcomparison2{
    public static void main(String args[]){
        String s1="Sachin";
        String s2="SACHIN";

        System.out.println(s1.equals(s2));//false
        System.out.println(s1.equalsIgnoreCase(s3));//true
    }
}
```

**Test it Now**

```
Output:false
      true
```

[Click me for more about equals\(\) method](#)

## 2) String compare by == operator

The == operator compares references not values.

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

**Test it Now**

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:

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

```
class Teststringcomparison4{
    public static void main(String args[]){
        String s1="Sachin";
        String s2="Sachin";
        String s3="Ratan";
        System.out.println(s1.compareTo(s2));//0
        System.out.println(s1.compareTo(s3));//1(because s1>s3)
        System.out.println(s3.compareTo(s1));//-1(because s3 < s1 )
    }
}
```

**Test it Now**

Output:  
0  
1  
-1

[Click me for more about compareTo\(\) method](#)

# 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:

```
class TestStringConcatenation1{
```

```

public static void main(String args[]){
    String s="Sachin"+ " Tendulkar";
    System.out.println(s);//Sachin Tendulkar
}

```

**Test it Now**

Output:Sachin Tendulkar

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

```
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:

```

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

```

**Test it Now**

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:

```
public String concat(String another)
```

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

```

class TestStringConcatenation3{
    public static void main(String args[]){
        String s1="Sachin ";
        String s2="Tendulkar";
        String s3=s1.concat(s2);
        System.out.println(s3);//Sachin Tendulkar
    }
}

```

**Test it Now**

Sachin Tendulkar

## Substring in Java

A part of string is called **substring**. In other words, substring is a subset of another string. In case of substring `startIndex` is inclusive and `endIndex` is exclusive.



**Note:** Index starts from 0.

You can get substring from the given string object by one of the two methods:

1. **public String substring(int startIndex):** This method returns new `String` object containing the substring of the given string from specified `startIndex` (inclusive).

2. **public String substring(int startIndex, int endIndex):** This method returns new String object containing the substring of the given string from specified startIndex to endIndex.

In case of string:

- **startIndex:** inclusive
- **endIndex:** exclusive

Let's understand the startIndex and endIndex by the code given below.

```
String s="hello";
System.out.println(s.substring(0,2));//he
```

In the above substring, 0 points to h but 2 points to e (because end index is exclusive).

## Example of java substring

```
public class TestSubstring{
    public static void main(String args[]){
        String s="Sachin Tendulkar";
        System.out.println(s.substring(6));//Tendulkar
        System.out.println(s.substring(0,6));//Sachin
    }
}
```

**Test it Now**

```
Tendulkar
Sachin
```

## Java String class methods

The `java.lang.String` class provides a lot of methods to work on string. By the help of these methods, we can perform operations on string such as trimming, concatenating, converting, comparing, replacing strings etc.

Java String is a powerful concept because everything is treated as a string if you submit any form in window based, web based or mobile application.

Let's see the important methods of String class.

### Java String `toUpperCase()` and `toLowerCase()` method

The java string `toUpperCase()` method converts this string into uppercase letter and string `toLowerCase()` method into lowercase letter.

```
String s="Sachin";
System.out.println(s.toUpperCase());//SACHIN
System.out.println(s.toLowerCase());//sachin
System.out.println(s);//Sachin(no change in original)
```

**Test it Now**

```
SACHIN
sachin
Sachin
```

### Java String `trim()` method

The string `trim()` method eliminates white spaces before and after string.

```
String s=" Sachin ";
System.out.println(s);// Sachin
System.out.println(s.trim());//Sachin
```

**Test it Now**

```
Sachin  
Sachin
```

## Java String startsWith() and endsWith() method

```
String s="Sachin";  
System.out.println(s.startsWith("Sa")); //true  
System.out.println(s.endsWith("n")); //true
```

**Test it Now**

```
true  
true
```

## Java String charAt() method

The string charAt() method returns a character at specified index.

```
String s="Sachin";  
System.out.println(s.charAt(0)); //S  
System.out.println(s.charAt(3)); //h
```

**Test it Now**

```
s  
h
```

## Java String length() method

The string length() method returns length of the string.

```
String s="Sachin";  
System.out.println(s.length()); //6
```

**Test it Now**

```
6
```

## Java String intern() method

A pool of strings, initially empty, is maintained privately by the class String.

When the intern method is invoked, if the pool already contains a string equal to this String object as determined by the equals(Object) method, then the string from the pool is returned. Otherwise, this String object is added to the pool and a reference to this String object is returned.

```
String s=new String("Sachin");  
String s2=s.intern();  
System.out.println(s2); //Sachin
```

**Test it Now**

```
Sachin
```

## Java String valueOf() method

The string valueOf() method converts given type such as int, long, float, double, boolean, char and char array into string.

```
int a=10;  
String s=String.valueOf(a);  
System.out.println(s+10);
```

Output:

```
1010
```

## Java String replace() method

The string replace() method replaces all occurrence of first sequence of character with second sequence of character.

```
String s1="Java is a programming language. Java is a platform. Java is an Island.";
String replaceString=s1.replace("Java","Kava");//replaces all occurrences of "Java" to "Kava"
System.out.println(replaceString);
```

Output:

```
Kava is a programming language. Kava is a platform. Kava is an Island.
```

## Java StringBuffer class

Java StringBuffer class is used to created mutable (modifiable) string. The StringBuffer class in java is same as String class except it is mutable i.e. it can be changed.



**Note:** Java StringBuffer class is thread-safe i.e. multiple threads cannot access it simultaneously. So it is safe and will result in an order.

## Important Constructors of StringBuffer class

1. **StringBuffer():** creates an empty string buffer with the initial capacity of 16.
2. **StringBuffer(String str):** creates a string buffer with the specified string.
3. **StringBuffer(int capacity):** creates an empty string buffer with the specified capacity as length.

## Important methods of StringBuffer class

1. **public synchronized StringBuffer append(String s):** is used to append the specified string with this string. The append() method is overloaded like append(char), append(boolean), append(int), append(float), append(double) etc.
2. **public synchronized StringBuffer insert(int offset, String s):** is used to insert the specified string with this string at the specified position. The insert() method is overloaded like insert(int, char), insert(int, boolean), insert(int, int), insert(int, float), insert(int, double) etc.
3. **public synchronized StringBuffer replace(int startIndex, int endIndex, String str):** is used to replace the string from specified startIndex and endIndex.
4. **public synchronized StringBuffer delete(int startIndex, int endIndex):** is used to delete the string from specified startIndex and endIndex.
5. **public synchronized StringBuffer reverse():** is used to reverse the string.
6. **public int capacity():** is used to return the current capacity.
7. **public void ensureCapacity(int minimumCapacity):** is used to ensure the capacity at least equal to the given minimum.
8. **public char charAt(int index):** is used to return the character at the specified position.
9. **public int length():** is used to return the length of the string i.e. total number of characters.
10. **public String substring(int beginIndex):** is used to return the substring from the specified beginIndex.
11. **public String substring(int beginIndex, int endIndex):** is used to return the substring from the specified beginIndex and endIndex.

## What is mutable string

A string that can be modified or changed is known as mutable string. StringBuffer and StringBuilder classes are used for creating mutable string.

### 1) StringBuffer append() method

The append() method concatenates the given argument with this string.

```
class A{
```

```
public static void main(String args[]){
StringBuffer sb=new StringBuffer("Hello ");
sb.append("Java");//now original string is changed
System.out.println(sb);//prints Hello Java
}
}
```

## 2) StringBuffer insert() method

The insert() method inserts the given string with this string at the given position.

```
class A{
public static void main(String args[]){
StringBuffer sb=new StringBuffer("Hello ");
sb.insert(1,"Java");//now original string is changed
System.out.println(sb);//prints HJavaello
}
}
```

## 3) StringBuffer replace() method

The replace() method replaces the given string from the specified beginIndex and endIndex.

```
class A{
public static void main(String args[]){
StringBuffer sb=new StringBuffer("Hello");
sb.replace(1,3,"Java");
System.out.println(sb);//prints HJava
}
}
```

## 4) StringBuffer delete() method

The delete() method of StringBuffer class deletes the string from the specified beginIndex to endIndex.

```
class A{
public static void main(String args[]){
StringBuffer sb=new StringBuffer("Hello");
sb.delete(1,3);
System.out.println(sb);//prints Hlo
}
}
```

## 5) StringBuffer reverse() method

The reverse() method of StringBuilder class reverses the current string.

```
class A{
public static void main(String args[]){
StringBuffer sb=new StringBuffer("Hello");
sb.reverse();
System.out.println(sb);//prints olleH
}
}
```

## 6) StringBuffer capacity() method

The capacity() method of StringBuffer class returns the current capacity of the buffer. The default capacity of the buffer is 16. If the number of character increases from its current capacity, it increases the capacity by  $(\text{oldcapacity} * 2) + 2$ . For example if your current capacity is 16, it will be  $(16 * 2) + 2 = 34$ .

```
class A{
```

```

public static void main(String args[]){
    StringBuffer sb=new StringBuffer();
    System.out.println(sb.capacity());//default 16
    sb.append("Hello");
    System.out.println(sb.capacity());//now 16
    sb.append("java is my favourite language");
    System.out.println(sb.capacity());//now (16*2)+2=34 i.e (oldcapacity*2)+2
}
}

```

## 7) StringBuffer ensureCapacity() method

The ensureCapacity() method of StringBuffer class ensures that the given capacity is the minimum to the current capacity. If it is greater than the current capacity, it increases the capacity by (oldcapacity\*2)+2. For example if your current capacity is 16, it will be (16\*2)+2=34.

```

class A{
    public static void main(String args[]){
        StringBuffer sb=new StringBuffer();
        System.out.println(sb.capacity());//default 16
        sb.append("Hello");
        System.out.println(sb.capacity());//now 16
        sb.append("java is my favourite language");
        System.out.println(sb.capacity());//now (16*2)+2=34 i.e (oldcapacity*2)+2
        sb.ensureCapacity(10);//now no change
        System.out.println(sb.capacity());//now 34
        sb.ensureCapacity(50);//now (34*2)+2
        System.out.println(sb.capacity());//now 70
    }
}

```

# Java StringBuilder class

Java StringBuilder class is used to create mutable (modifiable) string. The Java StringBuilder class is same as StringBuffer class except that it is non-synchronized. It is available since JDK 1.5.

## Important Constructors of StringBuilder class

1. **StringBuilder():** creates an empty string Builder with the initial capacity of 16.
2. **StringBuilder(String str):** creates a string Builder with the specified string.
3. **StringBuilder(int length):** creates an empty string Builder with the specified capacity as length.

## Important methods of StringBuilder class

Method	Description
public StringBuilder append(String s)	is used to append the specified string with this string. The append() method is overloaded like append(char), append(boolean), append(int), append(float), append(double) etc.
public StringBuilder insert(int offset, String s)	is used to insert the specified string with this string at the specified position. The insert() method is overloaded like insert(int, char), insert(int, boolean), insert(int, int), insert(int, float), insert(int, double) etc.
public StringBuilder replace(int startIndex, int endIndex, String str)	is used to replace the string from specified startIndex and endIndex.
public StringBuilder delete(int startIndex, int endIndex)	is used to delete the string from specified startIndex and endIndex.

public String reverse()	is used to reverse the string.
public int capacity()	is used to return the current capacity.
public void ensureCapacity(int minimumCapacity)	is used to ensure the capacity at least equal to the given minimum.
public char charAt(int index)	is used to return the character at the specified position.
public int length()	is used to return the length of the string i.e. total number of characters.
public String substring(int beginIndex)	is used to return the substring from the specified beginIndex.
public String substring(int beginIndex, int endIndex)	is used to return the substring from the specified beginIndex and endIndex.

## Java StringBuilder Examples

Let's see the examples of different methods of StringBuilder class.

### 1) StringBuilder append() method

The StringBuilder append() method concatenates the given argument with this string.

```
class A{
    public static void main(String args[]){
        StringBuilder sb=new StringBuilder("Hello ");
        sb.append("Java");//now original string is changed
        System.out.println(sb);//prints Hello Java
    }
}
```

### 2) StringBuilder insert() method

The StringBuilder insert() method inserts the given string with this string at the given position.

```
class A{
    public static void main(String args[]){
        StringBuilder sb=new StringBuilder("Hello ");
        sb.insert(1,"Java");//now original string is changed
        System.out.println(sb);//prints HJavaello
    }
}
```

### 3) StringBuilder replace() method

The StringBuilder replace() method replaces the given string from the specified beginIndex and endIndex.

```
class A{
    public static void main(String args[]){
        StringBuilder sb=new StringBuilder("Hello");
        sb.replace(1,3,"Java");
        System.out.println(sb);//prints HJava
    }
}
```

### 4) StringBuilder delete() method

The delete() method of StringBuilder class deletes the string from the specified beginIndex to endIndex.

```
class A{
    public static void main(String args[]){
        StringBuilder sb=new StringBuilder("Hello");
        sb.delete(1,3);
    }
}
```

```
System.out.println(sb); //prints Hlo
}
}
```

## 5) StringBuilder reverse() method

The reverse() method of StringBuilder class reverses the current string.

```
class A{
public static void main(String args[]){
StringBuilder sb=new StringBuilder("Hello");
sb.reverse();
System.out.println(sb); //prints olleH
}
}
```

## 6) StringBuilder capacity() method

The capacity() method of StringBuilder class returns the current capacity of the Builder. The default capacity of the Builder is 16. If the number of character increases from its current capacity, it increases the capacity by  $(\text{oldcapacity} \times 2) + 2$ . For example if your current capacity is 16, it will be  $(16 \times 2) + 2 = 34$ .

```
class A{
public static void main(String args[]){
StringBuilder sb=new StringBuilder();
System.out.println(sb.capacity()); //default 16
sb.append("Hello");
System.out.println(sb.capacity()); //now 16
sb.append("java is my favourite language");
System.out.println(sb.capacity()); //now  $(16 \times 2) + 2 = 34$  i.e.  $(\text{oldcapacity} \times 2) + 2$ 
}
}
```

## 7) StringBuilder ensureCapacity() method

The ensureCapacity() method of StringBuilder class ensures that the given capacity is the minimum to the current capacity. If it is greater than the current capacity, it increases the capacity by  $(\text{oldcapacity} \times 2) + 2$ . For example if your current capacity is 16, it will be  $(16 \times 2) + 2 = 34$ .

```
class A{
public static void main(String args[]){
StringBuilder sb=new StringBuilder();
System.out.println(sb.capacity()); //default 16
sb.append("Hello");
System.out.println(sb.capacity()); //now 16
sb.append("java is my favourite language");
System.out.println(sb.capacity()); //now  $(16 \times 2) + 2 = 34$  i.e.  $(\text{oldcapacity} \times 2) + 2$ 
sb.ensureCapacity(10); //now no change
System.out.println(sb.capacity()); //now 34
sb.ensureCapacity(50); //now  $(34 \times 2) + 2$ 
System.out.println(sb.capacity()); //now 70
}
}
```

# Difference between String and StringBuffer

There are many differences between String and StringBuffer. A list of differences between String and StringBuffer are given below:

No.	String	StringBuffer
1)	String class is immutable.	StringBuffer class is mutable.
2)	String is slow and consumes more memory when you concat too many strings because every time it creates new instance.	StringBuffer is fast and consumes less memory when you concat strings.
3)	String class overrides the equals() method of Object class. So you can compare the contents of two strings by equals() method.	StringBuffer class doesn't override the equals() method of Object class.

## Performance Test of String and StringBuffer

```

public class ConcatTest{
    public static String concatWithString(){
        String t = "Java";
        for (int i=0; i<10000; i++){
            t = t + "Tpoint";
        }
        return t;
    }

    public static String concatWithStringBuffer(){
        StringBuffer sb = new StringBuffer("Java");
        for (int i=0; i<10000; i++){
            sb.append("Tpoint");
        }
        return sb.toString();
    }

    public static void main(String[] args){
        long startTime = System.currentTimeMillis();
        concatWithString();
        System.out.println("Time taken by Concating with String: "+(System.currentTimeMillis()-startTime)+"ms");
        startTime = System.currentTimeMillis();
        concatWithStringBuffer();
        System.out.println("Time taken by Concating with StringBuffer: "+(System.currentTimeMillis()-startTime)+"ms");
    }
}

Time taken by Concating with String: 578ms
Time taken by Concating with StringBuffer: 0ms

```

## String and StringBuffer HashCode Test

As you can see in the program given below, String returns new hashCode value when you concat string but StringBuffer returns same.

```

public class InstanceTest{
    public static void main(String args[]){
        System.out.println("Hashcode test of String:");
        String str="java";
        System.out.println(str.hashCode());
        str=str+"tpoint";
        System.out.println(str.hashCode());

        System.out.println("Hashcode test of StringBuffer:");
        StringBuffer sb=new StringBuffer("java");
        System.out.println(sb.hashCode());
        sb.append("tpoint");
        System.out.println(sb.hashCode());
    }
}

Hashcode test of String:
3254818
229541438

```

```

Hashcode test of StringBuffer:
118352462
118352462

```

## Difference between StringBuffer and StringBuilder

There are many differences between StringBuffer and StringBuilder. A list of differences between StringBuffer and StringBuilder are given below:

No.	StringBuffer	StringBuilder
1)	StringBuffer is <i>synchronized</i> i.e. thread safe. It means two threads can't call the methods of StringBuffer simultaneously.	StringBuilder is <i>non-synchronized</i> i.e. not thread safe. It means two threads can call the methods of StringBuilder simultaneously.
2)	StringBuffer is <i>less efficient</i> than StringBuilder.	StringBuilder is <i>more efficient</i> than StringBuffer.

## StringBuffer Example

```

public class BufferTest{
    public static void main(String[] args){
        StringBuffer buffer=new StringBuffer("hello");
        buffer.append("java");
        System.out.println(buffer);
    }
}

hellojava

```

## StringBuilder Example

```

public class BuilderTest{
    public static void main(String[] args){
        StringBuilder builder=new StringBuilder("hello");
        builder.append("java");
        System.out.println(builder);
    }
}

hellojava

```

## Performance Test of StringBuffer and StringBuilder

Let's see the code to check the performance of StringBuffer and StringBuilder classes.

```

public class ConcatTest{
    public static void main(String[] args){
        long startTime = System.currentTimeMillis();
        StringBuffer sb = new StringBuffer("Java");
        for (int i=0; i<10000; i++){
            sb.append("Tpoint");
        }
        System.out.println("Time taken by StringBuffer: " + (System.currentTimeMillis() - startTime) + "ms");
        startTime = System.currentTimeMillis();
        StringBuilder sb2 = new StringBuilder("Java");
        for (int i=0; i<10000; i++){
            sb2.append("Tpoint");
        }
        System.out.println("Time taken by StringBuilder: " + (System.currentTimeMillis() - startTime) + "ms");
    }
}

```

```
}
```

```
Time taken by StringBuffer: 16ms
```

```
Time taken by StringBuilder: 0ms
```

## How to create Immutable class?

There are many immutable classes like String, Boolean, Byte, Short, Integer, Long, Float, Double etc. In short, all the wrapper classes and String class is immutable. We can also create immutable class by creating final class that have final data members as the example given below:

### Example to create Immutable class

In this example, we have created a final class named Employee. It have one final datamember, a parameterized constructor and getter method.

```
public final class Employee{  
    final String pancardNumber;  
  
    public Employee(String pancardNumber){  
        this.pancardNumber=pancardNumber;  
    }  
  
    public String getPancardNumber(){  
        return pancardNumber;  
    }  
}
```

---

The above class is immutable because:

- The instance variable of the class is final i.e. we cannot change the value of it after creating an object.
- The class is final so we cannot create the subclass.
- There is no setter methods i.e. we have no option to change the value of the instance variable.

These points makes this class as immutable.

## Java `toString()` method

If you want to represent any object as a string, **`toString()` method** comes into existence.

The `toString()` method returns the string representation of the object.

If you print any object, java compiler internally invokes the `toString()` method on the object. So overriding the `toString()` method, returns the desired output, it can be the state of an object etc. depends on your implementation.

### Advantage of Java `toString()` method

By overriding the `toString()` method of the Object class, we can return values of the object, so we don't need to write much code.

---

### Understanding problem without `toString()` method

Let's see the simple code that prints reference.

```

class Student{
    int rollno;
    String name;
    String city;

    Student(int rollno, String name, String city){
        this.rollno=rollno;
        this.name=name;
        this.city=city;
    }

    public static void main(String args[]){
        Student s1=new Student(101,"Raj","lucknow");
        Student s2=new Student(102,"Vijay","ghaziabad");

        System.out.println(s1); //compiler writes here s1.toString()
        System.out.println(s2); //compiler writes here s2.toString()
    }
}

```

Output:  
Student@1fee6fc  
Student@1eed786

As you can see in the above example, printing s1 and s2 prints the hashCode values of the objects but I want to print the values of these objects. Since java compiler internally calls `toString()` method, overriding this method will return the specified values. Let's understand it with the example given below:

## Example of Java `toString()` method

## StringTokenizer in Java

The **java.util.StringTokenizer** class allows you to break a string into tokens. It is simple way to break string.

It doesn't provide the facility to differentiate numbers, quoted strings, identifiers etc. like StreamTokenizer class. We will discuss about the StreamTokenizer class in I/O chapter.

### Constructors of StringTokenizer class

There are 3 constructors defined in the StringTokenizer class.

Constructor	Description
<code>StringTokenizer(String str)</code>	creates StringTokenizer with specified string.
<code>StringTokenizer(String str, String delim)</code>	creates StringTokenizer with specified string and delimiter.
<code>StringTokenizer(String str, String delim, boolean returnValue)</code>	creates StringTokenizer with specified string, delimiter and returnValue. If return value is true, delimiter characters are considered to be tokens. If it is false, delimiter characters serve to separate tokens.

### Methods of StringTokenizer class

The 6 useful methods of StringTokenizer class are as follows:

Public method	Description
<code>boolean hasMoreTokens()</code>	checks if there is more tokens available.
<code>String nextToken()</code>	returns the next token from the StringTokenizer object.
<code>String nextToken(String delim)</code>	returns the next token based on the delimiter.

boolean hasMoreElements()	same as hasMoreTokens() method.
Object nextElement()	same as nextToken() but its return type is Object.
int countTokens()	returns the total number of tokens.

## Simple example of StringTokenizer class

Let's see the simple example of StringTokenizer class that tokenizes a string "my name is khan" on the basis of whitespace.

```
import java.util.StringTokenizer;
public class Simple{
    public static void main(String args[]){
        StringTokenizer st = new StringTokenizer("my name is khan","");
        while (st.hasMoreTokens()) {
            System.out.println(st.nextToken());
        }
    }
}

Output:my
      name
      is
      khan
```

## Example of nextToken(String delim) method of StringTokenizer class

```
import java.util.*;

public class Test {
    public static void main(String[] args) {
        StringTokenizer st = new StringTokenizer("my,name,is,khan");

        // printing next token
        System.out.println("Next token is : " + st.nextToken(","));
    }
}

Output:Next token is : my
```



**StringTokenizer class is deprecated now. It is recommended to use split() method of String class or regex (Regular Expression).**

## Java String charAt

The **java string charAt()** method returns a char value at the given index number. The index number starts from 0.

### Signature

The signature of string charAt() method is given below:

```
public char charAt(int index)
```

## Parameter

**index** : index number, starts with 0

---

## Returns

**char value**

---

## Specified by

**CharSequence** interface

---

## Throws

**IndexOutOfBoundsException** : if index is negative value or greater than this string length.

---

## Java String charAt() method example

```
public class CharAtExample{  
    public static void main(String args[]){  
        String name="javatpoint";  
        char ch=name.charAt(4); //returns the char value at the 4th index  
        System.out.println(ch);  
    }  
}
```

[Test it Now](#)

t

## Java String compareTo

The **java string compareTo()** method compares the given string with current string lexicographically. It returns positive number, negative number or 0.

If first string is greater than second string, it returns positive number (difference of character value). If first string is less than second string, it returns negative number and if first string is equal to second string, it returns 0.

```
s1 > s2 => positive number  
s1 < s2 => negative number  
s1 == s2 => 0
```

---

## Signature

```
public int compareTo(String anotherString)
```

---

## Parameters

**anotherString**: represents string that is to be compared with current string

---

## Returns

an integer value

---

## Java String compareTo() method example

```
public class LastIndexOfExample{  
    public static void main(String args[]){  
        String s1="hello";  
        String s2="hello";  
        String s3="meklo";  
        String s4="hemlo";  
        System.out.println(s1.compareTo(s2));  
        System.out.println(s1.compareTo(s3));  
        System.out.println(s1.compareTo(s4));  
    }  
}
```

Output:

```
0  
-5  
-1
```

## Java String concat

The **java string concat()** method *combines specified string at the end of this string*. It returns combined string. It is like appending another string.

---

### Signature

The signature of string concat() method is given below:

```
public String concat(String anotherString)
```

### Parameter

**anotherString** : another string i.e. to be combined at the end of this string.

---

### Returns

combined string

---

## Java String concat() method example

```
public class ConcatExample{  
    public static void main(String args[]){  
        String s1="java string";  
        s1.concat(" is immutable");  
        System.out.println(s1);  
        s1=s1.concat(" is immutable so assign it explicitly");  
        System.out.println(s1);  
    }  
}
```

**Test it Now**

```
java string  
java string is immutable so assign it explicitly
```

# Java String contains

The **java string contains()** method searches the sequence of characters in this string. It returns *true* if sequence of char values are found in this string otherwise returns *false*.

---

## Signature

The signature of string contains() method is given below:

```
public boolean contains(CharSequence sequence)
```

---

## Parameter

**sequence** : specifies the sequence of characters to be searched.

---

## Returns

**true** if sequence of char value exists, otherwise **false**.

---

## Throws

**NullPointerException** : if sequence is null.

---

## Java String contains() method example

```
class ContainsExample{  
    public static void main(String args[]){  
        String name="what do you know about me";  
        System.out.println(name.contains("do you know"));  
        System.out.println(name.contains("about"));  
        System.out.println(name.contains("hello"));  
    }  
}
```

**Test it Now**

```
true  
true  
false
```

# Java String endsWith

The **java string endsWith()** method checks if this string ends with given suffix. It returns true if this string ends with given suffix else returns false.

---

## Signature

The syntax or signature of endsWith() method is given below.

```
public boolean endsWith(String suffix)
```

---

## Parameter

**suffix** : Sequence of character

---

## Returns

true or false

---

## Java String endsWith() method example

```
public class EndsWithExample{  
    public static void main(String args[]){  
        String s1="java by javatpoint";  
        System.out.println(s1.endsWith("t"));  
        System.out.println(s1.endsWith("point"));  
    }  
}
```

Output:

```
true  
true
```

## Java String equals

The **java string equals()** method compares the two given strings based on the content of the string. If any character is not matched, it returns false. If all characters are matched, it returns true.

The String equals() method overrides the equals() method of Object class.

---

## Signature

```
public boolean equals(Object anotherObject)
```

---

## Parameter

**anotherObject** : another object i.e. compared with this string.

---

## Returns

**true** if characters of both strings are equal otherwise **false**.

---

## Overrides

equals() method of java Object class.

---

## Java String equals() method example

```
public class EqualsExample{  
    public static void main(String args[]){  
        String s1="javatpoint";  
        String s2="javatpoint";  
        String s3="JAVATPOINT";  
        String s4="python";  
        System.out.println(s1.equals(s2));//true because content and case is same  
        System.out.println(s1.equals(s3));//false because case is not same  
        System.out.println(s1.equals(s4));//false because content is not same  
    }  
}
```

[Test it Now](#)

```
true  
false  
false
```

## Java String format

The **java string format()** method returns the formatted string by given locale, format and arguments.

If you don't specify the locale in String.format() method, it uses default locale by calling `Locale.getDefault()` method.

The format() method of java language is like `sprintf()` function in c language and `printf()` method of java language.

---

### Signature

There are two type of string format() method:

```
public static String format(String format, Object... args)  
and,  
public static String format(Locale locale, String format, Object... args)
```

---

### Parameters

**locale** : specifies the locale to be applied on the format() method.

**format** : format of the string.

**args** : arguments for the format string. It may be zero or more.

---

### Returns

formatted string

---

### Throws

**NullPointerException** : if format is null.

**IllegalFormatException** : if format is illegal or incompatible.

---

## Java String format() method example

```
public class FormatExample{  
public static void main(String args[]){  
String name="sonoo";  
String sf1=String.format("name is %s",name);  
String sf2=String.format("value is %f",32.33434);  
String sf3=String.format("value is %32.12f",32.33434);//returns 12 char fractional part filling with 0  
  
System.out.println(sf1);  
System.out.println(sf2);  
System.out.println(sf3);  
}}
```

**Test it Now**

```
name is sonoo  
value is 32.334340  
value is 32.334340000000
```

## Java String getBytes()

The **java string getBytes()** method returns the byte array of the string. In other words, it returns sequence of bytes.

### Signature

There are 3 variant of getBytes() method. The signature or syntax of string getBytes() method is given below:

```
public byte[] getBytes()
public byte[] getBytes(Charset charset)
public byte[] getBytes(String charsetName) throws UnsupportedEncodingException
```

---

### Returns

sequence of bytes.

---

## Java String getBytes() method example

```
public class StringGetBytesExample{
public static void main(String args[]){
String s1="ABCDEFG";
byte[] barr=s1.getBytes();
for(int i=0;i
```

**Test it Now**

Output:

```
65
66
67
68
69
70
71
```

## Java String indexOf

The **java string indexOf()** method returns index of given character value or substring. If it is not found, it returns -1. The index counter starts from zero.

---

### Signature

There are 4 types of indexOf method in java. The signature of indexOf methods are given below:

No.	Method	Description
1	int indexOf(int ch)	returns index position for the given char value
2	int indexOf(int ch, int fromIndex)	returns index position for the given char value and from index
3	int indexOf(String substring)	returns index position for the given substring
4	int indexOf(String substring, int fromIndex)	returns index position for the given substring and from index

## Parameters

**ch:** char value i.e. a single character e.g. 'a'

**fromIndex:** index position from where index of the char value or substring is returned

**substring:** substring to be searched in this string

## Returns

index of the string

## Java String indexOf() method example

```
public class IndexOfExample{  
    public static void main(String args[]){  
        String s1="this is index of example";  
        //passing substring  
        int index1=s1.indexOf("is");//returns the index of is substring  
        int index2=s1.indexOf("index");//returns the index of index substring  
        System.out.println(index1+" "+index2);//2 8  
  
        //passing substring with from index  
        int index3=s1.indexOf("is",4);//returns the index of is substring after 4th index  
        System.out.println(index3);//5 i.e. the index of another is  
  
        //passing char value  
        int index4=s1.indexOf('s');//returns the index of s char value  
        System.out.println(index4);//3  
    }  
}
```

**Test it Now**

```
2 8  
5  
3
```

## Java String intern

The **java string intern()** method returns the interned string. It returns the canonical representation of string.

It can be used to return string from pool memory, if it is created by new keyword.

## Signature

The signature of intern method is given below:

```
public String intern()
```

## Returns

interned string

## Java String intern() method example

```
public class InternExample{  
    public static void main(String args[]){  
        String s1=new String("hello");  
        String s2="hello";  
        String s3=s1.intern(); //returns string from pool, now it will be same as s2  
        System.out.println(s1==s2); //false because reference is different  
        System.out.println(s2==s3); //true because reference is same  
    }  
}
```

**Test it Now**

```
false  
true
```

## Java String isEmpty

The **java string isEmpty()** method checks if this string is empty. It returns *true*, if length of string is 0 otherwise *false*.

The isEmpty() method of String class is included in java string since JDK 1.6.

---

### Signature

The signature or syntax of string isEmpty() method is given below:

```
public boolean isEmpty()
```

---

### Returns

true if length is 0 otherwise false.

---

### Since

**1.6**

---

## Java String isEmpty() method example

```
public class IsEmptyExample{  
    public static void main(String args[]){  
        String s1="";  
        String s2="javatpoint";  
  
        System.out.println(s1.isEmpty());  
        System.out.println(s2.isEmpty());  
    }  
}
```

**Test it Now**

```
true  
false
```

## Java String join

The **java string join()** method returns a string joined with given delimiter. In string join method, delimiter is copied for each elements.

In case of null element, "null" is added. The join() method is included in java string since JDK 1.8.

There are two types of join() methods in java string.

---

## Signature

The signature or syntax of string join method is given below:

```
public static String join(CharSequence delimiter, CharSequence... elements)  
and  
public static String join(CharSequence delimiter, Iterable elements)
```

---

## Parameters

**delimiter** : char value to be added with each element

**elements** : char value to be attached with delimiter

---

## Returns

joined string with delimiter

---

## Throws

**NullPointerException** if element or delimiter is null.

---

## Since

**1.8**

---

## Java String join() method example

```
public class StringJoinExample{  
public static void main(String args[]){  
String joinString1=String.join("-","welcome","to","javatpoint");  
System.out.println(joinString1);  
}}
```

**Test it Now**

welcome-to-javatpoint

## Java String lastIndexOf

The **java string lastIndexOf()** method returns last index of the given character value or substring. If it is not found, it returns -1. The index counter starts from zero.

---

## Signature

There are 4 types of lastIndexOf method in java. The signature of lastIndexOf methods are given below:

No.	Method	Description
1	int lastIndexOf(int ch)	returns last index position for the given char value
2	int lastIndexOf(int ch, int fromIndex)	returns last index position for the given char value and from index

3	int lastIndexOf(String substring)	returns last index position for the given substring
4	int lastIndexOf(String substring, int fromIndex)	returns last index position for the given substring and from index

## Parameters

**ch:** char value i.e. a single character e.g. 'a'

**fromIndex:** index position from where index of the char value or substring is returned

**substring:** substring to be searched in this string

## Returns

last index of the string

## Java String lastIndexOf() method example

```
public class LastIndexOfExample{
public static void main(String args[]){
String s1="this is index of example";//there are 2 's' characters in this sentence
int index1=s1.lastIndexOf('s');//returns last index of 's' char value
System.out.println(index1);//6
}}
```

Output:

6

## Java String length

The **java string length()** method length of the string. It returns count of total number of characters. The length of java string is same as the unicode code units of the string.

## Signature

The signature of the string length() method is given below:

```
public int length()
```

## Specified by

CharSequence interface

## Returns

length of characters

## Java String length() method example

```
public class LengthExample{
public static void main(String args[]){
String s1="javatpoint";
String s2="python";
System.out.println("string length is: "+s1.length());//10 is the length of javatpoint string
}}
```

```
System.out.println("string length is: "+s2.length());//6 is the length of python string
}}
```

**Test it Now**

```
string length is: 10
string length is: 6
```

## Java String replace

The **java string replace()** method returns a string replacing all the old char or CharSequence to new char or CharSequence.

Since JDK 1.5, a new replace() method is introduced, allowing you to replace a sequence of char values.

### Signature

There are two type of replace methods in java string.

```
public String replace(char oldChar, char newChar)
and
public String replace(CharSequence target, CharSequence replacement)
```

The second replace method is added since JDK 1.5.

### Parameters

**oldChar** : old character

**newChar** : new character

**target** : target sequence of characters

**replacement** : replacement sequence of characters

### Returns

replaced string

### Java String replace(char old, char new) method example

```
public class ReplaceExample1{
public static void main(String args[]){
String s1="javatpoint is a very good website";
String replaceString=s1.replace('a','e');//replaces all occurrences of 'a' to 'e'
System.out.println(replaceString);
}}
```

**Test it Now**

```
jevetpoint is e very good website
```

### Java String replace(CharSequence target, CharSequence replacement) method example

```
public class ReplaceExample2{
public static void main(String args[]){
String s1="my name is khan my name is java";
String replaceString=s1.replace("is","was");//replaces all occurrences of "is" to "was"
System.out.println(replaceString);
}}
```

## Test it Now

```
my name was khan my name was java
```

# Java String replaceAll

The **java string replaceAll()** method returns a string replacing all the sequence of characters matching regex and replacement string.

## Signature

```
public String replaceAll(String regex, String replacement)
```

## Parameters

**regex** : regular expression

**replacement** : replacement sequence of characters

## Returns

replaced string

### Java String replaceAll() example: replace character

Let's see an example to replace all the occurrences of **a single character**.

```
public class ReplaceAllExample1{  
    public static void main(String args[]){  
        String s1="javatpoint is a very good website";  
        String replaceString=s1.replaceAll("a","e");//replaces all occurrences of "a" to "e"  
        System.out.println(replaceString);  
    }  
  
    jevetpoint is e very good website
```

### Java String replaceAll() example: replace word

Let's see an example to replace all the occurrences of **single word or set of words**

```
public class ReplaceAllExample2{  
    public static void main(String args[]){  
        String s1="My name is Khan. My name is Bob. My name is Sonoo.";  
        String replaceString=s1.replaceAll("is","was");//replaces all occurrences of "is" to "was"  
        System.out.println(replaceString);  
    }  
  
    My name was Khan. My name was Bob. My name was Sonoo.
```

### Java String replaceAll() example: remove white spaces

Let's see an example to remove all the occurrences of **white spaces**.

```
public class ReplaceAllExample3{  
    public static void main(String args[]){  
        String s1="My name is Khan. My name is Bob. My name is Sonoo.";  
        String replaceString=s1.replaceAll("\\s","");
        System.out.println(replaceString);  
    }  
  
    MynamewasKhan.MynamewasBob.MynamewasSonoo.
```

---

[Click me to know about regex](#)

---

## Java String split

The **java string split()** method splits this string against given regular expression and returns a char array.

---

### Signature

There are two signature for split() method in java string.

```
public String split(String regex)  
and,  
public String split(String regex, int limit)
```

---

### Parameter

**regex** : regular expression to be applied on string.

**limit** : limit for the number of strings in array. If it is zero, it will returns all the strings matching regex.

---

### Returns

array of strings

---

### Throws

**PatternSyntaxException** if pattern for regular expression is invalid

---

### Since

1.4

---

## Java String split() method example

The given example returns total number of words in a string excluding space only. It also includes special characters.

```
public class SplitExample{  
public static void main(String args[]){  
String s1="java string split method by javatpoint";  
String[] words=s1.split("\\s");//splits the string based on string  
//using java foreach loop to print elements of string array  
for(String w:words){  
System.out.println(w);  
}  
}}
```

**Test it Now**

```
java  
string  
split  
method  
by  
javatpoint
```

## Java String split() method with regex and length example

```
public class SplitExample2{  
    public static void main(String args[]){  
        String s1="welcome to split world";  
        System.out.println("returning words:");  
        for(String w:s1.split("\\s",0)){  
            System.out.println(w);  
        }  
        System.out.println("returning words:");  
        for(String w:s1.split("\\s",1)){  
            System.out.println(w);  
        }  
        System.out.println("returning words:");  
        for(String w:s1.split("\\s",2)){  
            System.out.println(w);  
        }  
  
    }}  
}
```

**Test it Now**

```
returning words:  
welcome  
to  
split  
world  
returning words:  
welcome to split world  
returning words:  
welcome  
to split world
```

## Java String startsWith

The **java string startsWith()** method checks if this string starts with given prefix. It returns true if this string starts with given prefix else returns false.

### Signature

The syntax or signature of startWith() method is given below.

```
public boolean startsWith(String prefix)  
public boolean startsWith(String prefix, int offset)
```

### Parameter

**prefix** : Sequence of character

### Returns

true or false

## Java String startsWith() method example

```
public class StartsWithExample{
public static void main(String args[]){
String s1="java string split method by javatpoint";
System.out.println(s1.startsWith("ja"));
System.out.println(s1.startsWith("java string"));
}}
```

Output:

```
true
true
```

## Java String substring

The **java string substring()** method returns a part of the string.

We pass begin index and end index number position in the java substring method where start index is inclusive and end index is exclusive.  
In other words, start index starts from 0 whereas end index starts from 1.

There are two types of substring methods in java string.

---

### Signature

```
public String substring(int startIndex)
and
public String substring(int startIndex, int endIndex)
```

If you don't specify endIndex, java substring() method will return all the characters from startIndex.

---

### Parameters

**startIndex** : starting index is inclusive

**endIndex** : ending index is exclusive

---

### Returns

specified string

---

### Throws

**StringIndexOutOfBoundsException** if start index is negative value or end index is lower than starting index.

---

## Java String substring() method example

```
public class SubstringExample{
public static void main(String args[]){
String s1="javatpoint";
System.out.println(s1.substring(2,4));//returns va
System.out.println(s1.substring(2));//returns vatpoint
}}
```

**Test it Now**

```
va
vatpoint
```

## Java String toCharArray

The **java string toCharArray()** method converts this string into character array. It returns a newly created character array, its length is similar to this string and its contents are initialized with the characters of this string.

### Signature

The signature or syntax of string toCharArray() method is given below:

```
public char[] toCharArray()
```

---

### Returns

character array

## Java String toCharArray() method example

```
public class StringToCharArrayExample{  
    public static void main(String args[]){  
        String s1="hello";  
        char[] ch=s1.toCharArray();  
        for(int i=0;i  
Output:  
hello
```

## Java String toLowerCase()

The **java string toLowerCase()** method returns the string in lowercase letter. In other words, it converts all characters of the string into lower case letter.

The toLowerCase() method works same as toLowerCase(Locale.getDefault()) method. It internally uses the default locale.

---

### Signature

There are two variant of toLowerCase() method. The signature or syntax of string toLowerCase() method is given below:

```
public String toLowerCase()  
public String toLowerCase(Locale locale)
```

The second method variant of toLowerCase(), converts all the characters into lowercase using the rules of given Locale.

---

### Returns

string in lowercase letter.

## Java String toLowerCase() method example

```
public class StringLowerExample{  
    public static void main(String args[]){  
        String s1="JAVATPOINT HELLO strIng";  
        String s1lower=s1.toLowerCase();
```

```
System.out.println(s1lower);
}}
```

**Test it Now**

Output:

```
javatpoint hello string
```

## Java String toUpperCase

The **java string toUpperCase()** method returns the string in uppercase letter. In other words, it converts all characters of the string into upper case letter.

The toUpperCase() method works same as toUpperCase(Locale.getDefault()) method. It internally uses the default locale.

### Signature

There are two variant of toUpperCase() method. The signature or syntax of string toUpperCase() method is given below:

```
public String toUpperCase()
public String toUpperCase(Locale locale)
```

The second method variant of toUpperCase(), converts all the characters into uppercase using the rules of given Locale.

### Returns

string in uppercase letter.

## Java String toUpperCase() method example

```
public class StringUpperExample{
public static void main(String args[]){
String s1="hello string";
String slupper=s1.toUpperCase();
System.out.println(slupper);
}}
```

**Test it Now**

Output:

```
HELLO STRING
```

## Java String trim

The **java string trim()** method eliminates leading and trailing spaces. The unicode value of space character is '\u0020'. The trim() method in java string checks this unicode value before and after the string, if it exists then removes the spaces and returns the omitted string.



*The string trim() method doesn't omit middle spaces.*

### Signature

The signature or syntax of string trim method is given below:

```
public String trim()
```

---

## Returns

string with omitted leading and trailing spaces

---

## Java String trim() method example

```
public class StringTrimExample{  
    public static void main(String args[]){  
        String s1=" hello string ";  
        System.out.println(s1+"javatpoint");//without trim()  
        System.out.println(s1.trim()+"javatpoint");//with trim()  
    }  
}
```

### Test it Now

```
hello string javatpoint  
hello stringjavatpoint
```

## Java String valueOf

The **java string valueOf()** method converts different types of values into string. By the help of string valueOf() method, you can convert int to string, long to string, boolean to string, character to string, float to string, double to string, object to string and char array to string.

## Signature

The signature or syntax of string valueOf() method is given below:

```
public static String valueOf(boolean b)  
public static String valueOf(char c)  
public static String valueOf(char[] c)  
public static String valueOf(int i)  
public static String valueOf(long l)  
public static String valueOf(float f)  
public static String valueOf(double d)  
public static String valueOf(Object o)
```

---

## Returns

string representation of given value

---

## Java String valueOf() method example

```
public class StringValueOfExample{  
    public static void main(String args[]){  
        int value=30;  
        String s1=String.valueOf(value);  
        System.out.println(s1+10);//concatenating string with 10  
    }  
}
```

Output:

3010

# Java Regex

The **Java Regex** or Regular Expression is an API to *define pattern for searching or manipulating strings* .

It is widely used to define constraint on strings such as password and email validation. After learning java regex tutorial, you will be able to test your own regular expressions by the Java Regex Tester Tool.

Java Regex API provides 1 interface and 3 classes in **java.util.regex** package.

## java.util.regex package

It provides following classes and interface for regular expressions. The Matcher and Pattern classes are widely used in java regular expression.

1. MatchResult interface
2. Matcher class
3. Pattern class
4. PatternSyntaxException class

## Matcher class

It implements **MatchResult** interface. It is a *regex engine* i.e. used to perform match operations on a character sequence.

No.	Method	Description
1	boolean matches()	test whether the regular expression matches the pattern.
2	boolean find()	finds the next expression that matches the pattern.
3	boolean find(int start)	finds the next expression that matches the pattern from the given start number.

## Pattern class

It is the *compiled version of a regular expression*. It is used to define a pattern for the regex engine.

No.	Method	Description
1	static Pattern compile(String regex)	compiles the given regex and return the instance of pattern.
2	Matcher matcher(CharSequence input)	creates a matcher that matches the given input with pattern.
3	static boolean matches(String regex, CharSequence input)	It works as the combination of compile and matcher methods. It compiles the regular expression and matches the given input with the pattern.
4	String[] split(CharSequence input)	splits the given input string around matches of given pattern.
5	String pattern()	returns the regex pattern.

## Example of Java Regular Expressions

There are three ways to write the regex example in java.

```
import java.util.regex.*;
public class RegexExample1{
public static void main(String args[]){
//1st way
Pattern p = Pattern.compile(".s");//. represents single character
Matcher m = p.matcher("as");
boolean b = m.matches();

//2nd way
boolean b2=Pattern.compile(".s").matcher("as").matches();
```

```
//3rd way
boolean b3 = Pattern.matches(".", "as");

System.out.println(b+ " "+b2+ " "+b3);
}}
```

**Test it Now**

## Output

```
true true true
```

## Regular Expression . Example

The . (dot) represents a single character.

```
import java.util.regex.*;
class RegexExample2{
public static void main(String args[]){
System.out.println(Pattern.matches(".", "as")); //true (2nd char is s)
System.out.println(Pattern.matches(".", "mk")); //false (2nd char is not s)
System.out.println(Pattern.matches(".", "mst")); //false (has more than 2 char)
System.out.println(Pattern.matches(".", "amms")); //false (has more than 2 char)
System.out.println(Pattern.matches("..s", "mas")); //true (3rd char is s)
}}
```

**Test it Now**

## Regex Character classes

No.	Character Class	Description
1	[abc]	a, b, or c (simple class)
2	[^abc]	Any character except a, b, or c (negation)
3	[a-zA-Z]	a through z or A through Z, inclusive (range)
4	[a-d[m-p]]	a through d, or m through p: [a-dm-p] (union)
5	[a-z&&[def]]	d, e, or f (intersection)
6	[a-z&&[^bc]]	a through z, except for b and c: [ad-z] (subtraction)
7	[a-z&&[^m-p]]	a through z, and not m through p: [a-lq-z](subtraction)

## Regular Expression Character classes Example

```
import java.util.regex.*;
class RegexExample3{
public static void main(String args[]){
System.out.println(Pattern.matches("[amn]", "abcd")); //false (not a or m or n)
System.out.println(Pattern.matches("[amn]", "a")); //true (among a or m or n)
System.out.println(Pattern.matches("[amn]", "ammrna")); //false (m and a comes more than once)
}}
```

**Test it Now**

## Regex Quantifiers

The quantifiers specify the number of occurrences of a character.

Regex	Description
X?	X occurs once or not at all

X+	X occurs once or more times
X*	X occurs zero or more times
X{n}	X occurs n times only
X{n,}	X occurs n or more times
X{y,z}	X occurs at least y times but less than z times

## Regular Expression Character classes and Quantifiers Example

```

import java.util.regex.*;
class RegexExample4{
public static void main(String args[]){
System.out.println("? quantifier ....");
System.out.println(Pattern.matches("[amn]?", "a")); //true (a or m or n comes one time)
System.out.println(Pattern.matches("[amn]?", "aaa")); //false (a comes more than one time)
System.out.println(Pattern.matches("[amn]?", "aammnn")); //false (a m and n comes more than one time)
System.out.println(Pattern.matches("[amn]?", "aazzta")); //false (a comes more than one time)
System.out.println(Pattern.matches("[amn]?", "am")); //false (a or m or n must come one time)

System.out.println("+ quantifier ....");
System.out.println(Pattern.matches("[amn]+", "a")); //true (a or m or n once or more times)
System.out.println(Pattern.matches("[amn]+", "aaa")); //true (a comes more than one time)
System.out.println(Pattern.matches("[amn]+", "aammnn")); //true (a or m or n comes more than once)
System.out.println(Pattern.matches("[amn]+", "aazzta")); //false (z and t are not matching pattern)

System.out.println("* quantifier ....");
System.out.println(Pattern.matches("[amn]*", "ammmna")); //true (a or m or n may come zero or more times)
}

```

**Test it Now**

## Regex Metacharacters

The regular expression metacharacters work as a short codes.

Regex	Description
.	Any character (may or may not match terminator)
\d	Any digits, short of [0-9]
\D	Any non-digit, short for [^0-9]
\s	Any whitespace character, short for [\t\n\x0B\f\r]
\S	Any non-whitespace character, short for [^\s]
\w	Any word character, short for [a-zA-Z_0-9]
\W	Any non-word character, short for [^\w]
\b	A word boundary
\B	A non word boundary

## Regular Expression Metacharacters Example

```

import java.util.regex.*;
class RegexExample5{
public static void main(String args[]){
System.out.println("metacharacters d...."); \\d means digit
}

```

```

System.out.println(Pattern.matches("\\d", "abc")); //false (non-digit)
System.out.println(Pattern.matches("\\d", "1")); //true (digit and comes once)
System.out.println(Pattern.matches("\\d", "4443")); //false (digit but comes more than once)
System.out.println(Pattern.matches("\\d", "323abc")); //false (digit and char)

System.out.println("metacharacters D...."); \\D means non-digit

System.out.println(Pattern.matches("\\D", "abc")); //false (non-digit but comes more than once)
System.out.println(Pattern.matches("\\D", "1")); //false (digit)
System.out.println(Pattern.matches("\\D", "4443")); //false (digit)
System.out.println(Pattern.matches("\\D", "323abc")); //false (digit and char)
System.out.println(Pattern.matches("\\D", "m")); //true (non-digit and comes once)

System.out.println("metacharacters D with quantifier....");
System.out.println(Pattern.matches("\\D*", "mak")); //true (non-digit and may come 0 or more times)

```

}}

**Test it Now**

## Regular Expression Question 1

```

/*Create a regular expression that accepts alpha numeric characters only. Its
length must be 6 characters long only.*/

import java.util.regex.*;
class RegexExample6{
public static void main(String args[]){
System.out.println(Pattern.matches("[a-zA-Z0-9]{6}", "arun32")); //true
System.out.println(Pattern.matches("[a-zA-Z0-9]{6}", "kkvarun32")); //false (more than 6 char)
System.out.println(Pattern.matches("[a-zA-Z0-9]{6}", "JA2Uk2")); //true
System.out.println(Pattern.matches("[a-zA-Z0-9]{6}", "arun$2")); //false ($ is not matched)
}}

```

**Test it Now**

## Regular Expression Question 2

```

/*Create a regular expression that accepts 10 digit numeric characters
starting with 7, 8 or 9 only.*/

import java.util.regex.*;
class RegexExample7{
public static void main(String args[]){
System.out.println("by character classes and quantifiers ...");
System.out.println(Pattern.matches("[789]{1}[0-9]{9}", "9953038949")); //true
System.out.println(Pattern.matches("[789][0-9]{9}", "9953038949")); //true

System.out.println(Pattern.matches("[789][0-9]{9}", "99530389490")); //false (11 characters)
System.out.println(Pattern.matches("[789][0-9]{9}", "6953038949")); //false (starts from 6)
System.out.println(Pattern.matches("[789][0-9]{9}", "8853038949")); //true

System.out.println("by metacharacters ...");
System.out.println(Pattern.matches("[789]{1}\\d{9}", "8853038949")); //true
System.out.println(Pattern.matches("[789]{1}\\d{9}", "3853038949")); //false (starts from 3)
}}

```

**Test it Now**

# Exception Handling in Java

The **exception handling in java** is one of the powerful *mechanism to handle the runtime errors* so that normal flow of the application can be maintained.

In this page, we will learn about java exception, its type and the difference between checked and unchecked exceptions.

---

## What is exception

**Dictionary Meaning:** Exception is an abnormal condition.

In java, exception is an event that disrupts the normal flow of the program. It is an object which is thrown at runtime.

---

## What is exception handling

Exception Handling is a mechanism to handle runtime errors such as ClassNotFound, IO, SQL, Remote etc.

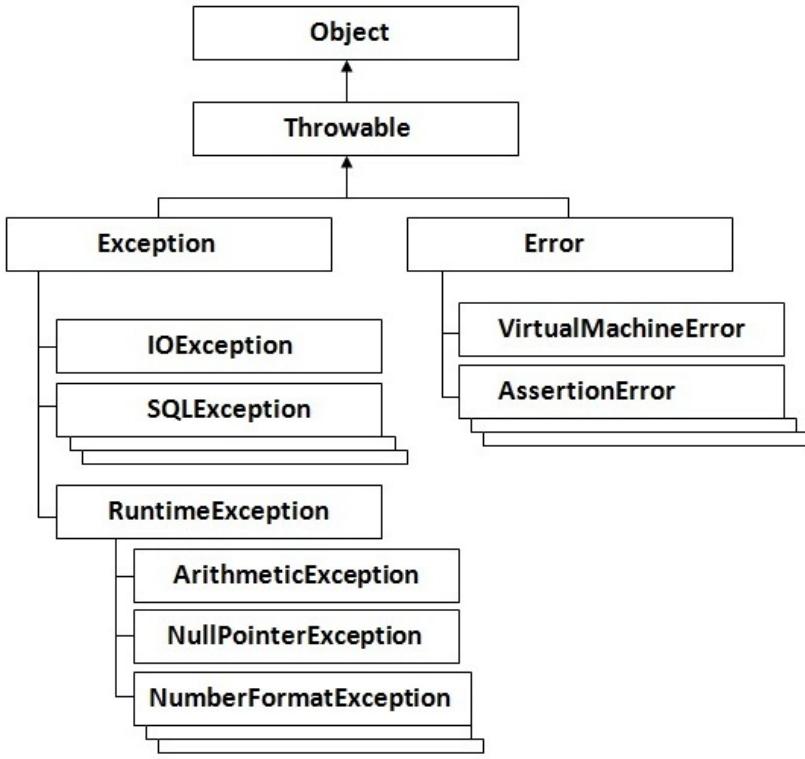
### Advantage of Exception Handling

The core advantage of exception handling is **to maintain the normal flow of the application**. Exception normally disrupts the normal flow of the application that is why we use exception handling. Let's take a scenario:

```
statement 1;  
statement 2;  
statement 3;  
statement 4;  
statement 5;//exception occurs  
statement 6;  
statement 7;  
statement 8;  
statement 9;  
statement 10;
```

Suppose there is 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.

## Hierarchy of Java Exception classes



## Types of Exception

There are mainly two types of exceptions: checked and unchecked where error is considered as unchecked exception. The sun microsystem says there are three types of exceptions:

1. Checked Exception
2. Unchecked Exception
3. Error

## Difference between checked and unchecked exceptions

### 1) Checked Exception

The classes that extend **Throwable** class except **RuntimeException** and **Error** are known as checked exceptions e.g. **IOException**, **SQLException** etc. Checked exceptions are checked at compile-time.

### 2) Unchecked Exception

The classes that extend **RuntimeException** are known as unchecked exceptions e.g. **ArithmetiException**, **NullPointerException**, **ArrayIndexOutOfBoundsException** etc. Unchecked exceptions are not checked at compile-time rather they are checked at runtime.

### 3) Error

Error is irrecoverable e.g. **OutOfMemoryError**, **VirtualMachineError**, **AssertionError** etc.

## Common scenarios where exceptions may occur

There are given some scenarios where unchecked exceptions can occur. They are as follows:

### 1) Scenario where **ArithmetiException** occurs

If we divide any number by zero, there occurs an **ArithmetiException**.

```
int a=50/0;//ArithmetiException
```

### 2) Scenario where **NullPointerException** occurs

If we have null value in any variable, performing any operation by the variable occurs an NullPointerException.

```
String s=null;
System.out.println(s.length());//NullPointerException
```

---

### 3) Scenario where NumberFormatException occurs

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.

```
String s="abc";
int i=Integer.parseInt(s);//NumberFormatException
```

---

### 4) Scenario where ArrayIndexOutOfBoundsException occurs

If you are inserting any value in the wrong index, it would result ArrayIndexOutOfBoundsException as shown below:

```
int a[]={};new int[5];
a[10]=50; //ArrayIndexOutOfBoundsException
```

---

## Java Exception Handling Keywords

There are 5 keywords used in java exception handling.

1. try
2. catch
3. finally
4. throw
5. throws

## Java try-catch

### 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.

#### Syntax of java try-catch

```
try{
//code that may throw exception
}catch(Exception_class_Name ref){}
```

#### Syntax of try-finally block

```
try{
//code that may throw exception
}finally{}
```

### Java catch block

### Java catch multiple exceptions

### Java Multi catch block

If you have to perform different tasks at the occurrence of different Exceptions, use java multi catch block.

Let's see a simple example of java multi-catch block.

```
public class TestMultipleCatchBlock{  
    public static void main(String args[]){  
        try{  
            int a[]={};  
            a[5]=30/0;  
        }  
        catch(ArithmaticException e){System.out.println("task1 is completed");}  
        catch(ArrayIndexOutOfBoundsException e){System.out.println("task 2 completed");}  
        catch(Exception e){System.out.println("common task completed");}  
  
        System.out.println("rest of the code...");  
    }  
}
```

#### Test it Now

Output:  
task1 completed  
rest of the code...



**Rule:** At a time only one Exception is occurred and at a time only one catch block is executed.



**Rule:** All catch blocks must be ordered from most specific to most general i.e. catch for ArithmaticException must come before catch for Exception .

```
class TestMultipleCatchBlock1{  
    public static void main(String args[]){  
        try{  
            int a[]={};  
            a[5]=30/0;  
        }  
        catch(Exception e){System.out.println("common task completed");}  
        catch(ArithmaticException e){System.out.println("task1 is completed");}  
        catch(ArrayIndexOutOfBoundsException e){System.out.println("task 2 completed");}  
        System.out.println("rest of the code...");  
    }  
}
```

#### Test it Now

Output:

Compile-time error

## Java Nested try block

The try block within a try block is known as nested try block in java.

### Why use nested try block

Sometimes a situation may arise where a part of a block may cause one error and the entire block itself may cause another error. In such cases, exception handlers have to be nested.

### Syntax:

```

.....
try
{
    statement 1;
    statement 2;
    try
    {
        statement 1;
        statement 2;
    }
    catch(Exception e)
    {
    }
}
catch(Exception e)
{
}
.....

```

## Java nested try example

Let's see a simple example of java nested try block.

```

class Excep6{
    public static void main(String args[]){
        try{
            try{
                System.out.println("going to divide");
                int b =39/0;
            }catch(ArithmetricException e){System.out.println(e);}

            try{
                int a[] =new int[5];
                a[5]=4;
            }catch(ArrayIndexOutOfBoundsException e){System.out.println(e);}

            System.out.println("other statement");
        }catch(Exception e){System.out.println("handede");}
        System.out.println("normal flow..");
    }
}

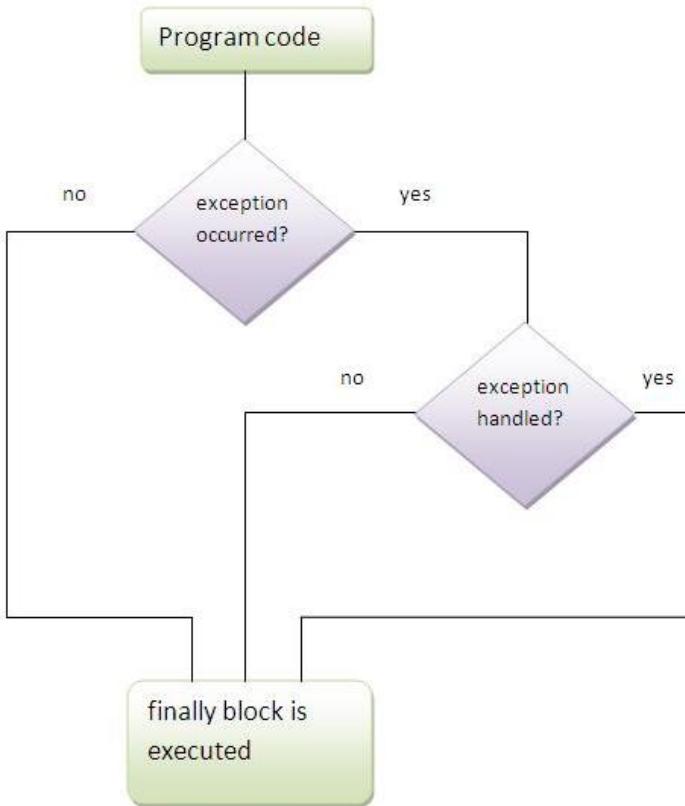
```

## Java finally block

**Java finally block** is a block that is used *to execute important code* such as closing connection, stream etc.

Java finally block is always executed whether exception is handled or not.

Java finally block must be followed by try or catch block.



**Note:** If you don't handle exception, before terminating the program, JVM executes finally block(if any).

## Why use java finally

- Finally block in java can be used to put "cleanup" code such as closing a file, closing connection etc.

## Usage of Java finally

Let's see the different cases where java finally block can be used.

### Case 1

Let's see the java finally example where **exception doesn't occur**.

```

class TestFinallyBlock{
    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..."); 
    }
}
  
```

**Test it Now**

Output:

```

finally block is always executed
rest of the code...
  
```

### Case 2

Let's see the java finally example where **exception occurs and not handled**.

```

class TestFinallyBlock1{
    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..."); 
    }
}

```

**Test it Now**

Output:finally block is always executed  
Exception in thread main java.lang.ArithmetricException:/ by zero

### Case 3

Let's see the java finally example where **exception occurs and handled**.

```

public class TestFinallyBlock2{
    public static void main(String args[]){
        try{
            int data=25/0;
            System.out.println(data);
        }
        catch(ArithmetricException e){System.out.println(e);}
        finally{System.out.println("finally block is always executed");}
        System.out.println("rest of the code..."); 
    }
}

```

**Test it Now**

Output:Exception in thread main java.lang.ArithmetricException:/ by zero  
finally block is always executed  
rest of the code...



**Rule:** For each try block there can be zero or more catch blocks, but only one finally block.



**Note:** The finally block will not be executed if program exits(either by calling System.exit() or by causing a fatal error that causes the process to abort).

## Java throw exception

### Java throw keyword

The Java throw keyword is used to explicitly throw an exception.

We can throw either checked or unchecked exception in java by throw keyword. The throw keyword is mainly used to throw custom exception. We will see custom exceptions later.

The syntax of java throw keyword is given below.

```
throw exception;
```

Let's see the example of throw IOException.

```
throw new IOException("sorry device error);
```

## java throw keyword example

In this example, we have created the validate method that takes integer value as a parameter. If the age is less than 18, we are throwing the ArithmeticException otherwise print a message welcome to vote.

```
public class TestThrow1{  
    static void validate(int age){  
        if(age<18)  
            throw new ArithmeticException("not valid");  
        else  
            System.out.println("welcome to vote");  
    }  
    public static void main(String args[]){  
        validate(13);  
        System.out.println("rest of the code...");  
    }  
}
```

**Test it Now**

Output:

```
Exception in thread main java.lang.ArithmetricException:not valid
```

## Java Exception propagation

An exception is first thrown from the top of the stack and if it is not caught, it drops down the call stack to the previous method,If not caught there, the exception again drops down to the previous method, and so on until they are caught or until they reach the very bottom of the call stack.This is called exception propagation.



**Rule: By default Unchecked Exceptions are forwarded in calling chain (propagated).**

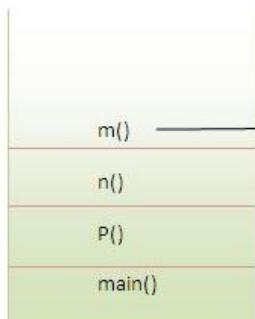
### Program of Exception Propagation

```
class TestExceptionPropagation1{  
    void m(){  
        int data=50/0;  
    }  
    void n(){  
        m();  
    }  
    void p(){  
        try{  
            n();  
        }catch(Exception e){System.out.println("exception handled");}  
    }  
    public static void main(String args[]){  
        TestExceptionPropagation1 obj=new TestExceptionPropagation1();  
        obj.p();  
        System.out.println("normal flow...");  
    }  
}
```

**Test it Now**

Output:exception handled

normal flow...



Call Stack

In the above example exception occurs in m() method where it is not handled, so it is propagated to previous n() method where it is not handled, again it is propagated to p() method where exception is handled.

Exception can be handled in any method in call stack either in main() method, p() method, n() method or m() method.



**Rule: By default, Checked Exceptions are not forwarded in calling chain (propagated).**

#### **Program which describes that checked exceptions are not propagated**

```
class TestExceptionPropagation2{  
    void m(){  
        throw new java.io.IOException("device error");//checked exception  
    }  
    void n(){  
        m();  
    }  
    void p(){  
        try{  
            n();  
        }catch(Exception e){System.out.println("exception handled");}  
    }  
    public static void main(String args[]){  
        TestExceptionPropagation2 obj=new TestExceptionPropagation2();  
        obj.p();  
        System.out.println("normal flow");  
    }  
}
```

**Test it Now**

Output: Compile Time Error

## Java throws keyword

The **Java throws keyword** is used to declare an exception. It gives an information to the programmer that there may occur an exception so it is better for the programmer to provide the exception handling code so that normal flow can be maintained.

Exception Handling is mainly used to handle the checked exceptions. If there occurs any unchecked exception such as NullPointerException, it is programmers fault that he is not performing check up before the code being used.

### Syntax of java throws

```
return_type method_name() throws exception_class_name{
```

```
//method code  
}
```

## Which exception should be declared

**Ans)** checked exception only, because:

- **unchecked Exception:** under your control so correct your code.
- **error:** beyond your control e.g. you are unable to do anything if there occurs VirtualMachineError or StackOverflowError.

## Advantage of Java throws keyword

Now Checked Exception can be propagated (forwarded in call stack).

It provides information to the caller of the method about the exception.

## Java throws example

Let's see the example of java throws clause which describes that checked exceptions can be propagated by throws keyword.

```
import java.io.IOException;  
class Testthrows1{  
    void m()throws IOException{  
        throw new IOException("device error");//checked exception  
    }  
    void n()throws IOException{  
        m();  
    }  
    void p(){  
        try{  
            n();  
        }catch(Exception e){System.out.println("exception handled");}  
    }  
    public static void main(String args[]){  
        Testthrows1 obj=new Testthrows1();  
        obj.p();  
        System.out.println("normal flow...");  
    }  
}
```

**Test it Now**

Output:

```
exception handled  
normal flow...
```



**Rule:** If you are calling a method that declares an exception, you must either caught or declare the exception.

There are two cases:

1. **Case1:** You caught the exception i.e. handle the exception using try/catch.
2. **Case2:** You declare the exception i.e. specifying throws with the method.

## Case1: You handle the exception

- In case you handle the exception, the code will be executed fine whether exception occurs during the program or not.

```
import java.io.*;  
class M{  
    void method()throws IOException{  
        throw new IOException("device error");  
    }  
}
```

```

}

}

public class Testthrows2{
    public static void main(String args[]){
        try{
            M m=new M();
            m.method();
        }catch(Exception e){System.out.println("exception handled");}
        System.out.println("normal flow...");
    }
}

```

**Test it Now**

Output:exception handled  
normal flow...

---

## Case2: You declare the exception

- A)In case you declare the exception, if exception does not occur, the code will be executed fine.
- B)In case you declare the exception if exception occurs, an exception will be thrown at runtime because throws does not handle the exception.

### A)Program if exception does not occur

```

import java.io.*;
class M{
    void method()throws IOException{
        System.out.println("device operation performed");
    }
}
class Testthrows3{
    public static void main(String args[])throws IOException{//declare exception
        M m=new M();
        m.method();

        System.out.println("normal flow...");
    }
}

```

**Test it Now**

Output:device operation performed  
normal flow...

### B)Program if exception occurs

```

import java.io.*;
class M{
    void method()throws IOException{
        throw new IOException("device error");
    }
}
class Testthrows4{
    public static void main(String args[])throws IOException{//declare exception
        M m=new M();
        m.method();

        System.out.println("normal flow...");
    }
}

```

}

### Test it Now

Output:Runtime Exception

## Difference between throw and throws

[Click me for details](#)

### Que) Can we rethrow an exception?

Yes, by throwing same exception in catch block.

## Difference between throw and throws in Java

There are many differences between throw and throws keywords. A list of differences between throw and throws are given below:

No. throw	throws
1) Java throw keyword is used to explicitly throw an exception.	Java throws keyword is used to declare an exception.
2) Checked exception cannot be propagated using throw only.	Checked exception can be propagated with throws.
3) Throw is followed by an instance.	Throws is followed by class.
4) Throw is used within the method.	Throws is used with the method signature.
5) You cannot throw multiple exceptions.	You can declare multiple exceptions e.g. public void method()throws IOException,SQLException.

## Java throw example

```
void m(){
    throw new ArithmeticException("sorry");
}
```

## Java throws example

```
void m()throws ArithmeticException{
    //method code
}
```

## Java throw and throws example

```
void m()throws ArithmeticException{
    throw new ArithmeticException("sorry");
}
```

## Difference between final, finally and finalize

There are many differences between final, finally and finalize. A list of differences between final, finally and finalize are given below:

No. final	finally	finalize
1) Final is used to apply restrictions on class, method and variable. Final class can't be inherited, final method can't be	Finally is used to place important code, it will be executed whether	Finalize is used to perform clean up processing just before

	overridden and final variable value can't be changed.	exception is handled or not.	object is garbage collected.
2)	Final is a keyword.	Finally is a block.	Finalize is a method.

## Java final example

```
class FinalExample{
public static void main(String[] args){
final int x=100;
x=200;//Compile Time Error
}}
```

## Java finally example

```
class FinallyExample{
public static void main(String[] args){
try{
int x=300;
}catch(Exception e){System.out.println(e);}
finally{System.out.println("finally block is executed");}
}}
```

## Java finalize example

```
class FinalizeExample{
public void finalize(){System.out.println("finalize called");}
public static void main(String[] args){
FinalizeExample f1=new FinalizeExample();
FinalizeExample f2=new FinalizeExample();
f1=null;
f2=null;
System.gc();
}}
```

## ExceptionHandling with MethodOverriding in Java

There are many rules if we talk about method overriding with exception handling. The Rules are as follows:

- **If the superclass method does not declare an exception**
  - If the superclass method does not declare an exception, subclass overridden method cannot declare the checked exception but it can declare unchecked exception.
- **If the superclass method declares an exception**
  - If the superclass method declares an exception, subclass overridden method can declare same, subclass exception or no exception but cannot declare parent exception.

### If the superclass method does not declare an exception

	<b>1) Rule: If the superclass method does not declare an exception, subclass overridden method cannot declare the checked exception.</b>
--	--

```
import java.io.*;
class Parent{
void msg(){System.out.println("parent");}
}

class TestExceptionChild extends Parent{
```

```

void msg()throws IOException{
    System.out.println("TestExceptionChild");
}
public static void main(String args[]){
    Parent p=new TestExceptionChild();
    p.msg();
}

```

**Test it Now**

Output:Compile Time Error

 **2) Rule:** If the superclass method does not declare an exception, subclass overridden method cannot declare the checked exception but can declare unchecked exception.

```

import java.io.*;
class Parent{
    void msg(){System.out.println("parent");}
}

class TestExceptionChild1 extends Parent{
    void msg()throws ArithmeticException{
        System.out.println("child");
    }
    public static void main(String args[]){
        Parent p=new TestExceptionChild1();
        p.msg();
    }
}

```

**Test it Now**

Output:child

If the superclass method declares an exception

 **1) Rule:** If the superclass method declares an exception, subclass overridden method can declare same, subclass exception or no exception but cannot declare parent exception.

Example in case subclass overridden method declares parent exception

```

import java.io.*;
class Parent{
    void msg()throws ArithmeticException{System.out.println("parent");}
}

class TestExceptionChild2 extends Parent{
    void msg()throws Exception{System.out.println("child");

    public static void main(String args[]){
        Parent p=new TestExceptionChild2();
        try{
            p.msg();
        }catch(Exception e){}
    }
}

```

**Test it Now**

Output:Compile Time Error

---

## Example in case subclass overridden method declares same exception

```
import java.io.*;
class Parent{
    void msg()throws Exception{System.out.println("parent");}
}

class TestExceptionChild3 extends Parent{
    void msg()throws Exception{System.out.println("child");}

    public static void main(String args[]){
        Parent p=new TestExceptionChild3();
        try{
            p.msg();
        }catch(Exception e){}
    }
}
```

**Test it Now**

Output:child

---

## Example in case subclass overridden method declares subclass exception

```
import java.io.*;
class Parent{
    void msg()throws Exception{System.out.println("parent");}
}

class TestExceptionChild4 extends Parent{
    void msg()throws ArithmeticException{System.out.println("child");}

    public static void main(String args[]){
        Parent p=new TestExceptionChild4();
        try{
            p.msg();
        }catch(Exception e){}
    }
}
```

**Test it Now**

Output:child

---

## Example in case subclass overridden method declares no exception

```
import java.io.*;
class Parent{
    void msg()throws Exception{System.out.println("parent");}
}

class TestExceptionChild5 extends Parent{
    void msg(){System.out.println("child");}

    public static void main(String args[]){
        Parent p=new TestExceptionChild5();
    }
}
```

```

try{
p.msg();
}catch(Exception e){}
}
}

```

**Test it Now**

Output:child

## Java Custom Exception

If you are creating your own Exception that is known as custom exception or user-defined exception. Java custom exceptions are used to customize the exception according to user need.

By the help of custom exception, you can have your own exception and message.

Let's see a simple example of java custom exception.

```

class InvalidAgeException extends Exception{
    InvalidAgeException(String s){
        super(s);
    }
}

class TestCustomException1{

    static void validate(int age)throws InvalidAgeException{
        if(age<18)
            throw new InvalidAgeException("not valid");
        else
            System.out.println("welcome to vote");
    }

    public static void main(String args[]){
        try{
            validate(13);
        }catch(Exception m){System.out.println("Exception occured: "+m);}

        System.out.println("rest of the code...");
    }
}

```

**Test it Now**

Output:Exception occured: InvalidAgeException:not valid  
rest of the code...

## Java Inner Class

**Java inner class** or nested class is a class i.e. declared inside the class or interface.

We use inner classes to logically group classes and interfaces in one place so that it can be more readable and maintainable.

Additionally, it can access all the members of outer class including private data members and methods.

### Syntax of Inner class

```
class Java_Outer_class{  
    //code  
    class Java_Inner_class{  
        //code  
    }  
}
```

## Advantage of java inner classes

There are basically three advantages of inner classes in java. They are as follows:

- 1) Nested classes represent a special type of relationship that is **it can access all the members (data members and methods) of outer class** including private.
- 2) Nested classes are used **to develop more readable and maintainable code** because it logically group classes and interfaces in one place only.
- 3) **Code Optimization:** It requires less code to write.

## Difference between nested class and inner class in Java

Inner class is a part of nested class. Non-static nested classes are known as inner classes.

---

## Types of Nested classes

There are two types of nested classes non-static and static nested classes. The non-static nested classes are also known as inner classes.

1. Non-static nested class(inner class)
  - a)Member inner class
  - b)Anonymous inner class
  - c)Local inner class
2. Static nested class

## Java Member inner class

A non-static class that is created inside a class but outside a method is called member inner class.

Syntax:

```
class Outer{  
    //code  
    class Inner{  
        //code  
    }  
}
```

## Java Member inner class example

In this example, we are creating msg() method in member inner class that is accessing the private data member of outer class.

```
class TestMemberOuter1{  
    private int data=30;  
    class Inner{  
        void msg(){System.out.println("data is "+data);}  
    }  
    public static void main(String args[]){  
        TestMemberOuter1 obj=new TestMemberOuter1();  
        TestMemberOuter1.Inner in=obj.new Inner();  
        in.msg();  
    }  
}
```

**Test it Now**

Output:

```
data is 30
```

## Internal working of Java member inner class

The java compiler creates two class files in case of inner class. The class file name of inner class is "Outer\$Inner". If you want to instantiate inner class, you must have to create the instance of outer class. In such case, instance of inner class is created inside the instance of outer class.

## Internal code generated by the compiler

The java compiler creates a class file named Outer\$Inner in this case. The Member inner class have the reference of Outer class that is why it can access all the data members of Outer class including private.

```
import java.io.PrintStream;
class Outer$Inner
{
    final Outer this$0;
    Outer$Inner()
    {
        super();
        this$0 = Outer.this;
    }
    void msg()
    {
        System.out.println((new StringBuilder()).append("data is ")
            .append(Outer.access$000(Outer.this)).toString());
    }
}
```

## Java Anonymous inner class

A class that have no name is known as anonymous inner class in java. It should be used if you have to override method of class or interface. Java Anonymous inner class can be created by two ways:

1. Class (may be abstract or concrete).
2. Interface

### Java anonymous inner class example using class

```
abstract class Person{
    abstract void eat();
}

class TestAnonymousInner{
    public static void main(String args[]){
        Person p=new Person(){
            void eat(){System.out.println("nice fruits");}
        };
        p.eat();
    }
}
```

**Test it Now**

Output:

```
nice fruits
```

## Internal working of given code

```
Person p=new Person(){  
void eat(){System.out.println("nice fruits");}  
};
```

1. A class is created but its name is decided by the compiler which extends the Person class and provides the implementation of the eat() method.
2. An object of Anonymous class is created that is referred by p reference variable of Person type.

## Internal class generated by the compiler

```
import java.io.PrintStream;  
static class TestAnonymousInner$1 extends Person  
{  
    TestAnonymousInner$1(){}
    void eat()
    {
        System.out.println("nice fruits");
    }
}
```

## Java anonymous inner class example using interface

```
interface Eatable{
    void eat();
}

class TestAnonymousInner1{
    public static void main(String args[]){
        Eatable e=new Eatable(){
            public void eat(){System.out.println("nice fruits");}
        };
        e.eat();
    }
}
```

**Test it Now**

Output:

```
nice fruits
```

## Internal working of given code

It performs two main tasks behind this code:

```
Eatable p=new Eatable(){  
void eat(){System.out.println("nice fruits");}  
};
```

1. A class is created but its name is decided by the compiler which implements the Eatable interface and provides the implementation of the eat() method.
2. An object of Anonymous class is created that is referred by p reference variable of Eatable type.

## Internal class generated by the compiler

```
import java.io.PrintStream;  
static class TestAnonymousInner1$1 implements Eatable  
{  
    TestAnonymousInner1$1(){}
    void eat(){System.out.println("nice fruits");}
}
```

## Java Local inner class

A class i.e. created inside a method is called local inner class in java. If you want to invoke the methods of local inner class, you must instantiate this class inside the method.

### Java local inner class example

```
public class localInner1{  
    private int data=30;//instance variable  
    void display(){  
        class Local{  
            void msg(){System.out.println(data);}  
        }  
        Local l=new Local();  
        l.msg();  
    }  
    public static void main(String args[]){  
        localInner1 obj=new localInner1();  
        obj.display();  
    }  
}
```

**Test it Now**

Output:

30

### Internal class generated by the compiler

In such case, compiler creates a class named Simple\$1Local that have the reference of the outer class.

```
import java.io.PrintStream;  
class localInner1$Local  
{  
    final localInner1 this$0;  
    localInner1$Local()  
    {  
        super();  
        this$0 = Simple.this;  
    }  
    void msg()  
    {  
        System.out.println(localInner1.access$000(localInner1.this));  
    }  
}
```



**Rule: Local variable can't be private, public or protected.**

### Rules for Java Local Inner class



**1) Local inner class cannot be invoked from outside the method.**



**2) Local inner class cannot access non-final local variable till JDK 1.7. Since JDK 1.8, it is possible to access the non-final local variable in local inner class.**

Example of local inner class with local variable

```

class localInner2{
    private int data=30;//instance variable
    void display(){
        int value=50;//local variable must be final till jdk 1.7 only
        class Local{
            void msg(){System.out.println(value);}
        }
        Local l=new Local();
        l.msg();
    }
    public static void main(String args[]){
        localInner2 obj=new localInner2();
        obj.display();
    }
}

```

**Test it Now**

Output:

50

## Java static nested class

A static class i.e. created inside a class is called static nested class in java. It cannot access non-static data members and methods. It can be accessed by outer class name.

- It can access static data members of outer class including private.
- Static nested class cannot access non-static (instance) data member or method.

## Java static nested class example with instance method

```

class TestOuter1{
    static int data=30;
    static class Inner{
        void msg(){System.out.println("data is "+data);}
    }
    public static void main(String args[]){
        TestOuter1.Inner obj=new TestOuter1.Inner();
        obj.msg();
    }
}

```

**Test it Now**

Output:

data is 30

In this example, you need to create the instance of static nested class because it has instance method msg(). But you don't need to create the object of Outer class because nested class is static and static properties, methods or classes can be accessed without object.

## Internal class generated by the compiler

```

import java.io.PrintStream;
static class TestOuter1$Inner
{
    TestOuter1$Inner(){}
    void msg(){

```

```
System.out.println((new StringBuilder()).append("data is ")
.append(TestOuter1.data).toString());
}
}
```

## Java static nested class example with static method

If you have the static member inside static nested class, you don't need to create instance of static nested class.

```
class TestOuter2{
    static int data=30;
    static class Inner{
        static void msg(){System.out.println("data is "+data);}
    }
    public static void main(String args[]){
        TestOuter2.Inner.msg(); //no need to create the instance of static nested class
    }
}
```

**Test it Now**

Output:

```
data is 30
```

## Java Nested Interface

An interface i.e. declared within another interface or class is known as nested interface. The nested interfaces are used to group related interfaces so that they can be easy to maintain. The nested interface must be referred by the outer interface or class. It can't be accessed directly.

### Points to remember for nested interfaces

There are given some points that should be remembered by the java programmer.

- Nested interface must be public if it is declared inside the interface but it can have any access modifier if declared within the class.
- Nested interfaces are declared static implicitly.

### Syntax of nested interface which is declared within the interface

```
interface interface_name{
    ...
    interface nested_interface_name{
        ...
    }
}
```

### Syntax of nested interface which is declared within the class

```
class class_name{
    ...
    interface nested_interface_name{
        ...
    }
}
```

---

### Example of nested interface which is declared within the interface

In this example, we are going to learn how to declare the nested interface and how we can access it.

```
interface Showable{
    void show();
    interface Message{
        void msg();
    }
}

class TestNestedInterface1 implements Showable.Message{
    public void msg(){System.out.println("Hello nested interface");}

    public static void main(String args[]){
        Showable.Message message=new TestNestedInterface1(); //upcasting here
        message.msg();
    }
}
```

#### Test it Now

[download the example of nested interface](#)

Output:hello nested interface

As you can see in the above example, we are accessing the Message interface by its outer interface Showable because it cannot be accessed directly. It is just like almirah inside the room, we cannot access the almirah directly because we must enter the room first. In collection framework, sun microsystem has provided a nested interface Entry. Entry is the subinterface of Map i.e. accessed by Map.Entry.

### Internal code generated by the java compiler for nested interface Message

The java compiler internally creates public and static interface as displayed below:.

```
public static interface Showable$Message
{
    public abstract void msg();
}
```

### Example of nested interface which is declared within the class

Let's see how can we define an interface inside the class and how can we access it.

```
class A{
    interface Message{
        void msg();
    }
}

class TestNestedInterface2 implements A.Message{
    public void msg(){System.out.println("Hello nested interface");}

    public static void main(String args[]){
        A.Message message=new TestNestedInterface2(); //upcasting here
        message.msg();
    }
}
```

## Test it Now

```
Output:hello nested interface
```

## Can we define a class inside the interface?

Yes, If we define a class inside the interface, java compiler creates a static nested class. Let's see how can we define a class within the interface:

```
interface M{  
    class A{}  
}
```

## Multithreading in Java

**Multithreading in java** is a process of executing multiple threads simultaneously.

Thread is basically a lightweight sub-process, a smallest unit of processing. Multiprocessing and multithreading, both are used to achieve multitasking.

But we use multithreading than multiprocessing because threads share a common memory area. They don't allocate separate memory area so saves memory, and context-switching between the threads takes less time than process.

Java Multithreading is mostly used in games, animation etc.

### Advantage of Java Multithreading

- 1) It **doesn't block the user** because threads are independent and you can perform multiple operations at same time.
- 2) You **can perform many operations together so it saves time**.
- 3) Threads are **independent** so it doesn't affect other threads if exception occur in a single thread.

### Multitasking

Multitasking is a process of executing multiple tasks simultaneously. We use multitasking to utilize the CPU. Multitasking can be achieved by two ways:

- Process-based Multitasking(Multiprocessing)
- Thread-based Multitasking(Multithreading)

### 1) Process-based Multitasking (Multiprocessing)

- Each process have its own address in memory i.e. each process allocates separate memory area.
- Process is heavyweight.
- Cost of communication between the process is high.
- Switching from one process to another require some time for saving and loading registers, memory maps, updating lists etc.

### 2) Thread-based Multitasking (Multithreading)

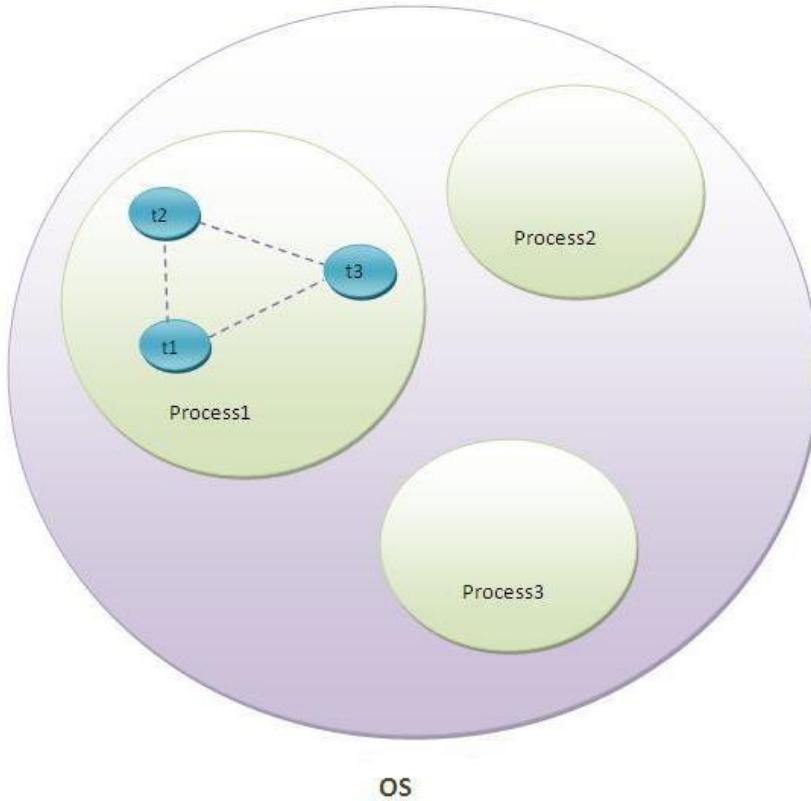
- Threads share the same address space.
- Thread is lightweight.
- Cost of communication between the thread is low.

 **Note:** At least one process is required for each thread.

## What is Thread in java

A thread is a lightweight sub process, a smallest unit of processing. It is a separate path of execution.

Threads are independent, if there occurs exception in one thread, it doesn't affect other threads. It shares a common memory area.



As shown in the above figure, 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.



**Note:** At a time one thread is executed only.

## Life cycle of a Thread (Thread States)

A thread can be in one of the five states. According to sun, there is only 4 states in **thread life cycle in java** new, runnable, non-runnable and terminated. There is no running state.

But for better understanding the threads, we are explaining it in the 5 states.

The life cycle of the thread in java is controlled by JVM. The java thread states are as follows:

## How to create thread

There are two ways to create a thread:

1. By extending Thread class
2. By implementing Runnable interface.

### Thread class:

Thread class provide 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:

1. **public void run():** is used to perform action for a thread.
2. **public void start():** starts the execution of the thread. JVM calls the run() method on the thread.
3. **public void sleep(long milliseconds):** Causes the currently executing thread to sleep (temporarily cease execution) for the specified number of milliseconds.
4. **public void join():** waits for a thread to die.
5. **public void join(long milliseconds):** waits for a thread to die for the specified milliseconds.
6. **public int getPriority():** returns the priority of the thread.
7. **public int setPriority(int priority):** changes the priority of the thread.
8. **public String getName():** returns the name of the thread.
9. **public void setName(String name):** changes the name of the thread.
10. **public Thread currentThread():** returns the reference of currently executing thread.
11. **public int getId():** returns the id of the thread.
12. **public Thread.State getState():** returns the state of the thread.
13. **public boolean isAlive():** tests if the thread is alive.
14. **public void yield():** causes the currently executing thread object to temporarily pause and allow other threads to execute.
15. **public void suspend():** is used to suspend the thread(deprecated).
16. **public void resume():** is used to resume the suspended thread(deprecated).
17. **public void stop():** is used to stop the thread(deprecated).
18. **public boolean isDaemon():** tests if the thread is a daemon thread.
19. **public void setDaemon(boolean b):** marks the thread as daemon or user thread.
20. **public void interrupt():** interrupts the thread.
21. **public boolean isInterrupted():** tests if the thread has been interrupted.
22. **public static boolean interrupted():** tests if the current thread has been interrupted.

## Runnable interface:

The Runnable interface should be implemented by any class whose instances are intended to be executed by a thread. Runnable interface have only one method named run().

1. **public void run():** is used to perform action for a thread.

## Starting a thread:

**start() method** of Thread class is used to start a newly created thread. It performs following tasks:

- A new thread starts(with new callstack).
- The thread moves from New state to the Runnable state.
- When the thread gets a chance to execute, its target run() method will run.

## 1)By extending Thread class:

```
class Multi extends Thread{
public void run(){
System.out.println("thread is running...");
}
public static void main(String args[]){
Multi t1=new Multi();
t1.start();
}
}

Output:thread is running...
```

## Who makes your class object as thread object?

**Thread class constructor** allocates a new thread object. When you create object of Multi class, your class constructor is invoked (provided by Compiler) from where Thread class constructor is invoked (by super() as first statement). So your Multi class object is thread object now.

---

## 2) By implementing the Runnable interface:

```
class Multi3 implements Runnable{  
    public void run(){  
        System.out.println("thread is running...");  
    }  
  
    public static void main(String args[]){  
        Multi3 m1=new Multi3();  
        Thread t1 =new Thread(m1);  
        t1.start();  
    }  
}
```

Output: thread is running...

If you are not extending the Thread class, your class object would not be treated as a thread object. So you need to explicitly create Thread class object. We are passing the object of your class that implements Runnable so that your class run() method may execute.

## Thread Scheduler in Java

**Thread scheduler** in java is the part of the JVM that decides which thread should run.

There is no guarantee that which runnable thread will be chosen to run by the thread scheduler.

Only one thread at a time can run in a single process.

The thread scheduler mainly uses preemptive or time slicing scheduling to schedule the threads.

## Difference between preemptive scheduling and time slicing

Under preemptive scheduling, the highest priority task executes until it enters the waiting or dead states or a higher priority task comes into existence. Under time slicing, a task executes for a predefined slice of time and then reenters the pool of ready tasks. The scheduler then determines which task should execute next, based on priority and other factors.

## Sleep method in java

The sleep() method of Thread class is used to sleep a thread for the specified amount of time.

### Syntax of sleep() method in java

The Thread class provides two methods for sleeping a thread:

- public static void sleep(long milliseconds) throws InterruptedException
- public static void sleep(long milliseconds, int nanos) throws InterruptedException

### Example of sleep method in java

```

class TestSleepMethod1 extends Thread{
    public void run(){
        for(int i=1;i<5;i++){
            try{Thread.sleep(500);}catch(InterruptedException e){System.out.println(e);}
            System.out.println(i);
        }
    }

    public static void main(String args[]){
        TestSleepMethod1 t1=new TestSleepMethod1();
        TestSleepMethod1 t2=new TestSleepMethod1();

        t1.start();
        t2.start();
    }
}

```

Output:

```

1
1
2
2
3
3
4
4

```

As you know well that at a time only one thread is executed. If you sleep a thread for the specified time, the thread scheduler picks up another thread and so on.

## Can we start a thread twice

No. After starting a thread, it can never be started again. If you do so, an *IllegalThreadStateException* is thrown. In such case, thread will run once but for second time, it will throw exception.

Let's understand it by the example given below:

```

public class TestThreadTwice1 extends Thread{
    public void run(){
        System.out.println("running...");
    }

    public static void main(String args[]){
        TestThreadTwice1 t1=new TestThreadTwice1();
        t1.start();
        t1.start();
    }
}

```

**Test it Now**

```

running
Exception in thread "main" java.lang.IllegalThreadStateException

```

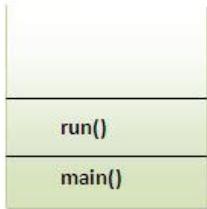
## What if we call run() method directly instead start() method?

- Each thread starts in a separate call stack.
- Invoking the run() method from main thread, the run() method goes onto the current call stack rather than at the beginning of a new call stack.

```
class TestCallRun1 extends Thread{
    public void run(){
        System.out.println("running...");
    }
    public static void main(String args[]){
        TestCallRun1 t1=new TestCallRun1();
        t1.run(); //fine, but does not start a separate call stack
    }
}
```

**Test it Now**

Output:running...



Stack  
(main thread)

**Problem if you direct call run() method**

```
class TestCallRun2 extends Thread{
    public void run(){
        for(int i=1;i<5;i++){
            try{Thread.sleep(500);}catch(InterruptedException e){System.out.println(e);}
            System.out.println(i);
        }
    }
    public static void main(String args[]){
        TestCallRun2 t1=new TestCallRun2();
        TestCallRun2 t2=new TestCallRun2();

        t1.run();
        t2.run();
    }
}
```

**Test it Now**

Output:1

2  
3  
4  
5  
1  
2  
3  
4  
5

As you can see in the above program that there is no context-switching because here t1 and t2 will be treated as normal object not thread

object.

## The join() method:

The join() method waits for a thread to die. In other words, it causes the currently running threads to stop executing until the thread it joins with completes its task.

### Syntax:

```
public void join()throws InterruptedException  
public void join(long milliseconds)throws InterruptedException
```

#### **Example of join() method**

```
class TestJoinMethod1 extends Thread{  
    public void run(){  
        for(int i=1;i<=5;i++){  
            try{  
                Thread.sleep(500);  
            }catch(Exception e){System.out.println(e);}  
            System.out.println(i);  
        }  
    }  
    public static void main(String args[]){  
        TestJoinMethod1 t1=new TestJoinMethod1();  
        TestJoinMethod1 t2=new TestJoinMethod1();  
        TestJoinMethod1 t3=new TestJoinMethod1();  
        t1.start();  
        try{  
            t1.join();  
        }catch(Exception e){System.out.println(e);}  
  
        t2.start();  
        t3.start();  
    }  
}
```

**Test it Now**

Output:

```
1  
2  
3  
4  
5  
1  
1  
2  
2  
3  
3  
4  
4  
5
```

As you can see in the above example, when t1 completes its task then t2 and t3 starts executing.

#### **Example of join(long milliseconds) method**

```
class TestJoinMethod2 extends Thread{  
    public void run(){  
        for(int i=1;i<=5;i++){  
            try{  
                Thread.sleep(500);  
            }catch(Exception e){System.out.println(e);}  
            System.out.println(i);  
        }  
    }  
  
    public static void main(String args[]){  
        TestJoinMethod2 t1=new TestJoinMethod2();  
        TestJoinMethod2 t2=new TestJoinMethod2();  
        TestJoinMethod2 t3=new TestJoinMethod2();  
        t1.start();  
        try{  
            t1.join(1500);  
        }catch(Exception e){System.out.println(e);}  
  
        t2.start();  
        t3.start();  
    }  
}
```

**Test it Now**

Output:  
1  
2  
3  
1  
4  
1  
2  
5  
2  
3  
3  
4  
4  
5  
5

In the above example, when t1 is completes its task for 1500 milliseconds(3 times) then t2 and t3 starts executing.

#### **getName(), setName(String) and getId() method:**

```
public String getName()  
public void setName(String name)  
public long getId()  
  
class TestJoinMethod3 extends Thread{  
    public void run(){  
        System.out.println("running...");  
    }  
    public static void main(String args[]){  
        TestJoinMethod3 t1=new TestJoinMethod3();
```

```

TestJoinMethod3 t2=new TestJoinMethod3();
System.out.println("Name of t1:"+t1.getName());
System.out.println("Name of t2:"+t2.getName());
System.out.println("id of t1:"+t1.getId());

t1.start();
t2.start();

t1.setName("Sonoo Jaiswal");
System.out.println("After changing name of t1:"+t1.getName());
}
}

```

**Test it Now**

```

Output:Name of t1:Thread-0
      Name of t2:Thread-1
      id of t1:8
      running...
      After changling name of t1:Sonoo Jaiswal
      running...

```

## The currentThread() method:

The `currentThread()` method returns a reference to the currently executing thread object.

### Syntax:

```
public static Thread currentThread()
```

#### **Example of `currentThread()` method**

```

class TestJoinMethod4 extends Thread{
    public void run(){
        System.out.println(Thread.currentThread().getName());
    }
}

public static void main(String args[]){
    TestJoinMethod4 t1=new TestJoinMethod4();
    TestJoinMethod4 t2=new TestJoinMethod4();

    t1.start();
    t2.start();
}
}

```

**Test it Now**

```

Output:Thread-0
      Thread-1

```

## Naming a thread:

The `Thread` class provides methods to change and get the name of a thread.

1. **public String getName():** is used to return the name of a thread.
2. **public void setName(String name):** is used to change the name of a thread.

## Example of naming a thread:

```
class TestMultiNaming1 extends Thread{
    public void run(){
        System.out.println("running...");
    }
    public static void main(String args[]){
        TestMultiNaming1 t1=new TestMultiNaming1();
        TestMultiNaming1 t2=new TestMultiNaming1();
        System.out.println("Name of t1:"+t1.getName());
        System.out.println("Name of t2:"+t2.getName());

        t1.start();
        t2.start();

        t1.setName("Sonoo Jaiswal");
        System.out.println("After changing name of t1:"+t1.getName());
    }
}
```

**Test it Now**

```
Output:Name of t1:Thread-0
      Name of t2:Thread-1
      id of t1:8
      running...
      After changeling name of t1:Sonoo Jaiswal
      running...
```

## The currentThread() method:

The currentThread() method returns a reference to the currently executing thread object.

### Syntax of currentThread() method:

- **public static Thread currentThread():** returns the reference of currently running thread.
- 

## Example of currentThread() method:

```
class TestMultiNaming2 extends Thread{
    public void run(){
        System.out.println(Thread.currentThread().getName());
    }
}

public static void main(String args[]){
    TestMultiNaming2 t1=new TestMultiNaming2();
    TestMultiNaming2 t2=new TestMultiNaming2();

    t1.start();
    t2.start();
}
```

**Test it Now**

```
Output:Thread-0
      Thread-1
```

## Priority of a Thread (Thread Priority):

Each thread have a priority. Priorities are represented by a number between 1 and 10. In most cases, thread scheduler schedules the threads according to their priority (known as preemptive scheduling). But it is not guaranteed because it depends on JVM specification that which scheduling it chooses.

### 3 constants defiend in Thread class:

1. public static int MIN\_PRIORITY
2. public static int NORM\_PRIORITY
3. public static int MAX\_PRIORITY

Default priority of a thread is 5 (NORM\_PRIORITY). The value of MIN\_PRIORITY is 1 and the value of MAX\_PRIORITY is 10.

### Example of priority of a Thread:

```
class TestMultiPriority1 extends Thread{
    public void run(){
        System.out.println("running thread name is:"+Thread.currentThread().getName());
        System.out.println("running thread priority is:"+Thread.currentThread().getPriority());
    }
    public static void main(String args[]){
        TestMultiPriority1 m1=new TestMultiPriority1();
        TestMultiPriority1 m2=new TestMultiPriority1();
        m1.setPriority(Thread.MIN_PRIORITY);
        m2.setPriority(Thread.MAX_PRIORITY);
        m1.start();
        m2.start();
    }
}
```

**Test it Now**

```
Output:running thread name is:Thread-0
      running thread priority is:10
      running thread name is:Thread-1
      running thread priority is:1
```

## Daemon Thread in Java

**Daemon thread in java** is a service provider thread that provides services to the user thread. Its life depend on the mercy of user threads

i.e. when all the user threads dies, JVM terminates this thread automatically.

There are many java daemon threads running automatically e.g. gc, finalizer etc.

You can see all the detail by typing the jconsole in the command prompt. The jconsole tool provides information about the loaded classes, memory usage, running threads etc.

## Points to remember for Daemon Thread in Java

- It provides services to user threads for background supporting tasks. It has no role in life than to serve user threads.
- Its life depends on user threads.
- It is a low priority thread.

### Why JVM terminates the daemon thread if there is no user thread?

The sole purpose of the daemon thread is that it provides services to user thread for background supporting task. If there is no user thread, why should JVM keep running this thread. That is why JVM terminates the daemon thread if there is no user thread.

### Methods for Java Daemon thread by Thread class

The `java.lang.Thread` class provides two methods for java daemon thread.

No.	Method	Description
1)	<code>public void setDaemon(boolean status)</code>	is used to mark the current thread as daemon thread or user thread.
2)	<code>public boolean isDaemon()</code>	is used to check that current is daemon.

### Simple example of Daemon thread in java

File: `MyThread.java`

```
public class TestDaemonThread1 extends Thread{
    public void run(){
        if(Thread.currentThread().isDaemon()){//checking for daemon thread
            System.out.println("daemon thread work");
        }
        else{
            System.out.println("user thread work");
        }
    }

    public static void main(String[] args){
        TestDaemonThread1 t1=new TestDaemonThread1(); //creating thread
        TestDaemonThread1 t2=new TestDaemonThread1();
        TestDaemonThread1 t3=new TestDaemonThread1();

        t1.setDaemon(true); //now t1 is daemon thread

        t1.start(); //starting threads
        t2.start();
        t3.start();
    }
}
```

**Test it Now**

### Output

```
daemon thread work
user thread work
user thread work
```



**Note:** If you want to make a user thread as Daemon, it must not be started otherwise it will throw `IllegalThreadStateException`.

File: MyThread.java

```
class TestDaemonThread2 extends Thread{  
    public void run(){  
        System.out.println("Name: "+Thread.currentThread().getName());  
        System.out.println("Daemon: "+Thread.currentThread().isDaemon());  
    }  
  
    public static void main(String[] args){  
        TestDaemonThread2 t1=new TestDaemonThread2();  
        TestDaemonThread2 t2=new TestDaemonThread2();  
        t1.start();  
        t1.setDaemon(true); //will throw exception here  
        t2.start();  
    }  
}
```

#### Test it Now

Output:exception in thread main: java.lang.IllegalThreadStateException

## Java Thread Pool

**Java Thread pool** represents a group of worker threads that are waiting for the job and reuse many times.

In case of thread pool, a group of fixed size threads are created. A thread from the thread pool is pulled out and assigned a job by the service provider. After completion of the job, thread is contained in the thread pool again.

### Advantage of Java Thread Pool

**Better performance** It saves time because there is no need to create new thread.

### Real time usage

It is used in Servlet and JSP where container creates a thread pool to process the request.

## Example of Java Thread Pool

Let's see a simple example of java thread pool using ExecutorService and Executors.

File: WorkerThread.java

```
import java.util.concurrent.ExecutorService;  
import java.util.concurrent.Executors;  
class WorkerThread implements Runnable {  
    private String message;  
    public WorkerThread(String s){  
        this.message=s;  
    }  
    public void run() {  
        System.out.println(Thread.currentThread().getName()+" (Start) message = "+message);  
        processmessage(); //call processmessage method that sleeps the thread for 2 seconds  
    }  
}
```

```

        System.out.println(Thread.currentThread().getName()+" (End)");//prints thread name
    }
    private void processmessage() {
        try { Thread.sleep(2000); } catch (InterruptedException e) { e.printStackTrace(); }
    }
}

```

*File: JavaThreadPoolExample.java*

```

public class TestThreadPool {
    public static void main(String[] args) {
        ExecutorService executor = Executors.newFixedThreadPool(5);//creating a pool of 5 threads
        for (int i = 0; i < 10; i++) {
            Runnable worker = new WorkerThread(" " + i);
            executor.execute(worker);//calling execute method of ExecutorService
        }
        executor.shutdown();
        while (!executor.isTerminated()) { }

        System.out.println("Finished all threads");
    }
}

```

**Test it Now**

[download this example](#)

Output:

```

pool-1-thread-1 (Start) message = 0
pool-1-thread-2 (Start) message = 1
pool-1-thread-3 (Start) message = 2
pool-1-thread-5 (Start) message = 4
pool-1-thread-4 (Start) message = 3
pool-1-thread-2 (End)
pool-1-thread-2 (Start) message = 5
pool-1-thread-1 (End)
pool-1-thread-1 (Start) message = 6
pool-1-thread-3 (End)
pool-1-thread-3 (Start) message = 7
pool-1-thread-4 (End)
pool-1-thread-4 (Start) message = 8
pool-1-thread-5 (End)
pool-1-thread-5 (Start) message = 9
pool-1-thread-2 (End)
pool-1-thread-1 (End)
pool-1-thread-4 (End)
pool-1-thread-3 (End)
pool-1-thread-5 (End)
finished all threads

```

## ThreadGroup in Java

Java provides a convenient way to group multiple threads in a single object. In such way, we can suspend, resume or interrupt group of threads by a single method call.



**Note: Now suspend(), resume() and stop() methods are deprecated.**

Java thread group is implemented by `java.lang.ThreadGroup` class.

## Constructors of ThreadGroup class

There are only two constructors of ThreadGroup class.

No.	Constructor	Description
1)	<code>ThreadGroup(String name)</code>	creates a thread group with given name.
2)	<code>ThreadGroup(ThreadGroup parent, String name)</code>	creates a thread group with given parent group and name.

## Important methods of ThreadGroup class

There are many methods in ThreadGroup class. A list of important methods are given below.

No.	Method	Description
1)	<code>int activeCount()</code>	returns no. of threads running in current group.
2)	<code>int activeGroupCount()</code>	returns a no. of active group in this thread group.
3)	<code>void destroy()</code>	destroys this thread group and all its sub groups.
4)	<code>String getName()</code>	returns the name of this group.
5)	<code>ThreadGroup getParent()</code>	returns the parent of this group.
6)	<code>void interrupt()</code>	interrupts all threads of this group.
7)	<code>void list()</code>	prints information of this group to standard console.

Let's see a code to group multiple threads.

```
ThreadGroup tg1 = new ThreadGroup("Group A");
Thread t1 = new Thread(tg1,new MyRunnable(),"one");
Thread t2 = new Thread(tg1,new MyRunnable(),"two");
Thread t3 = new Thread(tg1,new MyRunnable(),"three");
```

Now all 3 threads belong to one group. Here, tg1 is the thread group name, MyRunnable is the class that implements Runnable interface and "one", "two" and "three" are the thread names.

Now we can interrupt all threads by a single line of code only.

```
Thread.currentThread().getThreadGroup().interrupt();
```

## ThreadGroup Example

File: `ThreadGroupDemo.java`

```
public class ThreadGroupDemo implements Runnable{
    public void run() {
        System.out.println(Thread.currentThread().getName());
    }
    public static void main(String[] args) {
        ThreadGroupDemo runnable = new ThreadGroupDemo();
        ThreadGroup tg1 = new ThreadGroup("Parent ThreadGroup");

        Thread t1 = new Thread(tg1, runnable,"one");
        t1.start();
        Thread t2 = new Thread(tg1, runnable,"two");
        t2.start();
        Thread t3 = new Thread(tg1, runnable,"three");
        t3.start();

        System.out.println("Thread Group Name: "+tg1.getName());
        tg1.list();
    }
}
```

```
    }  
}
```

Output:

```
one  
two  
three  
Thread Group Name: Parent ThreadGroup  
java.lang.ThreadGroup[name=Parent ThreadGroup,maxpri=10]  
    Thread[one,5,Parent ThreadGroup]  
    Thread[two,5,Parent ThreadGroup]  
    Thread[three,5,Parent ThreadGroup]
```

## Shutdown Hook

The shutdown hook can be used to perform cleanup resource or save the state when JVM shuts down normally or abruptly. Performing clean resource means closing log file, sending some alerts or something else. So if you want to execute some code before JVM shuts down, use shutdown hook.

### When does the JVM shut down?

The JVM shuts down when:

- user presses **ctrl+c** on the command prompt
- **System.exit(int)** method is invoked
- user logoff
- user shutdown etc.

---

### The **addShutdownHook(Runnable r)** method

The **addShutdownHook()** method of **Runtime** class is used to register the thread with the Virtual Machine. Syntax:

```
public void addShutdownHook(Runnable r){}
```

The object of **Runtime** class can be obtained by calling the static factory method **getRuntime()**. For example:

```
Runtime r = Runtime.getRuntime();
```

### Factory method

The method that returns the instance of a class is known as factory method.

---

## Simple example of Shutdown Hook

```
class MyThread extends Thread{  
    public void run(){  
        System.out.println("shut down hook task completed..");  
    }  
}  
  
public class TestShutdown1{  
    public static void main(String[] args) throws Exception {
```

```

Runtime r=Runtime.getRuntime();
r.addShutdownHook(new MyThread());

System.out.println("Now main sleeping... press ctrl+c to exit");
try{Thread.sleep(3000);}catch (Exception e) {}
}
}

```

#### Test it Now

Output:Now main sleeping... press ctrl+c to exit  
shut down hook task completed..

 **Note:** The shutdown sequence can be stopped by invoking the halt(int) method of Runtime class.

### Same example of Shutdown Hook by anonymous class:

```

public class TestShutdown2{
public static void main(String[] args)throws Exception {

Runtime r=Runtime.getRuntime();

r.addShutdownHook(new Runnable(){
public void run(){
System.out.println("shut down hook task completed..");
}
}
);

System.out.println("Now main sleeping... press ctrl+c to exit");
try{Thread.sleep(3000);}catch (Exception e) {}
}
}

```

#### Test it Now

Output:Now main sleeping... press ctrl+c to exit  
shut down hook task completed..

## How to perform single task by multiple threads?

If you have to perform single task by many threads, have only one run() method. For example:

#### **Program of performing single task by multiple threads**

```

class TestMultitasking1 extends Thread{
public void run(){
System.out.println("task one");
}
public static void main(String args[]){
TestMultitasking1 t1=new TestMultitasking1();
TestMultitasking1 t2=new TestMultitasking1();
TestMultitasking1 t3=new TestMultitasking1();
}
}

```

```
t1.start();
t2.start();
t3.start();
}
}
```

**Test it Now**

```
Output:task one
      task one
      task one
```

**Program of performing single task by multiple threads**

```
class TestMultitasking2 implements Runnable{
public void run(){
System.out.println("task one");
}

public static void main(String args[]){
Thread t1 =new Thread(new TestMultitasking2());//passing anonymous object of TestMultitasking2 class
Thread t2 =new Thread(new TestMultitasking2());

t1.start();
t2.start();

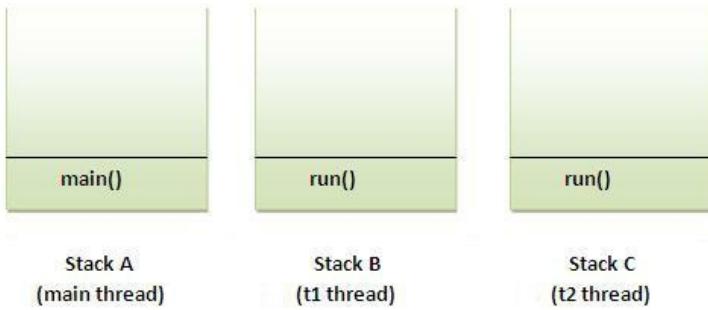
}
}
```

**Test it Now**

```
Output:task one
      task one
```

---

**Note: Each thread run in a separate callstack.**



---

**How to perform multiple tasks by multiple threads (multitasking in multithreading)?**

If you have to perform multiple tasks by multiple threads, have multiple run() methods. For example:

**Program of performing two tasks by two threads**

```
class Simple1 extends Thread{
public void run(){
```

```

        System.out.println("task one");
    }

}

class Simple2 extends Thread{
    public void run(){
        System.out.println("task two");
    }
}

class TestMultitasking3{
    public static void main(String args[]){
        Simple1 t1=new Simple1();
        Simple2 t2=new Simple2();

        t1.start();
        t2.start();
    }
}

```

**Test it Now**

Output:task one  
task two

Same example as above by anonymous class that extends Thread class:

**Program of performing two tasks by two threads**

```

class TestMultitasking4{
    public static void main(String args[]){
        Thread t1=new Thread(){
            public void run(){
                System.out.println("task one");
            }
        };
        Thread t2=new Thread(){
            public void run(){
                System.out.println("task two");
            }
        };

        t1.start();
        t2.start();
    }
}

```

**Test it Now**

Output:task one  
task two

Same example as above by anonymous class that implements Runnable interface:

**Program of performing two tasks by two threads**

```

class TestMultitasking5{
    public static void main(String args[]){
        Runnable r1=new Runnable(){
            public void run(){

```

```

        System.out.println("task one");
    }
};

Runnable r2=new Runnable(){
    public void run(){
        System.out.println("task two");
    }
};

Thread t1=new Thread(r1);
Thread t2=new Thread(r2);

t1.start();
t2.start();
}
}

```

**Test it Now**

Output:task one  
task two

## Java Garbage Collection

In java, garbage means unreferenced objects.

Garbage Collection is process of reclaiming the runtime unused memory automatically. In other words, it is a way to destroy the unused objects.

To do so, we were using free() function in C language and delete() in C++. But, in java it is performed automatically. So, java provides better memory management.

### Advantage of Garbage Collection

- It makes java **memory efficient** because garbage collector removes the unreferenced objects from heap memory.
- It is **automatically done** by the garbage collector(a part of JVM) so we don't need to make extra efforts.

### How can an object be unreferenced?

There are many ways:

- By nulling the reference
- By assigning a reference to another
- By anonymous object etc.

#### 1) By nulling a reference:

```

Employee e=new Employee();
e=null;

```

## 2) By assigning a reference to another:

```
Employee e1=new Employee();
Employee e2=new Employee();
e1=e2;//now the first object referred by e1 is available for garbage collection
```

## 3) By anonymous object:

```
new Employee();
```

### finalize() method

The finalize() method is invoked each time before the object is garbage collected. This method can be used to perform cleanup processing. This method is defined in Object class as:

```
protected void finalize(){}  
  
Note: The Garbage collector of JVM collects only those objects that are created by new keyword. So if you have created any object without new, you can use finalize method to perform cleanup processing (destroying remaining objects).
```

### gc() method

The gc() method is used to invoke the garbage collector to perform cleanup processing. The gc() is found in System and Runtime classes.

```
public static void gc(){}  
  
Note: Garbage collection is performed by a daemon thread called Garbage Collector(GC). This thread calls the finalize() method before object is garbage collected.
```

## Simple Example of garbage collection in java

```
public class TestGarbage1{
    public void finalize(){System.out.println("object is garbage collected");}
    public static void main(String args[]){
        TestGarbage1 s1=new TestGarbage1();
        TestGarbage1 s2=new TestGarbage1();
        s1=null;
        s2=null;
        System.gc();
    }
}
```

**Test it Now**

```
object is garbage collected
object is garbage collected
```



**Note:** Neither finalization nor garbage collection is guaranteed.

## Java Runtime class

**Java Runtime** class is used to interact with java runtime environment . Java Runtime class provides methods to execute a process, invoke GC, get total and free memory etc. There is only one instance of java.lang.Runtime class is available for one java application.

The **Runtime.getRuntime()** method returns the singleton instance of Runtime class.

### Important methods of Java Runtime class

No.	Method	Description
1)	public static Runtime getRuntime()	returns the instance of Runtime class.
2)	public void exit(int status)	terminates the current virtual machine.
3)	public void addShutdownHook(Thread hook)	registers new hook thread.
4)	public Process exec(String command) throws IOException	executes given command in a separate process.
5)	public int availableProcessors()	returns no. of available processors.
6)	public long freeMemory()	returns amount of free memory in JVM.
7)	public long totalMemory()	returns amount of total memory in JVM.

## Java Runtime exec() method

```
public class Runtime1{
    public static void main(String args[])throws Exception{
        Runtime.getRuntime().exec("notepad");//will open a new notepad
    }
}
```

## Java Runtime freeMemory() and totalMemory() method

In the given program, after creating 10000 instance, free memory will be less than the previous free memory. But after gc() call, you will get more free memory.

```
public class MemoryTest{
    public static void main(String args[])throws Exception{
        Runtime r=Runtime.getRuntime();
        System.out.println("Total Memory: "+r.totalMemory());
        System.out.println("Free Memory: "+r.freeMemory());

        for(int i=0;i<10000;i++){
            new MemoryTest();
        }
        System.out.println("After creating 10000 instance, Free Memory: "+r.freeMemory());
        System.gc();
        System.out.println("After gc(), Free Memory: "+r.freeMemory());
    }
}

Total Memory: 100139008
Free Memory: 99474824
After creating 10000 instance, Free Memory: 99310552
After gc(), Free Memory: 100182832
```

## Synchronization in Java

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

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

## Why use Synchronization

The synchronization is mainly used to

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

# Types of Synchronization

There are two types of synchronization

1. Process Synchronization
2. Thread Synchronization

Here, we will discuss only thread synchronization.

## Thread Synchronization

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

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

## Mutual Exclusive

Mutual Exclusive helps keep threads from interfering with one another while sharing data. This can be done by three ways in java:

1. by synchronized method
2. by synchronized block
3. by static synchronization

## Concept of Lock in Java

Synchronization is built around an internal entity known as the lock or monitor. Every object has an lock associated with it. By convention, a thread that needs consistent access to an object's fields has to acquire the object's lock before accessing them, and then release the lock when it's done with them.

From Java 5 the package `java.util.concurrent.locks` contains several lock implementations.

## Understanding the problem without Synchronization

In this example, there is no synchronization, so output is inconsistent. Let's see the example:

```
Class Table{  
  
void printTable(int n){//method not synchronized  
    for(int i=1;i<=5;i++){  
        System.out.println(n*i);  
        try{  
            Thread.sleep(400);  
        }catch(Exception e){System.out.println(e);}  
    }  
}  
  
class MyThread1 extends Thread{  
Table t;  
MyThread1(Table t){  
this.t=t;
```

```

}

public void run(){
t.printTable(5);
}

}

class MyThread2 extends Thread{
Table t;
MyThread2(Table t){
this.t=t;
}
public void run(){
t.printTable(100);
}
}

class TestSynchronization1{
public static void main(String args[]){
Table obj = new Table(); //only one object
MyThread1 t1=new MyThread1(obj);
MyThread2 t2=new MyThread2(obj);
t1.start();
t2.start();
}
}

Output: 5
    100
    10
    200
    15
    300
    20
    400
    25
    500

```

---

## Java synchronized method

If you declare any method as synchronized, it is known as synchronized method.

Synchronized method is used to lock an object for any shared resource.

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

```

//example of java synchronized method
class Table{
    synchronized void printTable(int n){ //synchronized method
        for(int i=1;i<=5;i++){
            System.out.println(n*i);
            try{
                Thread.sleep(400);
            }catch(Exception e){System.out.println(e);}
        }
    }

    class MyThread1 extends Thread{

```

```

Table t;
MyThread1(Table t){
this.t=t;
}
public void run(){
t.printTable(5);
}

}

class MyThread2 extends Thread{
Table t;
MyThread2(Table t){
this.t=t;
}
public void run(){
t.printTable(100);
}
}

public class TestSynchronization2{
public static void main(String args[]){
Table obj = new Table();//only one object
MyThread1 t1=new MyThread1(obj);
MyThread2 t2=new MyThread2(obj);
t1.start();
t2.start();
}
}

Output: 5
10
15
20
25
100
200
300
400
500

```

## Example of synchronized method by using anonymous class

In this program, we have created the two threads by anonymous class, so less coding is required.

```

//Program of synchronized method by using anonymous class
class Table{
synchronized void printTable(int n){//synchronized method
for(int i=1;i<=5;i++){
System.out.println(n*i);
try{
Thread.sleep(400);
}catch(Exception e){System.out.println(e);}
}
}

public class TestSynchronization3{
public static void main(String args[]){

```

```

final Table obj = new Table(); //only one object

Thread t1=new Thread(){
public void run(){
obj.printTable(5);
}
};

Thread t2=new Thread(){
public void run(){
obj.printTable(100);
}
};

t1.start();
t2.start();
}

}

Output: 5
    10
    15
    20
    25
    100
    200
    300
    400
    500

```

## Synchronized block in java

Synchronized block can be used to perform synchronization on any specific resource of the method.

Suppose you have 50 lines of code in your method, but you want to synchronize only 5 lines, you can use synchronized block.

If you put all the codes of the method in the synchronized block, it will work same as the synchronized method.

### Points to remember for Synchronized block

- Synchronized block is used to lock an object for any shared resource.
- Scope of synchronized block is smaller than the method.

### Syntax to use synchronized block

```

synchronized (object reference expression) {
    //code block
}

```

## Example of synchronized block

Let's see the simple example of synchronized block.

### Program of synchronized block

```

class Table{

void printTable(int n){

```

```

synchronized(this){//synchronized block
    for(int i=1;i<=5;i++){
        System.out.println(n*i);
        try{
            Thread.sleep(400);
        }catch(Exception e){System.out.println(e);}
    }
}
}//end of the method
}

class MyThread1 extends Thread{
Table t;
MyThread1(Table t){
this.t=t;
}
public void run(){
t.printTable(5);
}
}

class MyThread2 extends Thread{
Table t;
MyThread2(Table t){
this.t=t;
}
public void run(){
t.printTable(100);
}
}

public class TestSynchronizedBlock1{
public static void main(String args[]){
Table obj = new Table();//only one object
MyThread1 t1=new MyThread1(obj);
MyThread2 t2=new MyThread2(obj);
t1.start();
t2.start();
}
}

```

**Test it Now**

Output:5

```

10
15
20
25
100
200
300
400
500

```

Same Example of synchronized block by using anonymous class:

**//Program of synchronized block by using anonymous class**

```
class Table{
```

```

void printTable(int n){
    synchronized(this){//synchronized block
        for(int i=1;i<=5;i++){
            System.out.println(n*i);
            try{
                Thread.sleep(400);
            }catch(Exception e){System.out.println(e);}
        }
    }
}

public class TestSynchronizedBlock2{
public static void main(String args[]){
final Table obj = new Table();//only one object

Thread t1=new Thread(){
public void run(){
obj.printTable(5);
}
};

Thread t2=new Thread(){
public void run(){
obj.printTable(100);
}
};

t1.start();
t2.start();
}
}

```

**Test it Now**

Output:5

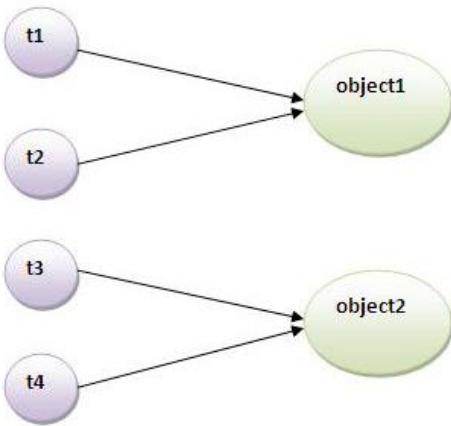
```

10
15
20
25
100
200
300
400
500

```

## Static synchronization

If you make any static method as synchronized, the lock will be on the class not on object.



## Problem without static synchronization

Suppose there are two objects of a shared class(e.g. Table) named object1 and object2.In case of synchronized method and synchronized block there cannot be interference between t1 and t2 or t3 and t4 because t1 and t2 both refers to a common object that have a single lock.But there can be interference between t1 and t3 or t2 and t4 because t1 acquires another lock and t3 acquires another lock.I want no interference between t1 and t3 or t2 and t4.Static synchronization solves this problem.

## Example of static synchronization

In this example we are applying synchronized keyword on the static method to perform static synchronization.

```

class Table{

    synchronized static void printTable(int n){
        for(int i=1;i<=10;i++){
            System.out.println(n*i);
            try{
                Thread.sleep(400);
            }catch(Exception e){}
        }
    }

    class MyThread1 extends Thread{
        public void run(){
            Table.printTable(1);
        }
    }

    class MyThread2 extends Thread{
        public void run(){
            Table.printTable(10);
        }
    }

    class MyThread3 extends Thread{
        public void run(){
            Table.printTable(100);
        }
    }

    class MyThread4 extends Thread{

```

```
public void run(){
Table.printTable(1000);
}

public class TestSynchronization4{
public static void main(String t[]){
MyThread1 t1=new MyThread1();
MyThread2 t2=new MyThread2();
MyThread3 t3=new MyThread3();
MyThread4 t4=new MyThread4();
t1.start();
t2.start();
t3.start();
t4.start();
}
}
```

**Test it Now**

Output: 1

2  
3  
4  
5  
6  
7  
8  
9  
10  
10  
20  
30  
40  
50  
60  
70  
80  
90  
100  
100  
200  
300  
400  
500  
600  
700  
800  
900  
1000  
1000  
2000  
3000  
4000  
5000  
6000  
7000  
8000  
9000  
10000

---

## Same example of static synchronization by anonymous class

In this example, we are using anonymous class to create the threads.

```
class Table{  
  
    synchronized static void printTable(int n){  
        for(int i=1;i<=10;i++){  
            System.out.println(n*i);  
            try{  
                Thread.sleep(400);  
            }catch(Exception e){}  
        }  
    }  
  
    public class TestSynchronization5 {  
        public static void main(String[] args) {  
  
            Thread t1=new Thread(){  
                public void run(){  
                    Table.printTable(1);  
                }  
            };  
  
            Thread t2=new Thread(){  
                public void run(){  
                    Table.printTable(10);  
                }  
            };  
  
            Thread t3=new Thread(){  
                public void run(){  
                    Table.printTable(100);  
                }  
            };  
  
            Thread t4=new Thread(){  
                public void run(){  
                    Table.printTable(1000);  
                }  
            };  
            t1.start();  
            t2.start();  
            t3.start();  
            t4.start();  
  
        }  
    }  
}
```

**Test it Now**

Output: 1  
2  
3  
4  
5  
6

```
7  
8  
9  
10  
20  
30  
40  
50  
60  
70  
80  
90  
100  
100  
200  
300  
400  
500  
600  
700  
800  
900  
1000  
1000  
2000  
3000  
4000  
5000  
6000  
7000  
8000  
9000  
10000
```

---

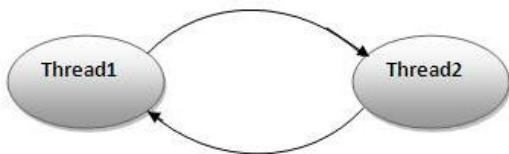
## Synchronized block on a class lock:

The block synchronizes on the lock of the object denoted by the reference .class name .class. A static synchronized method printTable(int n) in class Table is equivalent to the following declaration:

```
static void printTable(int n) {  
    synchronized (Table.class) {          // Synchronized block on class A  
        // ...  
    }  
}
```

## Deadlock in java

Deadlock in java is a part of multithreading. Deadlock can occur in a situation when a thread is waiting for an object lock, that is acquired by another thread and second thread is waiting for an object lock that is acquired by first thread. Since, both threads are waiting for each other to release the lock, the condition is called deadlock.



## Example of Deadlock in java

```

public class TestDeadlockExample1 {
    public static void main(String[] args) {
        final String resource1 = "ratan jaiswal";
        final String resource2 = "vimal jaiswal";
        // t1 tries to lock resource1 then resource2
        Thread t1 = new Thread() {
            public void run() {
                synchronized (resource1) {
                    System.out.println("Thread 1: locked resource 1");

                    try { Thread.sleep(100); } catch (Exception e) {}

                    synchronized (resource2) {
                        System.out.println("Thread 1: locked resource 2");
                    }
                }
            }
        };
        // t2 tries to lock resource2 then resource1
        Thread t2 = new Thread() {
            public void run() {
                synchronized (resource2) {
                    System.out.println("Thread 2: locked resource 2");

                    try { Thread.sleep(100); } catch (Exception e) {}

                    synchronized (resource1) {
                        System.out.println("Thread 2: locked resource 1");
                    }
                }
            }
        };
        t1.start();
        t2.start();
    }
}

Output: Thread 1: locked resource 1
        Thread 2: locked resource 2
    
```

# Inter-thread communication in Java

**Inter-thread communication** or **Co-operation** is all about allowing synchronized threads to communicate with each other.

Cooperation (Inter-thread communication) is a mechanism in which a thread is paused running in its critical section and another thread is allowed to enter (or lock) in the same critical section to be executed. It is implemented by following methods of **Object class**:

- `wait()`
- `notify()`
- `notifyAll()`

## 1) `wait()` method

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

The current thread must own this object's monitor, so it must be called from the synchronized method only otherwise it will throw exception.

Method	Description
<code>public final void wait() throws InterruptedException</code>	waits until object is notified.
<code>public final void wait(long timeout) throws InterruptedException</code>	waits for the specified amount of time.

## 2) `notify()` method

Wakes up a single thread that is waiting on this object's monitor. If any threads are waiting on this object, one of them is chosen to be awakened. The choice is arbitrary and occurs at the discretion of the implementation. Syntax:

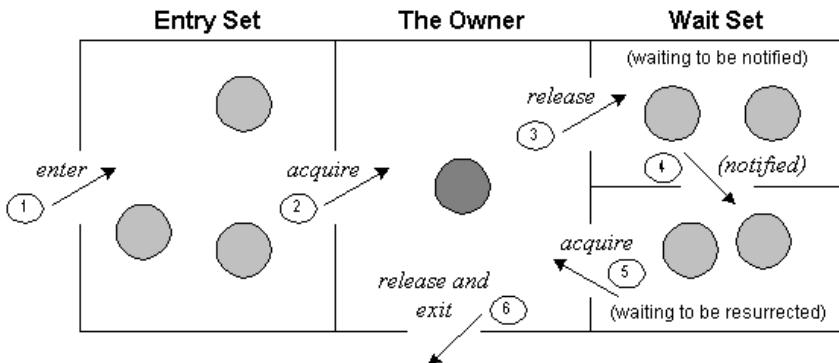
```
public final void notify()
```

## 3) `notifyAll()` method

Wakes up all threads that are waiting on this object's monitor. Syntax:

```
public final void notifyAll()
```

## Understanding the process of inter-thread communication



The point to point explanation of the above diagram is as follows:

1. Threads enter to acquire lock.
2. Lock is acquired by on thread.
3. Now thread goes to waiting state if you call `wait()` method on the object. Otherwise it releases the lock and exits.
4. If you call `notify()` or `notifyAll()` method, thread moves to the notified state (runnable state).
5. Now thread is available to acquire lock.
6. After completion of the task, thread releases the lock and exits the monitor state of the object.

## Why `wait()`, `notify()` and `notifyAll()` methods are defined in Object class not Thread class?

It is because they are related to lock and object has a lock.

## Difference between wait and sleep?

Let's see the important differences between wait and sleep methods.

wait()	sleep()
wait() method releases the lock	sleep() method doesn't release the lock.
is the method of Object class	is the method of Thread class
is the non-static method	is the static method
is the non-static method	is the static method
should be notified by notify() or notifyAll() methods	after the specified amount of time, sleep is completed.

## Example of inter thread communication in java

Let's see the simple example of inter thread communication.

```
class Customer{
int amount=10000;

synchronized void withdraw(int amount){
System.out.println("going to withdraw...");

if(this.amount<amount){
System.out.println("Less balance; waiting for deposit...");
try{wait();}catch(Exception e){}
}

this.amount-=amount;
System.out.println("withdraw completed...");
}

synchronized void deposit(int amount){
System.out.println("going to deposit...");
this.amount+=amount;
System.out.println("deposit completed... ");
notify();
}
}

class Test{
public static void main(String args[]){
final Customer c=new Customer();
new Thread(){
public void run(){c.withdraw(15000);}
}.start();
new Thread(){
public void run(){c.deposit(10000);}
}.start();

}}
}
```

Output: going to withdraw...  
Less balance; waiting for deposit...  
going to deposit...  
deposit completed...  
withdraw completed

## Interrupting a Thread:

If any thread is in sleeping or waiting state (i.e. sleep() or wait() is invoked), calling the interrupt() method on the thread, breaks out the sleeping or waiting state throwing InterruptedException. If the thread is not in the sleeping or waiting state, calling the interrupt() method performs normal behaviour and doesn't interrupt the thread but sets the interrupt flag to true. Let's first see the methods provided by the Thread class for thread interruption.

### The 3 methods provided by the Thread class for interrupting a thread

- **public void interrupt()**
- **public static boolean interrupted()**
- **public boolean isInterrupted()**

### Example of interrupting a thread that stops working

In this example, after interrupting the thread, we are propagating it, so it will stop working. If we don't want to stop the thread, we can handle it where sleep() or wait() method is invoked. Let's first see the example where we are propagating the exception.

```
class TestInterruptingThread1 extends Thread{
    public void run(){
        try{
            Thread.sleep(1000);
            System.out.println("task");
        }catch(InterruptedException e){
            throw new RuntimeException("Thread interrupted..."+e);
        }
    }

    public static void main(String args[]){
        TestInterruptingThread1 t1=new TestInterruptingThread1();
        t1.start();
        try{
            t1.interrupt();
        }catch(Exception e){System.out.println("Exception handled "+e);}
    }
}
```

**Test it Now**

[download this example](#)

```
Output:Exception in thread-0
java.lang.RuntimeException: Thread interrupted...
java.lang.InterruptedException: sleep interrupted
at A.run(A.java:7)
```

### Example of interrupting a thread that doesn't stop working

In this example, after interrupting the thread, we handle the exception, so it will break out the sleeping but will not stop working.

```
class TestInterruptingThread2 extends Thread{
```

```

public void run(){
try{
Thread.sleep(1000);
System.out.println("task");
} catch(InterruptedException e){
System.out.println("Exception handled "+e);
}
System.out.println("thread is running...");
}

public static void main(String args[]){
TestInterruptingThread2 t1=new TestInterruptingThread2();
t1.start();

t1.interrupt();

}
}

```

**Test it Now**

[download this example](#)

```

Output:Exception handled
java.lang.InterruptedException: sleep interrupted
thread is running...

```

## Example of interrupting thread that behaves normally

If thread is not in sleeping or waiting state, calling the interrupt() method sets the interrupted flag to true that can be used to stop the thread by the java programmer later.

```

class TestInterruptingThread3 extends Thread{

public void run(){
for(int i=1;i<=5;i++)
System.out.println(i);
}

public static void main(String args[]){
TestInterruptingThread3 t1=new TestInterruptingThread3();
t1.start();

t1.interrupt();

}
}

```

**Test it Now**

```

Output:1
2
3
4
5

```

## What about isInterrupted and interrupted method?

The isInterrupted() method returns the interrupted flag either true or false. The static interrupted() method returns the interrupted flag

after that it sets the flag to false if it is true.

```
public class TestInterruptingThread4 extends Thread{

    public void run(){
        for(int i=1;i<=2;i++){
            if(Thread.interrupted()){
                System.out.println("code for interrupted thread");
            }
            else{
                System.out.println("code for normal thread");
            }
        }
        //end of for loop
    }

    public static void main(String args[]){
        TestInterruptingThread4 t1=new TestInterruptingThread4();
        TestInterruptingThread4 t2=new TestInterruptingThread4();

        t1.start();
        t1.interrupt();

        t2.start();
    }
}
```

#### Test it Now

```
Output:Code for interrupted thread
       code for normal thread
       code for normal thread
       code for normal thread
```

[download this example](#)

## Reentrant Monitor in Java

According to Sun Microsystems, **Java monitors are reentrant** means java thread can reuse the same monitor for different synchronized methods if method is called from the method.

### Advantage of Reentrant Monitor

It eliminates the possibility of single thread deadlocking

Let's understand the java reentrant monitor by the example given below:

```
class Reentrant {
    public synchronized void m() {
        n();
        System.out.println("this is m() method");
    }
    public synchronized void n() {
```

```
System.out.println("this is n() method");
}
}
```

In this class, m and n are the synchronized methods. The m() method internally calls the n() method.

Now let's call the m() method on a thread. In the class given below, we are creating thread using anonymous class.

```
public class ReentrantExample{
public static void main(String args[]){
final ReentrantExample re=new ReentrantExample();

Thread t1=new Thread(){
public void run(){
re.m(); //calling method of Reentrant class
}
};

t1.start();
}}
```

#### Test it Now

```
Output: this is n() method
this is m() method
```

## Java I/O Tutorial

**Java I/O** (Input and Output) is used to process the input and produce the output based on the input.

Java uses the concept of stream to make I/O operation fast. The java.io package contains all the classes required for input and output operations.

We can perform **file handling in java** by java IO API.

## Stream

A stream is a sequence of data. In Java a stream is composed of bytes. It's called a stream because it's like a stream of water that continues to flow.

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

**1) System.out:** standard output stream

**2) System.in:** standard input stream

**3) System.err:** standard error stream

Let's see the code to print **output and error** message to the console.

```
System.out.println("simple message");
System.err.println("error message");
```

Let's see the code to get **input** from console.

```
int i=System.in.read(); //returns ASCII code of 1st character
System.out.println((char)i); //will print the character
```

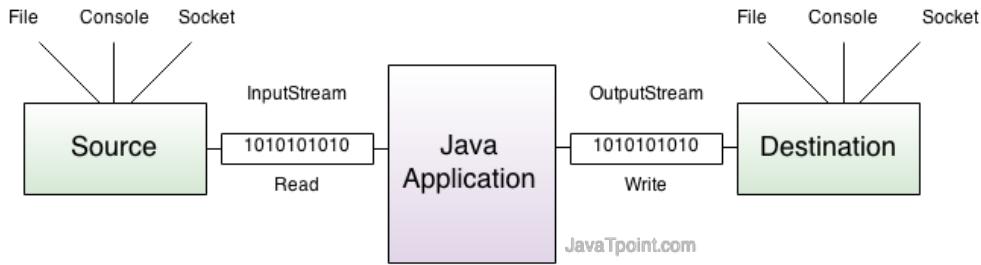
## OutputStream

Java application uses an output stream to write data to a destination, it may be a file, an array, peripheral device or socket.

## InputStream

Java application uses an input stream to read data from a source, it may be a file, an array, peripheral device or socket.

Let's understand working of Java OutputStream and InputStream by the figure given below.

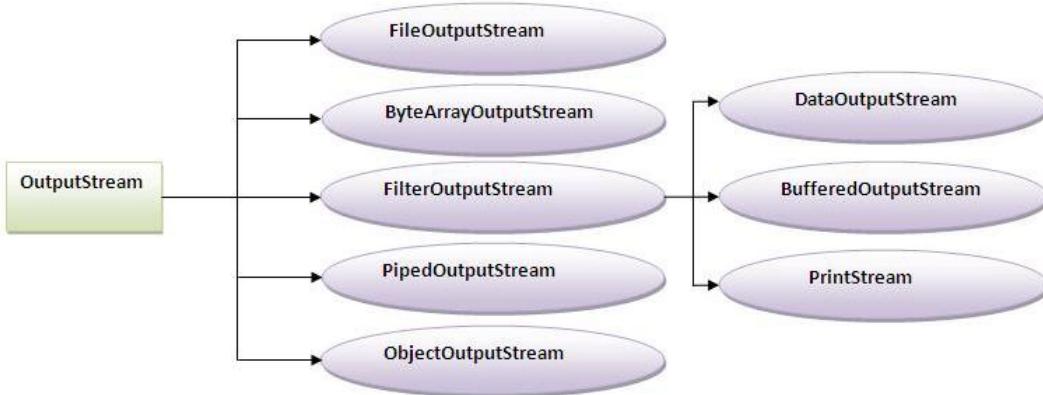


## OutputStream class

OutputStream class is an abstract class. It is the superclass of all classes representing an output stream of bytes. An output stream accepts output bytes and sends them to some sink.

### Commonly used methods of OutputStream class

Method	Description
<b>1) public void write(int) throws IOException:</b>	is used to write a byte to the current output stream.
<b>2) public void write(byte[]) throws IOException:</b>	is used to write an array of byte to the current output stream.
<b>3) public void flush() throws IOException:</b>	flushes the current output stream.
<b>4) public void close() throws IOException:</b>	is used to close the current output stream.

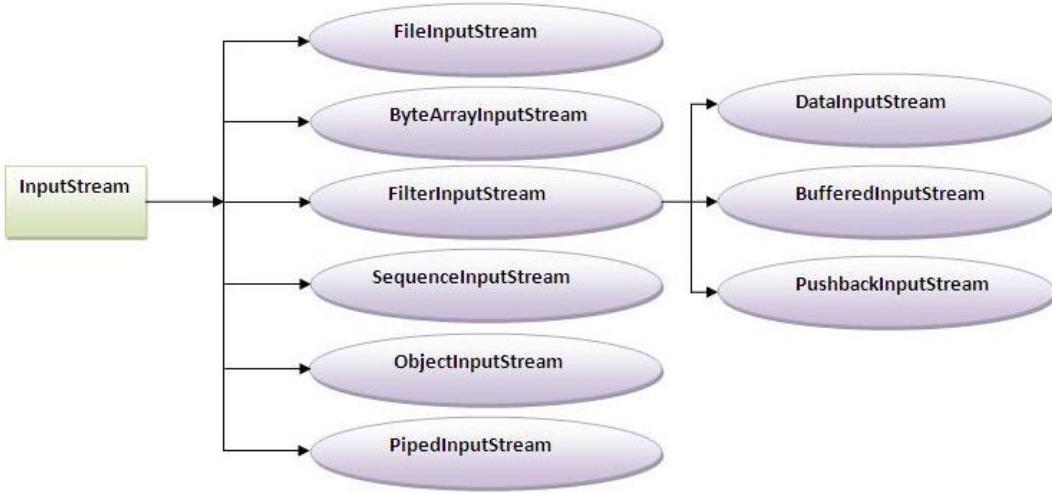


## InputStream class

InputStream class is an abstract class. It is the superclass of all classes representing an input stream of bytes.

### Commonly used methods of InputStream class

Method	Description
<b>1) public abstract int read() throws IOException:</b>	reads the next byte of data from the input stream. It returns -1 at the end of file.
<b>2) public int available() throws IOException:</b>	returns an estimate of the number of bytes that can be read from the current input stream.
<b>3) public void close() throws IOException:</b>	is used to close the current input stream.



## FileInputStream and FileOutputStream (File Handling)

In Java, `FileInputStream` and `FileOutputStream` classes are used to read and write data in file. In other words, they are used for file handling in java.

### Java FileOutputStream class

Java `FileOutputStream` is an output stream for writing data to a file.

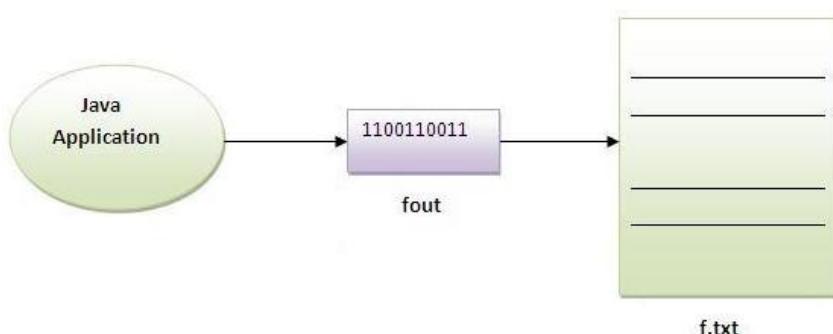
If you have to write primitive values then use `FileOutputStream`. Instead, for character-oriented data, prefer `FileWriter`. But you can write byte-oriented as well as character-oriented data.

### Example of Java FileOutputStream class

```

import java.io.*;
class Test{
    public static void main(String args[]){
        try{
            FileOutputStream fout=new FileOutputStream("abc.txt");
            String s="Sachin Tendulkar is my favourite player";
            byte b[]={s.getBytes()}; //converting string into byte array
            fout.write(b);
            fout.close();
            System.out.println("success...");
        }catch(Exception e){System.out.println(e);}
    }
}
  
```

Output:success...



## Java FileInputStream class

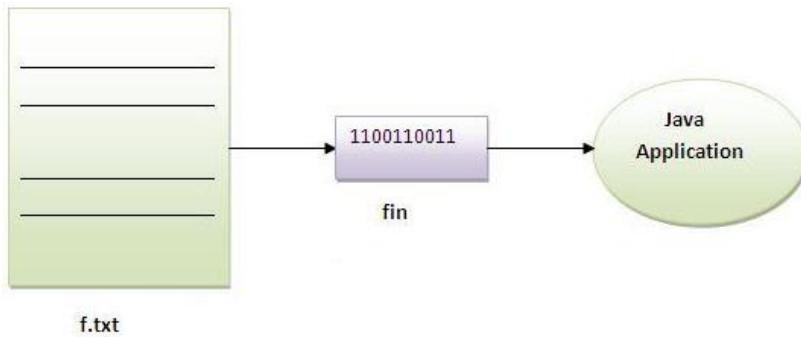
Java FileInputStream class obtains input bytes from a file. It is used for reading streams of raw bytes such as image data. For reading streams of characters, consider using FileReader.

It should be used to read byte-oriented data for example to read image, audio, video etc.

### Example of FileInputStream class

```
import java.io.*;
class SimpleRead{
public static void main(String args[]){
try{
    FileInputStream fin=new FileInputStream("abc.txt");
    int i=0;
    while((i=fin.read())!=-1){
        System.out.println((char)i);
    }
    fin.close();
}catch(Exception e){System.out.println(e);}
}
}
```

Output: Sachin is my favourite player.



### Example of Reading the data of current java file and writing it into another file

We can read the data of any file using the FileInputStream class whether it is java file, image file, video file etc. In this example, we are reading the data of C.java file and writing it into another file M.java.

```
import java.io.*;
class C{
public static void main(String args[])throws Exception{
FileInputStream fin=new FileInputStream("C.java");
FileOutputStream fout=new FileOutputStream("M.java");
int i=0;
while((i=fin.read())!=-1){
fout.write((byte)i);
}
fin.close();
}
}
```

[download this example](#)

## Java ByteArrayOutputStream class

Java ByteArrayOutputStream class is used to write data into multiple files. In this stream, the data is written into a byte array that can be written to multiple stream.

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

The buffer of ByteArrayOutputStream automatically grows according to data.



**Closing the ByteArrayOutputStream has no effect.**

## Constructors of ByteArrayOutputStream class

Constructor	Description
ByteArrayOutputStream()	creates a new byte array output stream with the initial capacity of 32 bytes, though its size increases if necessary.
ByteArrayOutputStream(int size)	creates a new byte array output stream, with a buffer capacity of the specified size, in bytes.

## Methods of ByteArrayOutputStream class

Method	Description
1) public synchronized void writeTo(OutputStream out) throws IOException	writes the complete contents of this byte array output stream to the specified output stream.
2) public void write(byte b) throws IOException	writes byte into this stream.
3) public void write(byte[] b) throws IOException	writes byte array into this stream.
4) public void flush()	flushes this stream.
5) public void close()	has no affect, it doesn't closes the bytarrayoutputstream.

## Java ByteArrayOutputStream Example

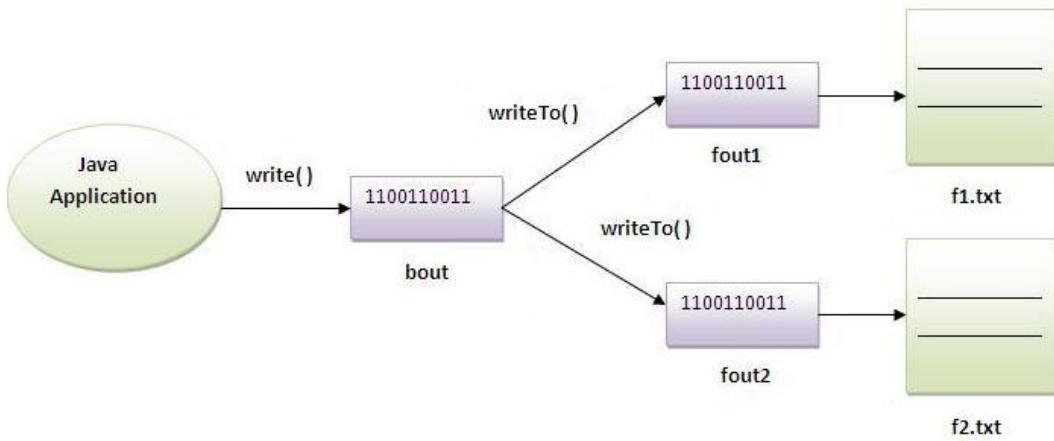
Let's see a simple example of java ByteArrayOutputStream class to write data into 2 files.

```
import java.io.*;
class S{
    public static void main(String args[])throws Exception{
        FileOutputStream fout1=new FileOutputStream("f1.txt");
        FileOutputStream fout2=new FileOutputStream("f2.txt");

        ByteArrayOutputStream bout=new ByteArrayOutputStream();
        bout.write(139);
        bout.writeTo(fout1);
        bout.writeTo(fout2);

        bout.flush();
        bout.close();//has no effect
        System.out.println("success...");
    }
}

success...
```



## Java SequenceInputStream class

Java SequenceInputStream class is used to read data from multiple streams. It reads data of streams one by one.

**Constructors of SequenceInputStream class:**

Constructor	Description
<b>1) SequenceInputStream(InputStream s1, InputStream s2)</b>	creates a new input stream by reading the data of two input stream in order, first s1 and then s2.
<b>2) SequenceInputStream(Enumeration e)</b>	creates a new input stream by reading the data of an enumeration whose type is InputStream.

### Simple example of SequenceInputStream class

In this example, we are printing the data of two files f1.txt and f2.txt.

```
import java.io.*;
class Simple{
    public static void main(String args[])throws Exception{
        FileInputStream fin1=new FileInputStream("f1.txt");
        FileInputStream fin2=new FileInputStream("f2.txt");

        SequenceInputStream sis=new SequenceInputStream(fin1,fin2);
        int i;
        while((i=sis.read())!=-1){
            System.out.println((char)i);
        }
        sis.close();
        fin1.close();
        fin2.close();
    }
}
```

### Example of SequenceInputStream that reads the data from two files

In this example, we are writing the data of two files f1.txt and f2.txt into another file named f3.txt.

```
//reading data of 2 files and writing it into one file

import java.io.*;
class Simple{
```

```

public static void main(String args[])throws Exception{

    FileInputStream fin1=new FileInputStream("f1.txt");
    FileInputStream fin2=new FileInputStream("f2.txt");

    FileOutputStream fout=new FileOutputStream("f3.txt");

    SequenceInputStream sis=new SequenceInputStream(fin1,fin2);
    int i;
    while((i=sis.read())!=-1)
    {
        fout.write(i);
    }
    sis.close();
    fout.close();
    fin1.close();
    fin2.close();

}

}

```

---

## Example of SequenceInputStream class that reads the data from multiple files using enumeration

If we need to read the data from more than two files, we need to have these information in the Enumeration object. Enumeration object can be get by calling elements method of the Vector class. Let's see the simple example where we are reading the data from the 4 files.

```

import java.io.*;
import java.util.*;

class B{
    public static void main(String args[])throws IOException{

        //creating the FileInputStream objects for all the files
        FileInputStream fin=new FileInputStream("A.java");
        FileInputStream fin2=new FileInputStream("abc2.txt");
        FileInputStream fin3=new FileInputStream("abc.txt");
        FileInputStream fin4=new FileInputStream("B.java");

        //creating Vector object to all the stream
        Vector v=new Vector();
        v.add(fin);
        v.add(fin2);
        v.add(fin3);
        v.add(fin4);

        //creating enumeration object by calling the elements method
        Enumeration e=v.elements();

        //passing the enumeration object in the constructor
        SequenceInputStream bin=new SequenceInputStream(e);
        int i=0;

        while((i=bin.read())!=-1){
            System.out.print((char)i);
        }

        bin.close();
    }
}

```

```

fin.close();
fin2.close();
}
}

```

[download this example of SequenceInputStream](#)

## Java BufferedOutputStream and BufferedInputStream

### Java BufferedOutputStream class

## Java FileWriter and FileReader (File Handling in java)

Java FileWriter and FileReader classes are used to write and read data from text files. These are character-oriented classes, used for file handling in java.

Java has suggested not to use the FileInputStream and FileOutputStream classes if you have to read and write the textual information.

### Java FileWriter class

Java FileWriter class is used to write character-oriented data to the file.

#### Constructors of FileWriter class

Constructor	Description
FileWriter(String file)	creates a new file. It gets file name in string.
FileWriter(File file)	creates a new file. It gets file name in File object.

#### Methods of FileWriter class

Method	Description
1) public void write(String text)	writes the string into FileWriter.
2) public void write(char c)	writes the char into FileWriter.
3) public void write(char[] c)	writes char array into FileWriter.
4) public void flush()	flushes the data of FileWriter.
5) public void close()	closes FileWriter.

### Java FileWriter Example

In this example, we are writing the data in the file abc.txt.

```

import java.io.*;
class Simple{
public static void main(String args[]){
try{
FileWriter fw=new FileWriter("abc.txt");
fw.write("my name is sachin");
fw.close();
}catch(Exception e){System.out.println(e);}
System.out.println("success");
}
}

```

Output:

success...

## Java FileReader class

Java FileReader class is used to read data from the file. It returns data in byte format like FileInputStream class.

### Constructors of FileWriter class

Constructor	Description
FileReader(String file)	It gets filename in string. It opens the given file in read mode. If file doesn't exist, it throws FileNotFoundException.
FileReader(File file)	It gets filename in file instance. It opens the given file in read mode. If file doesn't exist, it throws FileNotFoundException.

### Methods of FileReader class

Method	Description
1) public int read()	returns a character in ASCII form. It returns -1 at the end of file.
2) public void close()	closes FileReader.

## Java FileReader Example

In this example, we are reading the data from the file abc.txt file.

```
import java.io.*;
class Simple{
    public static void main(String args[])throws Exception{
        FileReader fr=new FileReader("abc.txt");
        int i;
        while((i=fr.read())!=-1)
            System.out.println((char)i);

        fr.close();
    }
}
```

Output:

```
my name is sachin
```

## CharArrayWriter class:

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

### Example of CharArrayWriter class:

In this example, we are writing a common data to 4 files a.txt, b.txt, c.txt and d.txt.

```
import java.io.*;
class Simple{
    public static void main(String args[])throws Exception{
```

```

CharArrayWriter out=new CharArrayWriter();
out.write("my name is");

FileWriter f1=new FileWriter("a.txt");
FileWriter f2=new FileWriter("b.txt");
FileWriter f3=new FileWriter("c.txt");
FileWriter f4=new FileWriter("d.txt");

out.writeTo(f1);
out.writeTo(f2);
out.writeTo(f3);
out.writeTo(f4);

f1.close();
f2.close();
f3.close();
f4.close();
}
}

```

## Reading data from keyboard

There are many ways to read data from the keyboard. For example:

- InputStreamReader
- Console
- Scanner
- DataInputStream etc.

## InputStreamReader class

InputStreamReader class can be used to read data from keyboard. It performs two tasks:

- connects to input stream of keyboard
- converts the byte-oriented stream into character-oriented stream

## BufferedReader class

BufferedReader class can be used to read data line by line by readLine() method.

## Example of reading data from keyboard by InputStreamReader and BufferdReader class

In this example, we are connecting the BufferedReader stream with the InputStreamReader stream for reading the line by line data from the keyboard.

```

import java.io.*;
class G5{
public static void main(String args[])throws Exception{

InputStreamReader r=new InputStreamReader(System.in);
BufferedReader br=new BufferedReader(r);

System.out.println("Enter your name");
String name=br.readLine();
System.out.println("Welcome "+name);
}

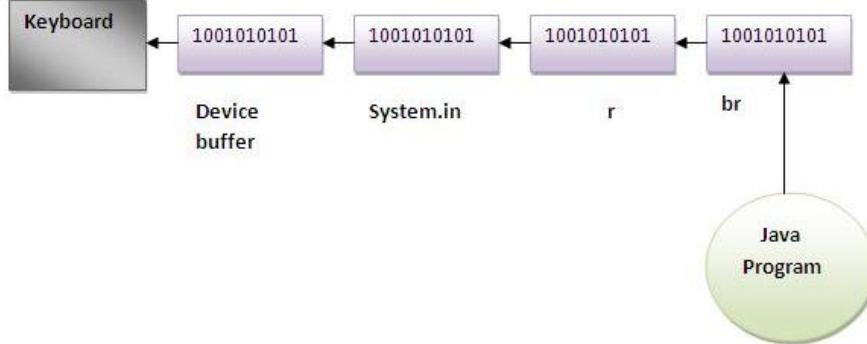
```

```

}
}

Output:Enter your name
Amit
Welcome Amit

```



## Another Example of reading data from keyboard by InputStreamReader and BufferedReader class until the user writes stop

In this example, we are reading and printing the data until the user prints stop.

```

import java.io.*;
class G5{
public static void main(String args[])throws Exception{

InputStreamReader r=new InputStreamReader(System.in);
BufferedReader br=new BufferedReader(r);

String name="";

while(!name.equals("stop")){
System.out.println("Enter data: ");
name=br.readLine();
System.out.println("data is: "+name);
}

br.close();
r.close();
}
}

Output:Enter data: Amit
data is: Amit
Enter data: 10
data is: 10
Enter data: stop
data is: stop

```

## Java Console class

The Java Console class is used to get input from console. It provides methods to read text and password.

If you read password using Console class, it will not be displayed to the user.

The `java.io.Console` class is attached with system console internally. The `Console` class is introduced since 1.5.

Let's see a simple example to read text from console.

```
String text=System.console().readLine();
System.out.println("Text is: "+text);
```

## Methods of Console class

Let's see the commonly used methods of Console class.

Method	Description
1) public String readLine()	is used to read a single line of text from the console.
2) public String readLine(String fmt, Object... args)	it provides a formatted prompt then reads the single line of text from the console.
3) public char[] readPassword()	is used to read password that is not being displayed on the console.
4) public char[] readPassword(String fmt, Object... args)	it provides a formatted prompt then reads the password that is not being displayed on the console.

## How to get the object of Console

System class provides a static method `console()` that returns the unique instance of Console class.

```
public static Console console(){};
```

Let's see the code to get the instance of Console class.

```
Console c=System.console();
```

## Java Console Example

```
import java.io.*;
class ReadStringTest{
public static void main(String args[]){
Console c=System.console();
System.out.println("Enter your name: ");
String n=c.readLine();
System.out.println("Welcome "+n);
}
}
```

Output:

```
Enter your name: james gosling
Welcome james gosling
```

## Java Console Example to read password

```
import java.io.*;
class ReadPasswordTest{
public static void main(String args[]){
Console c=System.console();
System.out.println("Enter password: ");
char[] ch=c.readPassword();
String pass=String.valueOf(ch);//converting char array into string
System.out.println("Password is: "+pass);
}
}
```

Output:

```
Enter password:
Password is: sonoo
```

# Java Scanner class

There are various ways to read input from the keyboard, the `java.util.Scanner` class is one of them.

The **Java Scanner** class breaks the input into tokens using a delimiter that is whitespace by default. It provides many methods to read and parse various primitive values.

Java Scanner class is widely used to parse text for string and primitive types using regular expression.

Java Scanner class extends Object class and implements Iterator and Closeable interfaces.

## Commonly used methods of Scanner class

There is a list of commonly used Scanner class methods:

Method	Description
<code>public String next()</code>	it returns the next token from the scanner.
<code>public String nextLine()</code>	it moves the scanner position to the next line and returns the value as a string.
<code>public byte nextByte()</code>	it scans the next token as a byte.
<code>public short nextShort()</code>	it scans the next token as a short value.
<code>public int nextInt()</code>	it scans the next token as an int value.
<code>public long nextLong()</code>	it scans the next token as a long value.
<code>public float nextFloat()</code>	it scans the next token as a float value.
<code>public double nextDouble()</code>	it scans the next token as a double value.

## Java Scanner Example to get input from console

Let's see the simple example of the Java Scanner class which reads the int, string and double value as an input:

```
import java.util.Scanner;
class ScannerTest{
    public static void main(String args[]){
        Scanner sc=new Scanner(System.in);

        System.out.println("Enter your rollno");
        int rollno=sc.nextInt();
        System.out.println("Enter your name");
        String name=sc.next();
        System.out.println("Enter your fee");
        double fee=sc.nextDouble();
        System.out.println("Rollno:"+rollno+" name:"+name+" fee:"+fee);
        sc.close();
    }
}
```

[download this scanner example](#)

Output:

```
Enter your rollno
111
Enter your name
Ratan
Enter
450000
Rollno:111 name:Ratan fee:450000
```

## Java Scanner Example with delimiter

Let's see the example of Scanner class with delimiter. The \s represents whitespace.

```
import java.util.*;
public class ScannerTest2{
public static void main(String args[]){
    String input = "10 tea 20 coffee 30 tea biscuits";
    Scanner s = new Scanner(input).useDelimiter("\s");
    System.out.println(s.nextInt());
    System.out.println(s.next());
    System.out.println(s.nextInt());
    System.out.println(s.next());
    s.close();
}}
```

Output:

```
10
tea
20
coffee
```

## java.io.PrintStream class:

The PrintStream class provides methods to write data to another stream. The PrintStream class automatically flushes the data so there is no need to call flush() method. Moreover, its methods don't throw IOException.

### Commonly used methods of PrintStream class:

There are many methods in PrintStream class. Let's see commonly used methods of PrintStream class:

- **public void print(boolean b):** it prints the specified boolean value.
- **public void print(char c):** it prints the specified char value.
- **public void print(char[] c):** it prints the specified character array values.
- **public void print(int i):** it prints the specified int value.
- **public void print(long l):** it prints the specified long value.
- **public void print(float f):** it prints the specified float value.
- **public void print(double d):** it prints the specified double value.
- **public void print(String s):** it prints the specified string value.
- **public void print(Object obj):** it prints the specified object value.
- **public void println(boolean b):** it prints the specified boolean value and terminates the line.
- **public void println(char c):** it prints the specified char value and terminates the line.
- **public void println(char[] c):** it prints the specified character array values and terminates the line.
- **public void println(int i):** it prints the specified int value and terminates the line.
- **public void println(long l):** it prints the specified long value and terminates the line.
- **public void println(float f):** it prints the specified float value and terminates the line.
- **public void println(double d):** it prints the specified double value and terminates the line.
- **public void println(String s):** it prints the specified string value and terminates the line.
- **public void println(Object obj):** it prints the specified object value and terminates the line.
- **public void println():** it terminates the line only.
- **public void printf(Object format, Object... args):** it writes the formatted string to the current stream.
- **public void printf(Locale l, Object format, Object... args):** it writes the formatted string to the current stream.
- **public void format(Object format, Object... args):** it writes the formatted string to the current stream using specified format.
- **public void format(Locale l, Object format, Object... args):** it writes the formatted string to the current stream using specified format.

---

### Example of java.io.PrintStream class:

In this example, we are simply printing integer and string values.

```
import java.io.*;
class PrintStreamTest{
    public static void main(String args[])throws Exception{

        FileOutputStream fout=new FileOutputStream("mfile.txt");
        PrintStream pout=new PrintStream(fout);
        pout.println(1900);
        pout.println("Hello Java");
        pout.println("Welcome to Java");
        pout.close();
        fout.close();

    }
}
```

[download this PrintStream example](#)

---

## Example of printf() method of java.io.PrintStream class:

Let's see the simple example of printing integer value by format specifier.

```
class PrintStreamTest{
    public static void main(String args[]){
        int a=10;
        System.out.printf("%d",a); //Note, out is the object of PrintStream class
    }
}

Output:10
```

## Compressing and Uncompressing File

The DeflaterOutputStream and InflaterInputStream classes provide mechanism to compress and uncompress the data in the **deflate compression format**.

---

### DeflaterOutputStream class:

The DeflaterOutputStream class is used to compress the data in the deflate compression format. It provides facility to the other compression filters, such as GZIPOutputStream.

## Example of Compressing file using DeflaterOutputStream class

In this example, we are reading data of a file and compressing it into another file using DeflaterOutputStream class. You can compress any file, here we are compressing the Deflater.java file

```
import java.io.*;
import java.util.zip.*;

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

        try{
            FileInputStream fin=new FileInputStream("Deflater.java");

            FileOutputStream fout=new FileOutputStream("def.txt");
            DeflaterOutputStream out=new DeflaterOutputStream(fout);


```

```

int i;
while((i=fin.read())!=-1){
out.write((byte)i);
out.flush();
}

fin.close();
out.close();

}catch(Exception e){System.out.println(e);}
System.out.println("rest of the code");
}
}

```

[download this example](#)

---

### InflaterInputStream class:

The InflaterInputStream class is used to uncompress the file in the deflate compression format. It provides facility to the other uncompression filters, such as GZIPInputStream class.

### Example of uncompressing file using InflaterInputStream class

In this example, we are decompressing the compressed file def.txt into D.java .

```

import java.io.*;
import java.util.zip.*;

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

try{
FileInputStream fin=new FileInputStream("def.txt");
InflaterInputStream in=new InflaterInputStream(fin);

FileOutputStream fout=new FileOutputStream("D.java");

int i;
while((i=in.read())!=-1){
fout.write((byte)i);
fout.flush();
}

fin.close();
fout.close();
in.close();

}catch(Exception e){System.out.println(e);}
System.out.println("rest of the code");
}
}

```

[download this example](#)

### PipedInputStream and PipedOutputStream classes

The PipedInputStream and PipedOutputStream classes can be used to read and write data simultaneously. Both streams are connected with

each other using the connect() method of the PipedOutputStream class.

## Example of PipedInputStream and PipedOutputStream classes using threads

Here, we have created two threads t1 and t2. The **t1** thread writes the data using the PipedOutputStream object and the **t2** thread reads the data from that pipe using the PipedInputStream object. Both the piped stream object are connected with each other.

```
import java.io.*;
class PipedWR{
public static void main(String args[])throws Exception{
final PipedOutputStream pout=new PipedOutputStream();
final PipedInputStream pin=new PipedInputStream();

pout.connect(pin);//connecting the streams
//creating one thread t1 which writes the data
Thread t1=new Thread(){
public void run(){
for(int i=65;i<=90;i++){
try{
pout.write(i);
Thread.sleep(1000);
}catch(Exception e){}
}
}
};

//creating another thread t2 which reads the data
Thread t2=new Thread(){
public void run(){
try{
for(int i=65;i<=90;i++)
System.out.println(pin.read());
}catch(Exception e){}
}
};
//starting both threads
t1.start();
t2.start();
}}
```

[download this example](#)

## Serialization in Java

**Serialization in java** is a mechanism of *writing the state of an object into a byte stream* .

It is mainly used in Hibernate, RMI, JPA, EJB, JMS technologies.

The reverse operation of serialization is called *deserialization*.

The String class and all the wrapper classes implements *java.io.Serializable* interface by default.

## Java Transient Keyword

**Java transient** keyword is used in serialization. If you define any data member as transient, it will not be serialized.

Let's take an example, I have declared a class as Student, it has three data members id, name and age. If you serialize the object, all the values will be serialized but I don't want to serialize one value, e.g. age then we can declare the age data member as transient.

## Example of Java Transient Keyword

In this example, we have created the two classes Student and PersistExample. The age data member of the Student class is declared as transient, its value will not be serialized.

If you deserialize the object, you will get the default value for transient variable.

Let's create a class with transient variable.

```
import java.io.Serializable;
public class Student implements Serializable{
    int id;
    String name;
    transient int age;//Now it will not be serialized
    public Student(int id, String name,int age) {
        this.id = id;
        this.name = name;
        this.age=age;
    }
}
```

Now write the code to serialize the object.

```
import java.io.*;
class PersistExample{
    public static void main(String args[])throws Exception{
        Student s1 =new Student(211,"ravi",22);//creating object
        //writing object into file
        FileOutputStream f=new FileOutputStream("f.txt");
        ObjectOutputStream out=new ObjectOutputStream(f);
        out.writeObject(s1);
        out.flush();

        out.close();
        f.close();
        System.out.println("success");
    }
}
```

Output:

```
success
```

Now write the code for deserialization.

```
import java.io.*;
class DePersist{
    public static void main(String args[])throws Exception{
        ObjectInputStream in=new ObjectInputStream(new FileInputStream("f.txt"));
        Student s=(Student)in.readObject();
        System.out.println(s.id+ " "+s.name+ " "+s.age);
        in.close();
    }
}

211 ravi 0
```

As you can see, printing age of the student returns 0 because value of age was not serialized.

## Java Networking

Java Networking is a concept of connecting two or more computing devices together so that we can share resources.

Java socket programming provides facility to share data between different computing devices.

### Advantage of Java Networking

1. sharing resources
2. centralize software management

## Java Networking Terminology

The widely used java networking terminologies are given below:

1. IP Address
2. Protocol
3. Port Number
4. MAC Address
5. Connection-oriented and connection-less protocol
6. Socket

### 1) IP Address

IP address is a unique number assigned to a node of a network e.g. 192.168.0.1 . It is composed of octets that range from 0 to 255.

It is a logical address that can be changed.

### 2) Protocol

A protocol is a set of rules basically that is followed for communication. For example:

- TCP
- FTP
- Telnet
- SMTP
- POP etc.

### 3) Port Number

The port number is used to uniquely identify different applications. It acts as a communication endpoint between applications.

The port number is associated with the IP address for communication between two applications.

### 4) MAC Address

MAC (Media Access Control) Address is a unique identifier of NIC (Network Interface Controller). A network node can have multiple NIC but each with unique MAC.

### 5) Connection-oriented and connection-less protocol

In connection-oriented protocol, acknowledgement is sent by the receiver. So it is reliable but slow. The example of connection-oriented protocol is TCP.

But, in connection-less protocol, acknowledgement is not sent by the receiver. So it is not reliable but fast. The example of connection-less protocol is UDP.

### 6) Socket

A socket is an endpoint between two way communication.

Visit next page for java socket programming.

# Java Socket Programming

Java Socket programming is used for communication between the applications running on different JRE.

Java Socket programming can be connection-oriented or connection-less.

Socket and ServerSocket classes are used for connection-oriented socket programming and DatagramSocket and DatagramPacket classes are used for connection-less socket programming.

The client in socket programming must know two information:

1. IP Address of Server, and
2. Port number.

**Description**

## Socket class

A socket is simply an endpoint for communications between the machines. The Socket class can be used to create a socket.

## Important methods

### Method

## Java URL

The **Java URL** class represents an URL. URL is an acronym for Uniform Resource Locator. It points to a resource on the World Wide Web.

For example:

`http://www.javatpoint.com/java-tutorial`

A URL contains many information:

1. **Protocol:** In this case, http is the protocol.
2. **Server name or IP Address:** In this case, www.javatpoint.com is the server name.
3. **Port Number:** It is an optional attribute. If we write `http://www.javatpoint.com:80/sonoojaiswal/`, 80 is the port number. If port number is not mentioned in the URL, it returns -1.
4. **File Name or directory name:** In this case, index.jsp is the file name.

## Commonly used methods of Java URL class

## Java URLConnection class

The **Java URLConnection** class represents a communication link between the URL and the application. This class can be used to read and write data to the specified resource referred by the URL.

## How to get the object of URLConnection class

The `openConnection()` method of URL class returns the object of URLConnection class. Syntax:

```
public URLConnection openConnection() throws IOException{}
```

## Displaying source code of a webpage by URLConnecton class

The URLConnection class provides many methods, we can display all the data of a webpage by using the `getInputStream()` method. The `getInputStream()` method returns all the data of the specified URL in the stream that can be read and displayed.

## Example of Java URLConnecton class

```
import java.io.*;
import java.net.*;
public class URLConnectionExample {
public static void main(String[] args){
try{
URL url=new URL("http://www.javatpoint.com/java-tutorial");
URLConnection urlcon=url.openConnection();
```

```

InputStream stream=urlcon.getInputStream();
int i;
while((i=stream.read())!=-1){
System.out.print((char)i);
}
}catch(Exception e){System.out.println(e);}
}
}

```

[download this example](#)

## Java HttpURLConnection class

The **Java HttpURLConnection** class is http specific URLConnection. It works for HTTP protocol only.

By the help of HttpURLConnection class, you can get information of any HTTP URL such as header information, status code, response code etc.

The `java.net.HttpURLConnection` is subclass of `URLConnection` class.

### How to get the object of HttpURLConnection class

The `openConnection()` method of `URL` class returns the object of `URLConnection` class. Syntax:

```
public URLConnection openConnection() throws IOException{}
```

You can typecast it to `HttpURLConnection` type as given below.

```
URL url=new URL("http://www.javatpoint.com/java-tutorial");
HttpURLConnection huc=(HttpURLConnection)url.openConnection();
```

## Java HttpURLConnection Example

### Java InetAddress class

**Java InetAddress** class represents an IP address. The `java.net.InetAddress` class provides methods to get the IP of any host name for example `www.javatpoint.com`, `www.google.com`, `www.facebook.com` etc.

### Commonly used methods of InetAddress class

Method	Description
<code>public static InetAddress getByName(String host) throws UnknownHostException</code>	it returns the instance of <code>InetAddress</code> containing LocalHost IP and name.
<code>public static InetAddress getLocalHost() throws UnknownHostException</code>	it returns the instance of <code>InetAddress</code> containing local host name and address.
<code>public String getHostName()</code>	it returns the host name of the IP address.
<code>public String getHostAddress()</code>	it returns the IP address in string format.

## Example of Java InetAddress class

Let's see a simple example of `InetAddress` class to get ip address of `www.javatpoint.com` website.

```

import java.io.*;
import java.net.*;
public class InetDemo{
public static void main(String[] args){
try{
InetAddress ip=InetAddress.getByName("www.javatpoint.com");

```

```

System.out.println("Host Name: "+ip.getHostName());
System.out.println("IP Address: "+ip.getHostAddress());
}catch(Exception e){System.out.println(e);}
}
}

```

[Test it Now](#)

Output:

```

Host Name: www.javatpoint.com
IP Address: 206.51.231.148

```

[download this example](#)

## Java DatagramSocket and DatagramPacket

Java DatagramSocket and DatagramPacket classes are used for connection-less socket programming.

---

### Java DatagramSocket class

**Java DatagramSocket** class represents a connection-less socket for sending and receiving datagram packets.

A datagram is basically an information but there is no guarantee of its content, arrival or arrival time.

#### Commonly used Constructors of DatagramSocket class

- **DatagramSocket()** **throws SocketException**: it creates a datagram socket and binds it with the available Port Number on the localhost machine.
  - **DatagramSocket(int port)** **throws SocketException**: it creates a datagram socket and binds it with the given Port Number.
  - **DatagramSocket(int port, InetAddress address)** **throws SocketException**: it creates a datagram socket and binds it with the specified port number and host address.
- 

### Java DatagramPacket class

**Java DatagramPacket** is a message that can be sent or received. If you send multiple packet, it may arrive in any order. Additionally, packet delivery is not guaranteed.

#### Commonly used Constructors of DatagramPacket class

- **DatagramPacket(byte[] barr, int length)**: it creates a datagram packet. This constructor is used to receive the packets.
  - **DatagramPacket(byte[] barr, int length, InetAddress address, int port)**: it creates a datagram packet. This constructor is used to send the packets.
- 

## Example of Sending DatagramPacket by DatagramSocket

```

//DSender.java
import java.net.*;
public class DSender{
    public static void main(String[] args) throws Exception {
        DatagramSocket ds = new DatagramSocket();
        String str = "Welcome java";
        InetAddress ip = InetAddress.getByName("127.0.0.1");

        DatagramPacket dp = new DatagramPacket(str.getBytes(), str.length(), ip, 3000);
        ds.send(dp);
        ds.close();
    }
}

```

## Example of Receiving DatagramPacket by DatagramSocket

```
//DReceiver.java
import java.net.*;
public class DReceiver{
    public static void main(String[] args) throws Exception {
        DatagramSocket ds = new DatagramSocket(3000);
        byte[] buf = new byte[1024];
        DatagramPacket dp = new DatagramPacket(buf, 1024);
        ds.receive(dp);
        String str = new String(dp.getData(), 0, dp.getLength());
        System.out.println(str);
        ds.close();
    }
}
```

[download this example](#)

## Java AWT Tutorial

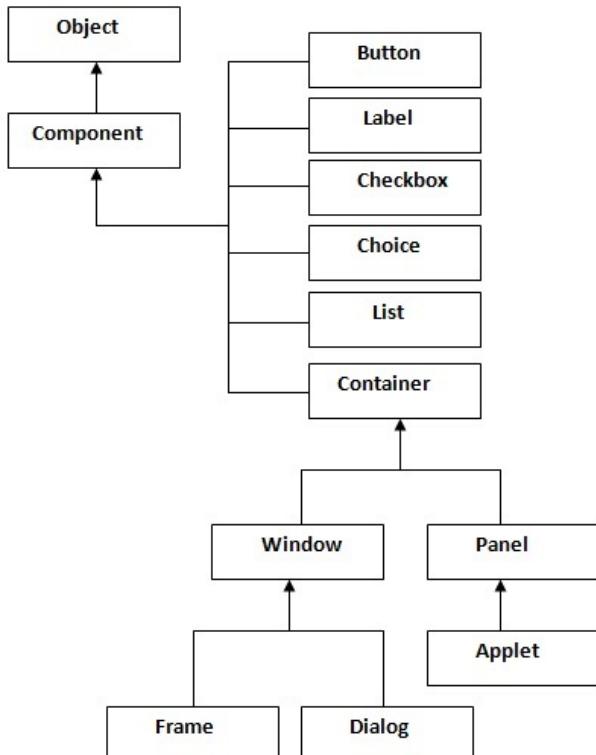
**Java AWT** (Abstract Windowing Toolkit) is an API to develop GUI or window-based application in java .

Java AWT components are platform-dependent i.e. components are displayed according to the view of operating system. AWT is heavyweight i.e. its components uses the resources of system.

The `java.awt` package provides classes for AWT api such as `TextField`, `Label`, `pre`, `RadioButton`, `CheckBox`, `Choice`, `List` etc.

## Java AWT Hierarchy

The hierarchy of Java AWT classes are given below.



## Container

The Container is a component in AWT that can contain another components like buttons, textfields, labels etc. The classes that extends Container class are known as container such as Frame, Dialog and Panel.

---

## Window

The window is the container that have no borders and menu bars. You must use frame, dialog or another window for creating a window.

---

## Panel

The Panel is the container that doesn't contain title bar and menu bars. It can have other components like button, textfield etc.

---

## Frame

The Frame is the container that contain title bar and can have menu bars. It can have other components like button, textfield etc.

---

## Useful Methods of Component class

Method	Description
public void add(Component c)	inserts a component on this component.
public void setSize(int width,int height)	sets the size (width and height) of the component.
public void setLayout(LayoutManager m)	defines the layout manager for the component.
public void setVisible(boolean status)	changes the visibility of the component, by default false.

## Java AWT Example

To create simple awt example, you need a frame. There are two ways to create a frame in AWT.

- By extending Frame class (inheritance)
  - By creating the object of Frame class (association)
- 

## Simple example of AWT by inheritance

```
import java.awt.*;
class First extends Frame{
First(){
Button b=new Button("click me");
b.setBounds(30,100,80,30);// setting button position

add(b);//adding button into frame
setSize(300,300);//frame size 300 width and 300 height
setLayout(null);//no layout manager
setVisible(true);//now frame will be visible, by default not visible
}
public static void main(String args[]){
First f=new First();
}}
```

[download this example](#)

The setBounds(int xaxis, int yaxis, int width, int height) method is used in the above example that sets the position of the awt button.



---

## Simple example of AWT by association

```
import java.awt.*;
class First2{
First2(){
Frame f=new Frame();

Button b=new Button("click me");
b.setBounds(30,50,80,30);

f.add(b);
f.setSize(300,300);
f.setLayout(null);
f.setVisible(true);
}
public static void main(String args[]){
First2 f=new First2();
}}
```

[download this example](#)

## Event and Listener (Java Event Handling)

Changing the state of an object is known as an event. For example, click on button, dragging mouse etc. The `java.awt.event` package provides many event classes and Listener interfaces for event handling.

### Event classes and Listener interfaces:

Event Classes	Listener Interfaces
ActionEvent	ActionListener
MouseEvent	MouseListener and MouseMotionListener
MouseWheelEvent	MouseWheelListener
KeyEvent	KeyListener
ItemEvent	ItemListener
TextEvent	TextListener

AdjustmentEvent	AdjustmentListener
WindowEvent	WindowListener
ComponentEvent	ComponentListener
ContainerEvent	ContainerListener
FocusEvent	FocusListener

## Steps to perform Event Handling

Following steps are required to perform event handling:

1. Implement the Listener interface and overrides its methods
2. Register the component with the Listener

---

For registering the component with the Listener, many classes provide the registration methods. For example:

- **Button**
  - public void addActionListener(ActionListener a){}
- **MenuItem**
  - public void addActionListener(ActionListener a){}
- **TextField**
  - public void addActionListener(ActionListener a){}
  - public void addTextListener(TextListener a){}
- **pre**
  - public void addTextListener(TextListener a){}
- **Checkbox**
  - public void addItemListener(ItemListener a){}
- **Choice**
  - public void addItemListener(ItemListener a){}
- **List**
  - public void addActionListener(ActionListener a){}
  - public void addItemListener(ItemListener a){}

---

## EventHandling Codes:

We can put the event handling code into one of the following places:

1. Same class
2. Other class
3. Anonymous class

## Example of event handling within class:

```

import java.awt.*;
import java.awt.event.*;

class AEvent extends Frame implements ActionListener{
TextField tf;
AEvent(){
tf=new TextField();
tf.setBounds(60,50,170,20);

Button b=new Button("click me");
b.setBounds(100,120,80,30);

```

```

b.addActionListener(this);

add(b);add(tf);

setSize(300,300);
setLayout(null);
setVisible(true);

}

public void actionPerformed(ActionEvent e){
tf.setText("Welcome");
}

public static void main(String args[]){
new AEvent();
}
}

```

**public void setBounds(int xaxis, int yaxis, int width, int height);** have been used in the above example that sets the position of the component it may be button, textfield etc.



## 2) Example of event handling by Outer class:

```

import java.awt.*;
import java.awt.event.*;
class AEvent2 extends Frame{
TextField tf;
AEvent2(){
tf=new TextField();
tf.setBounds(60,50,170,20);

Button b=new Button("click me");
b.setBounds(100,120,80,30);

Outer o=new Outer(this);
b.addActionListener(o);//passing outer class instance

add(b);add(tf);

setSize(300,300);
}

```

```

setLayout(null);
setVisible(true);
}
public static void main(String args[]){
new AEvent2();
}
}

import java.awt.event.*;
class Outer implements ActionListener{
AEvent2 obj;
Outer(AEvent2 obj){
this.obj=obj;
}
public void actionPerformed(ActionEvent e){
obj.tf.setText("welcome");
}
}

```

---

### 3) Example of event handling by Anonymous class:

```

import java.awt.*;
import java.awt.event.*;
class AEvent3 extends Frame{
TextField tf;
AEvent3(){
tf=new TextField();
tf.setBounds(60,50,170,20);
Button b=new Button("click me");
b.setBounds(50,120,80,30);

b.addActionListener(new ActionListener(){
public void actionPerformed(){
tf.setText("hello");
}
});
add(b);add(tf);
setSize(300,300);
setLayout(null);
setVisible(true);
}
public static void main(String args[]){
new AEvent3();
}
}

```

## Java Swing Tutorial

**Java Swing tutorial** is a part of Java Foundation Classes (JFC) that is *used to create window-based applications*. It is built on the top of AWT (Abstract Windowing Toolkit) API and entirely written in java.

Unlike AWT, Java Swing provides platform-independent and lightweight components.

The javax.swing package provides classes for java swing API such as JButton, JTextField, Jpre, JRadioButton, JCheckbox, JMenu, JColorChooser etc.

---

## Difference between AWT and Swing

There are many differences between java awt and swing that are given below.

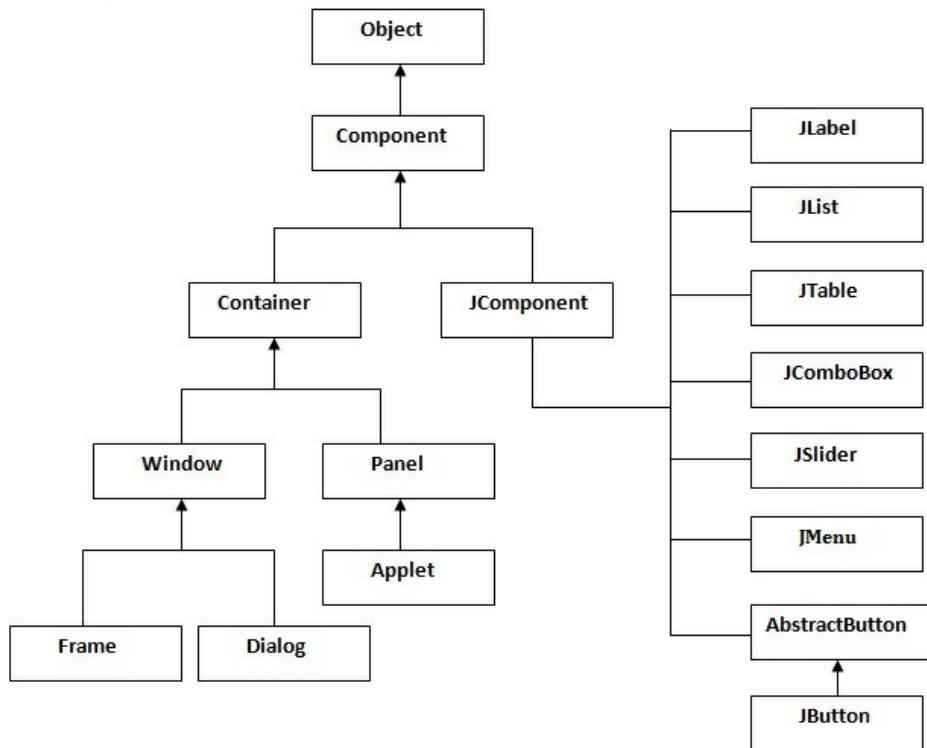
No.	Java AWT	Java Swing
1)	AWT components are <b>platform-dependent</b> .	Java swing components are <b>platform-independent</b> .
2)	AWT components are <b>heavyweight</b> .	Swing components are <b>lightweight</b> .
3)	AWT <b>doesn't support pluggable look and feel</b> .	Swing <b>supports pluggable look and feel</b> .
4)	AWT provides <b>less components</b> than Swing.	Swing provides <b>more powerful components</b> such as tables, lists, scrollpanes, colorchooser, tabbedpane etc.
5)	AWT <b>doesn't follows MVC</b> (Model View Controller) where model represents data, view represents presentation and controller acts as an interface between model and view.	Swing <b>follows MVC</b> .

## What is JFC

The Java Foundation Classes (JFC) are a set of GUI components which simplify the development of desktop applications.

## Hierarchy of Java Swing classes

The hierarchy of java swing API is given below.



## Commonly used Methods of Component class

The methods of Component class are widely used in java swing that are given below.

Method	Description
public void add(Component c)	add a component on another component.
public void setSize(int width,int height)	sets size of the component.
public void setLayout(LayoutManager m)	sets the layout manager for the component.

```
public void setVisible(boolean b)
```

sets the visibility of the component. It is by default false.

## Java Swing Examples

There are two ways to create a frame:

- By creating the object of Frame class (association)
- By extending Frame class (inheritance)

We can write the code of swing inside the main(), constructor or any other method.

### Simple Java Swing Example

Let's see a simple swing example where we are creating one button and adding it on the JFrame object inside the main() method.

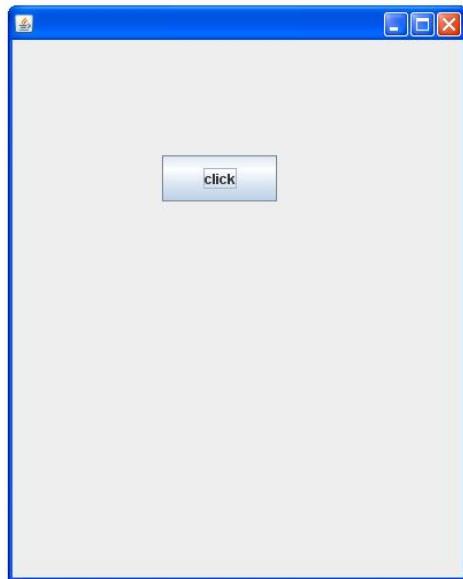
*File: FirstSwingExample.java*

```
import javax.swing.*;
public class FirstSwingExample {
public static void main(String[] args) {
JFrame f=new JFrame(); //creating instance of JFrame

JButton b=new JButton("click"); //creating instance of JButton
b.setBounds(130,100,100, 40); //x axis, y axis, width, height

f.add(b); //adding button in JFrame

f.setSize(400,500); //400 width and 500 height
f.setLayout(null); //using no layout managers
f.setVisible(true); //making the frame visible
}
}
```



### Example of Swing by Association inside constructor

We can also write all the codes of creating JFrame, JButton and method call inside the java constructor.

*File: Simple.java*

```
import javax.swing.*;
public class Simple {
JFrame f;
```

```

Simple(){
f=new JFrame();//creating instance of JFrame

JButton b=new JButton("click");//creating instance of JButton
b.setBounds(130,100,100, 40);

f.add(b);//adding button in JFrame

f.setSize(400,500);//400 width and 500 height
f.setLayout(null);//using no layout managers
f.setVisible(true);//making the frame visible
}

public static void main(String[] args) {
new Simple();
}
}

```

The `setBounds(int xaxis, int yaxis, int width, int height)` is used in the above example that sets the position of the button.

---

## Simple example of Swing by inheritance

We can also inherit the `JFrame` class, so there is no need to create the instance of `JFrame` class explicitly.

*File: Simple2.java*

```

import javax.swing.*;
public class Simple2 extends JFrame{//inheriting JFrame
JFrame f;
Simple2(){
JButton b=new JButton("click");//create button
b.setBounds(130,100,100, 40);

add(b);//adding button on frame
setSize(400,500);
setLayout(null);
setVisible(true);
}
public static void main(String[] args) {
new Simple2();
}}

```

[download this example](#)

## JButton class:

The `JButton` class is used to create a button that have platform-independent implementation.

### Commonly used Constructors:

- **`JButton()`:** creates a button with no text and icon.
- **`JButton(String s)`:** creates a button with the specified text.
- **`JButton(Icon i)`:** creates a button with the specified icon object.

### Commonly used Methods of AbstractButton class:

**1) `public void setText(String s)`:** is used to set specified text on button.

- 2) public String getText():** is used to return the text of the button.
- 3) public void setEnabled(boolean b):** is used to enable or disable the button.
- 4) public void setIcon(Icon b):** is used to set the specified Icon on the button.
- 5) public Icon getIcon():** is used to get the Icon of the button.
- 6) public void setMnemonic(int a):** is used to set the mnemonic on the button.
- 7) public void addActionListener(ActionListener a):** is used to add the action listener to this object.



**Note:** The JButton class extends AbstractButton class.

### Example of displaying image on the button:

```
import java.awt.event.*;
import javax.swing.*;

public class ImageButton{
ImageButton(){
JFrame f=new JFrame();

JButton b=new JButton(new ImageIcon("b.jpg"));
b.setBounds(130,100,100, 40);

f.add(b);

f.setSize(300,400);
f.setLayout(null);
f.setVisible(true);

f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

}

public static void main(String[] args) {
new ImageButton();
}
}
```

[download this example](#)

## JRadioButton class

The JRadioButton class is used to create a radio button. It is used to choose one option from multiple options. It is widely used in exam systems or quiz.

It should be added in ButtonGroup to select one radio button only.

### Commonly used Constructors of JRadioButton class:

- **JRadioButton():** creates an unselected radio button with no text.
- **JRadioButton(String s):** creates an unselected radio button with specified text.
- **JRadioButton(String s, boolean selected):** creates a radio button with the specified text and selected status.

### Commonly used Methods of AbstractButton class:

- 1) public void setText(String s):** is used to set specified text on button.
- 2) public String getText():** is used to return the text of the button.
- 3) public void setEnabled(boolean b):** is used to enable or disable the button.
- 4) public void setIcon(Icon b):** is used to set the specified Icon on the button.
- 5) public Icon getIcon():** is used to get the Icon of the button.
- 6) public void setMnemonic(int a):** is used to set the mnemonic on the button.
- 7) public void addActionListener(ActionListener a):** is used to add the action listener to this object.



**Note:** The JRadioButton class extends the JToggleButton class that extends AbstractButton class.

## Example of JRadioButton class:

```
import javax.swing.*;
public class Radio {
JFrame f;

Radio(){
f=new JFrame();

JRadioButton r1=new JRadioButton("A) Male");
JRadioButton r2=new JRadioButton("B) FeMale");
r1.setBounds(50,100,70,30);
r2.setBounds(50,150,70,30);

ButtonGroup bg=new ButtonGroup();
bg.add(r1);bg.add(r2);

f.add(r1);f.add(r2);

f.setSize(300,300);
f.setLayout(null);
f.setVisible(true);
}

public static void main(String[] args) {
new Radio();
}
```

[download this example](#)

## ButtonGroup class:

The ButtonGroup class can be used to group multiple buttons so that at a time only one button can be selected.

## JRadioButton example with event handling

```
import javax.swing.*;
import java.awt.event.*;
class RadioExample extends JFrame implements ActionListener{
JRadioButton rb1,rb2;
JButton b;
RadioExample(){
rb1=new JRadioButton("Male");
rb1.setBounds(100,50,100,30);
```

```

rb2=new JRadioButton("Female");
rb2.setBounds(100,100,100,30);

ButtonGroup bg=new ButtonGroup();
bg.add(rb1);bg.add(rb2);

b=new JButton("click");
b.setBounds(100,150,80,30);
b.addActionListener(this);

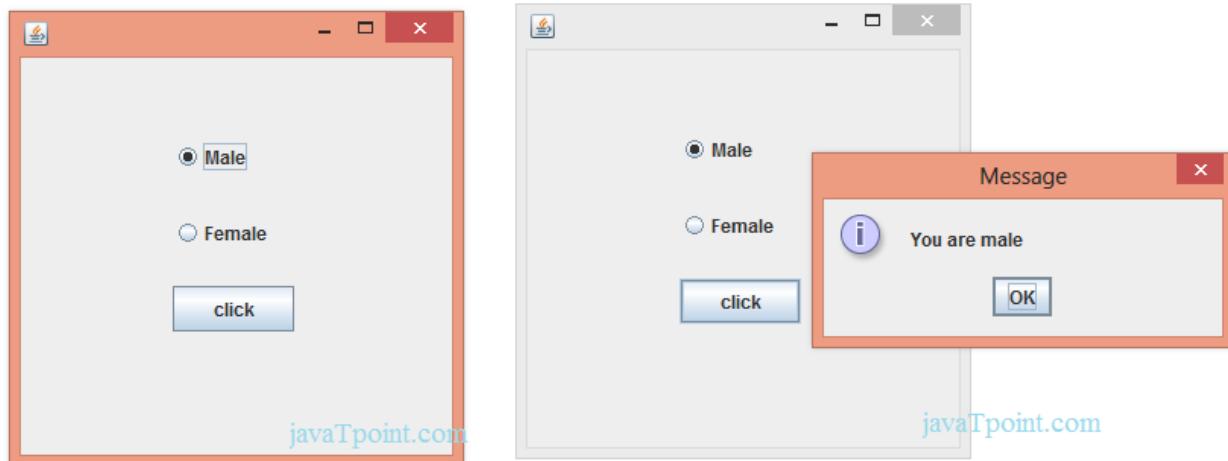
add(rb1);add(rb2);add(b);

setSize(300,300);
setLayout(null);
setVisible(true);
}

public void actionPerformed(ActionEvent e){
if(rb1.isSelected()){
JOptionPane.showMessageDialog(this,"You are male");
}
if(rb2.isSelected()){
JOptionPane.showMessageDialog(this,"You are female");
}
}

public static void main(String args[]){
new RadioExample();
}}

```



## Jpre class (Swing Tutorial):

The **Jpre** class is used to create a text area. It is a multiline area that displays the plain text only.

### Commonly used Constructors:

- **Jpre()**: creates a text area that displays no text initially.
- **Jpre(String s)**: creates a text area that displays specified text initially.
- **Jpre(int row, int column)**: creates a text area with the specified number of rows and columns that displays no text initially..
- **Jpre(String s, int row, int column)**: creates a text area with the specified number of rows and columns that displays specified text.

### Commonly used methods of Jpre class:

- 1) public void setRows(int rows):** is used to set specified number of rows.
- 2) public void setColumns(int cols):** is used to set specified number of columns.

- 3) public void setFont(Font f):** is used to set the specified font.
- 4) public void insert(String s, int position):** is used to insert the specified text on the specified position.
- 5) public void append(String s):** is used to append the given text to the end of the document.

### Example of JTextField class:

```
import java.awt.Color;
import javax.swing.*;

public class TArea {
    JArea area;
    JFrame f;
    TArea(){
        f=new JFrame();
        area=new JArea(300,300);
        area.setBounds(10,30,300,300);

        area.setBackground(Color.black);
        area.setForeground(Color.white);

        f.add(area);

        f.setSize(400,400);
        f.setLayout(null);
        f.setVisible(true);
    }
    public static void main(String[] args) {
        new TArea();
    }
}
```

[download this example](#)

## JComboBox class:

The JComboBox class is used to create the combobox (drop-down list). At a time only one item can be selected from the item list.

### Commonly used Constructors of JComboBox class:

```
JComboBox()
JComboBox(Object[] items)
JComboBox(Vector<?> items)
```

### Commonly used methods of JComboBox class:

- 1) public void addItem(Object anObject):** is used to add an item to the item list.
- 2) public void removeItem(Object anObject):** is used to delete an item to the item list.
- 3) public void removeAllItems():** is used to remove all the items from the list.
- 4) public void setEditable(boolean b):** is used to determine whether the JComboBox is editable.
- 5) public void addActionListener(ActionListener a):** is used to add the ActionListener.
- 6) public void addItemListener(ItemListener i):** is used to add the ItemListener.

## Example of JComboBox class:

```
import javax.swing.*;
public class Combo {
JFrame f;
Combo(){
f=new JFrame("Combo ex");

String country[]={ "India", "Aus", "U.S.A", "England", "Newzeland" };

JComboBox cb=new JComboBox(country);
cb.setBounds(50, 50,90,20);
f.add(cb);

f.setLayout(null);
f.setSize(400,500);
f.setVisible(true);

}
public static void main(String[] args) {
new Combo();

}
}
```

## JTable class (Swing Tutorial):

The JTable class is used to display the data on two dimensional tables of cells.

### Commonly used Constructors of JTable class:

- **JTable():** creates a table with empty cells.
- **JTable(Object[][] rows, Object[] columns):** creates a table with the specified data.

### Example of JTable class:

```
import javax.swing.*;
public class MyTable {
JFrame f;
MyTable(){
f=new JFrame();

String data[][]={ {"101","Amit","670000"},
 {"102","Jai","780000"}, {"101","Sachin","700000"}};
String column[]={ "ID", "NAME", "SALARY" };

JTable jt=new JTable(data,column);
jt.setBounds(30,40,200,300);

JScrollPane sp=new JScrollPane(jt);
f.add(sp);
```

```

f.setSize(300,400);
// f.setLayout(null);
f.setVisible(true);
}
public static void main(String[] args) {
new MyTable();
}
}

```

[download this example](#)

## JColorChooser class:

The JColorChooser class is used to create a color chooser dialog box so that user can select any color.

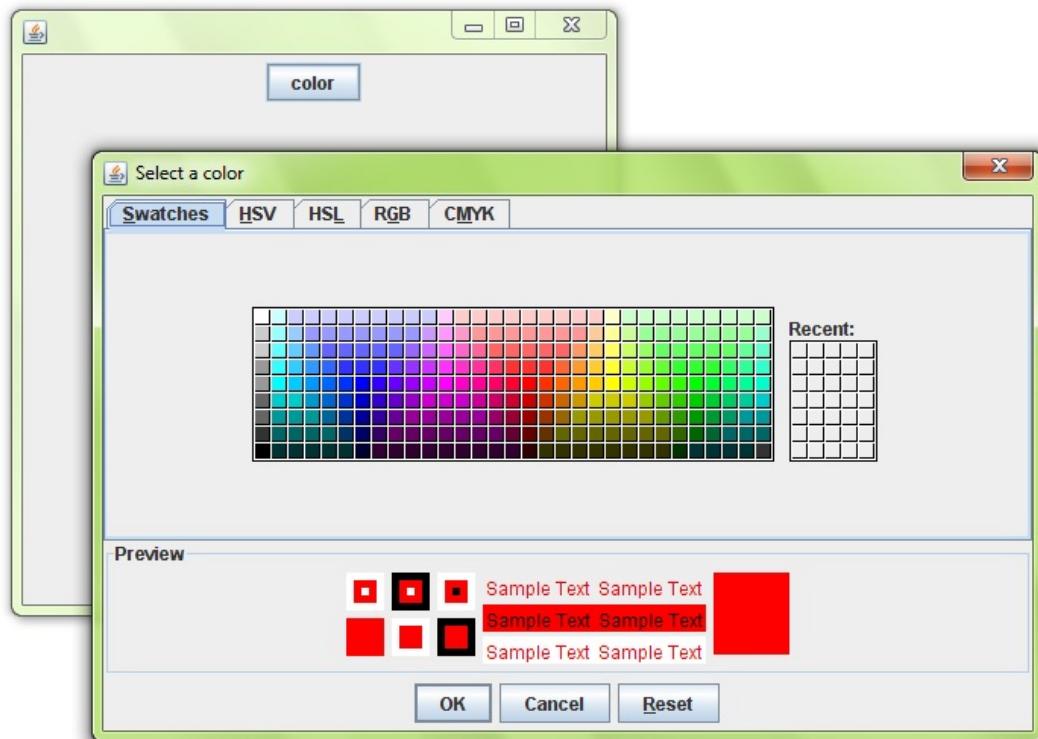
### Commonly used Constructors of JColorChooser class:

- **JColorChooser():** is used to create a color chooser pane with white color initially.
- **JColorChooser(Color initialColor):** is used to create a color chooser pane with the specified color initially.

### Commonly used methods of JColorChooser class:

**public static Color showDialog(Component c, String title, Color initialColor):** is used to show the color-chooser dialog box.

### Example of JColorChooser class:



```

import java.awt.event.*;
import java.awt.*;
import javax.swing.*;

```

```

public class JColorChooserExample extends JFrame implements ActionListener{
JButton b;
Container c;

JColorChooserExample(){
c=getContentPane();
c.setLayout(new FlowLayout());

b=new JButton("color");
b.addActionListener(this);

c.add(b);
}

public void actionPerformed(ActionEvent e) {
Color initialcolor=Color.RED;
Color color=JColorChooser.showDialog(this,"Select a color",initialcolor);
c.setBackground(color);
}

public static void main(String[] args) {
JColorChooserExample ch=new JColorChooserExample();
ch.setSize(400,400);
ch.setVisible(true);
ch.setDefaultCloseOperation(EXIT_ON_CLOSE);
}
}

```

[download this example](#)

## JProgressBar class:

The JProgressBar class is used to display the progress of the task.

### Commonly used Constructors of JProgressBar class:

- **JProgressBar():** is used to create a horizontal progress bar but no string text.
- **JProgressBar(int min, int max):** is used to create a horizontal progress bar with the specified minimum and maximum value.
- **JProgressBar(int orient):** is used to create a progress bar with the specified orientation, it can be either Vertical or Horizontal by using SwingConstants.VERTICAL and SwingConstants.HORIZONTAL constants.
- **JProgressBar(int orient, int min, int max):** is used to create a progress bar with the specified orientation, minimum and maximum value.

### Commonly used methods of JProgressBar class:

- 1) **public void setStringPainted(boolean b):** is used to determine whether string should be displayed.
- 2) **public void setString(String s):** is used to set value to the progress string.
- 3) **public void setOrientation(int orientation):** is used to set the orientation, it may be either vertical or horizontal by using SwingConstants.VERTICAL and SwingConstants.HORIZONTAL constants..
- 4) **public void setValue(int value):** is used to set the current value on the progress bar.

## Example of JProgressBar class:

```

import javax.swing.*;
public class MyProgress extends JFrame{

```

```

JProgressBar jb;
int i=0,num=0;

MyProgress(){
jb=new JProgressBar(0,2000);
jb.setBounds(40,40,200,30);

jb.setValue(0);
jb.setStringPainted(true);

add(jb);
setSize(400,400);
setLayout(null);
}

public void iterate(){
while(i<=2000){
jb.setValue(i);
i=i+20;
try{Thread.sleep(150);}catch(Exception e){}
}
}

public static void main(String[] args) {
MyProgress m=new MyProgress();
m.setVisible(true);
m.iterate();
}
}

```

[download this example](#)

## JSlider class:

The JSlider is used to create the slider. By using JSlider a user can select a value from a specific range.

### Commonly used Constructors of JSlider class:

- **JSlider()**: creates a slider with the initial value of 50 and range of 0 to 100.
- **JSlider(int orientation)**: creates a slider with the specified orientation set by either JSlider.HORIZONTAL or JSlider.VERTICAL with the range 0 to 100 and initial value 50.
- **JSlider(int min, int max)**: creates a horizontal slider using the given min and max.
- **JSlider(int min, int max, int value)**: creates a horizontal slider using the given min, max and value.
- **JSlider(int orientation, int min, int max, int value)**: creates a slider using the given orientation, min, max and value.

### Commonly used Methods of JSlider class:

- 1) **public void setMinorTickSpacing(int n)**: is used to set the minor tick spacing to the slider.
- 2) **public void setMajorTickSpacing(int n)**: is used to set the major tick spacing to the slider.
- 3) **public void setPaintTicks(boolean b)**: is used to determine whether tick marks are painted.
- 4) **public void setPaintLabels(boolean b)**: is used to determine whether labels are painted.
- 5) **public void setPaintTracks(boolean b)**: is used to determine whether track is painted.

### Simple example of JSlider class:



```
import javax.swing.*;  
  
public class SliderExample1 extends JFrame{  
  
    public SliderExample1() {  
        JSlider slider = new JSlider(JSeparator.HORIZONTAL, 0, 50, 25);  
        JPanel panel=new JPanel();  
        panel.add(slider);  
  
        add(panel);  
    }  
  
    public static void main(String s[]) {  
        SliderExample1 frame=new SliderExample1();  
        frame.pack();  
        frame.setVisible(true);  
    }  
}
```

[download this example](#)

---

### Example of JSlider class that paints ticks:

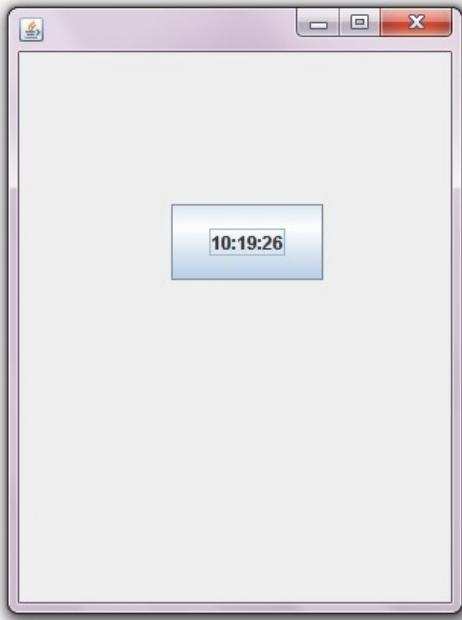


```
import javax.swing.*;  
  
public class SliderExample extends JFrame{  
  
    public SliderExample() {  
  
        JSlider slider = new JSlider(JSeparator.HORIZONTAL, 0, 50, 25);  
        slider.setMinorTickSpacing(2);  
        slider.setMajorTickSpacing(10);  
  
        slider.setPaintTicks(true);  
        slider.setPaintLabels(true);  
  
        JPanel panel=new JPanel();  
        panel.add(slider);  
        add(panel);  
    }  
  
    public static void main(String s[]) {  
        SliderExample frame=new SliderExample();  
        frame.pack();  
        frame.setVisible(true);  
    }  
}
```

```
}
```

[download this example](#)

## Example of digital clock in swing:



```
import javax.swing.*;
import java.awt.*;
import java.text.*;
import java.util.*;
public class DigitalWatch implements Runnable{
JFrame f;
Thread t=null;
int hours=0, minutes=0, seconds=0;
String timeString = "";
JButton b;

DigitalWatch(){
f=new JFrame();

t = new Thread(this);
t.start();

b=new JButton();
b.setBounds(100,100,100,50);

f.add(b);
f.setSize(300,400);
f.setLayout(null);
f.setVisible(true);
}

public void run() {
try {
while (true) {
```

```

        Calendar cal = Calendar.getInstance();
        hours = cal.get( Calendar.HOUR_OF_DAY );
        if ( hours > 12 ) hours -= 12;
        minutes = cal.get( Calendar.MINUTE );
        seconds = cal.get( Calendar.SECOND );

        SimpleDateFormat formatter = new SimpleDateFormat("hh:mm:ss");
        Date date = cal.getTime();
        timeString = formatter.format( date );

        printTime();

        t.sleep( 1000 ); // interval given in milliseconds
    }
}

catch (Exception e) { }

}

public void printTime(){
b.setText(timeString);
}

public static void main(String[] args) {
new DigitalWatch();

}

}

```

[download this example](#)

## Displaying graphics in swing:

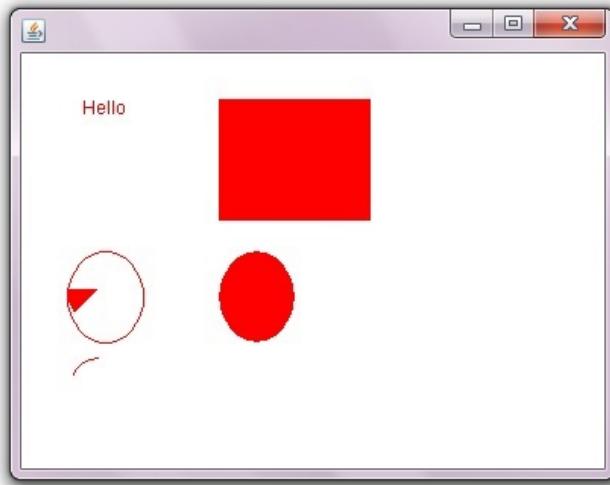
java.awt.Graphics class provides many methods for graphics programming.

### Commonly used methods of Graphics class:

1. **public abstract void drawString(String str, int x, int y):** is used to draw the specified string.
2. **public void drawRect(int x, int y, int width, int height):** draws a rectangle with the specified width and height.
3. **public abstract void fillRect(int x, int y, int width, int height):** is used to fill rectangle with the default color and specified width and height.
4. **public abstract void drawOval(int x, int y, int width, int height):** is used to draw oval with the specified width and height.
5. **public abstract void fillOval(int x, int y, int width, int height):** is used to fill oval with the default color and specified width and height.
6. **public abstract void drawLine(int x1, int y1, int x2, int y2):** is used to draw line between the points(x1, y1) and (x2, y2).
7. **public abstract boolean drawImage(Image img, int x, int y, ImageObserver observer):** is used draw the specified image.
8. **public abstract void drawArc(int x, int y, int width, int height, int startAngle, int arcAngle):** is used draw a circular or elliptical arc.
9. **public abstract void fillArc(int x, int y, int width, int height, int startAngle, int arcAngle):** is used to fill a circular or elliptical arc.
10. **public abstract void setColor(Color c):** is used to set the graphics current color to the specified color.
11. **public abstract void setFont(Font font):** is used to set the graphics current font to the specified font.

---

### Example of displaying graphics in swing:



```
import java.awt.*;
import javax.swing.JFrame;

public class DisplayGraphics extends Canvas{

    public void paint(Graphics g) {
        g.drawString("Hello",40,40);
        setBackground(Color.WHITE);
        g.fillRect(130, 30,100, 80);
        g.drawOval(30,130,50, 60);
        setForeground(Color.RED);
        g.fillOval(130,130,50, 60);
        g.drawArc(30, 200, 40,50,90,60);
        g.fillArc(30, 130, 40,50,180,40);

    }
    public static void main(String[] args) {
        DisplayGraphics m=new DisplayGraphics();
        JFrame f=new JFrame();
        f.add(m);
        f.setSize(400,400);
        //f.setLayout(null);
        f.setVisible(true);
    }
}
```

[download this example](#)

## Displaying image in swing:

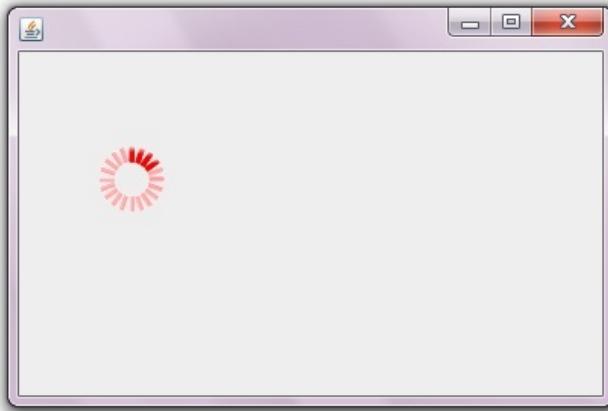
For displaying image, we can use the method `drawImage()` of `Graphics` class.

### Syntax of `drawImage()` method:

1. **public abstract boolean drawImage(Image img, int x, int y, ImageObserver observer):** is used draw the specified image.

---

## Example of displaying image in swing:



```
import java.awt.*;
import javax.swing.JFrame;

public class MyCanvas extends Canvas{

    public void paint(Graphics g) {

        Toolkit t=Toolkit.getDefaultToolkit();
        Image i=t.getImage("p3.gif");
        g.drawImage(i, 120,100,this);

    }
    public static void main(String[] args) {
        MyCanvas m=new MyCanvas();
        JFrame f=new JFrame();
        f.add(m);
        f.setSize(400,400);
        f.setVisible(true);
    }
}
```

[download this example](#)

## Example of creating Edit menu for Notepad:

```
import javax.swing.*;
import java.awt.event.*;

public class Notepad implements ActionListener{
JFrame f;
JMenuBar mb;
JMenu file,edit,help;
JMenuItem cut,copy,paste,selectAll;
JTextArea ta;

Notepad(){

```

```

f=new JFrame();

cut=new JMenuItem("cut");
copy=new JMenuItem("copy");
paste=new JMenuItem("paste");
selectAll=new JMenuItem("selectAll");

cut.addActionListener(this);
copy.addActionListener(this);
paste.addActionListener(this);
selectAll.addActionListener(this);

mb=new JMenuBar();
mb.setBounds(5,5,400,40);

file=new JMenu("File");
edit=new JMenu("Edit");
help=new JMenu("Help");

edit.add(cut);edit.add(copy);edit.add(paste);edit.add(selectAll);

mb.add(file);mb.add(edit);mb.add(help);

ta=new JTextPane();
ta.setBounds(5,30,460,460);

f.add(mb);f.add(ta);

f.setLayout(null);
f.setSize(500,500);
f.setVisible(true);
}

public void actionPerformed(ActionEvent e) {
if(e.getSource()==cut)
ta.cut();
if(e.getSource()==paste)
ta.paste();
if(e.getSource()==copy)
ta.copy();
if(e.getSource()==selectAll)
ta.selectAll();
}

public static void main(String[] args) {
new Notepad();
}
}

```

[download this example](#)

## Example of open dialog box:

```
import java.awt.*;
import javax.swing.*;
import java.awt.event.*;
import java.io.*;

public class OpenMenu extends JFrame implements ActionListener{
JMenuBar mb;
JMenu file;
JMenuItem open;
JTextPane ta;
OpenMenu(){
open=new JMenuItem("Open File");
open.addActionListener(this);

file=new JMenu("File");
file.add(open);

mb=new JMenuBar();
mb.setBounds(0,0,800,20);
mb.add(file);

ta=new JTextPane(800,800);
ta.setBounds(0,20,800,800);

add(mb);
add(ta);

}

public void actionPerformed(ActionEvent e) {
if(e.getSource()==open){
openFile();
}
}

void openFile(){
JFileChooser fc=new JFileChooser();
int i=fc.showOpenDialog(this);

if(i==JFileChooser.APPROVE_OPTION){
File f=fc.getSelectedFile();
String filepath=f.getPath();

displayContent(filepath);
}

}

void displayContent(String fpath){
try{
BufferedReader br=new BufferedReader(new FileReader(fpath));
String s1="",s2="";

while((s1=br.readLine())!=null){
s2+=s1+"\n";
}
ta.setText(s2);
br.close();
}catch (Exception e) {e.printStackTrace(); }
}
```

```

}

public static void main(String[] args) {
    OpenMenu om=new OpenMenu();
    om.setSize(800,800);
    om.setLayout(null);
    om.setVisible(true);
    om.setDefaultCloseOperation(EXIT_ON_CLOSE);
}
}

```

[download this example](#)

## Notepad in Java with source code

**Notepad in Java with source code:** We can develop Notepad in java with the help of AWT/Swing with event handling. Let's see the code of creating Notepad in java.

```

import java.io.*;
import java.util.Date;
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.event.*;

/*****************/
class FileOperation
{
    Notepad npd;

    boolean saved;
    boolean newFileFlag;
    String fileName;
    String applicationTitle="Notepad - JavaTpoint";

    File fileRef;
    JFileChooser chooser;
    /////////////////////////
    boolean isSave(){return saved;}
    void setSave(boolean saved){this.saved=saved;}
    String getFileName(){return new String(fileName);}
    void setFileName(String fileName){this.fileName=new String(fileName);}
    /////////////////////////
    FileOperation(Notepad npd)
    {
        this.npd=npd;

        saved=true;
        newFileFlag=true;
        fileName=new String("Untitled");
        fileRef=new File(fileName);
        this.npd.f.setTitle(fileName+" - "+applicationTitle);

        chooser=new JFileChooser();
        chooser.addChoosableFileFilter(new MyFileFilter(".java","Java Source Files (*.java)"));
        chooser.addChoosableFileFilter(new MyFileFilter(".txt","Text Files (*.txt)"));
        chooser.setCurrentDirectory(new File("."));
    }
}

```

```

}

///////////



boolean saveFile(File temp)
{
FileWriter fout=null;
try
{
fout=new FileWriter(temp);
fout.write(npd.ta.getText());
}
catch(IOException ioe){updateStatus(temp,false);return false;}
finally
{try{fout.close();}catch(IOException excp){}}
updateStatus(temp,true);
return true;
}
///////////
boolean saveThisFile()
{

if(!newFileFlag)
{return saveFile(fileRef);}

return saveAsFile();
}
///////////
boolean saveAsFile()
{
File temp=null;
chooser.setDialogTitle("Save As...");
chooser.setApproveButtonText("Save Now");
chooser.setApproveButtonMnemonic(KeyEvent.VK_S);
chooser.setApproveButtonToolTipText("Click me to save!");

do
{
if(chooser.showSaveDialog(this.npd.f)!=JFileChooser.APPROVE_OPTION)
return false;
temp=chooser.getSelectedFile();
if(!temp.exists()) break;
if(JOptionPane.showConfirmDialog(
this.npd.f,""+temp.getPath()+" already exists.
Do you want to replace it?",
"Save As",JOptionPane.YES_NO_OPTION
)==JOptionPane.YES_OPTION)
break;
}while(true);

return saveFile(temp);
}

///////////
boolean openFile(File temp)
{
FileInputStream fin=null;
BufferedReader din=null;

```

```

try
{
fin=new FileInputStream(temp);
din=new BufferedReader(new InputStreamReader(fin));
String str=" ";
while(str!=null)
{
str=din.readLine();
if(str==null)
break;
this.npd.ta.append(str+"\n");
}

}
catch(IOException ioe){updateStatus(temp,false);return false;}
finally
{try{din.close();fin.close();}catch(IOException excp){}}
updateStatus(temp,true);
this.npd.ta.setCaretPosition(0);
return true;
}
///////////
void openFile()
{
if(!confirmSave()) return;
chooser.setDialogTitle("Open File...");
chooser.setApproveButtonText("Open this");
chooser.setApproveButtonMnemonic(KeyEvent.VK_O);
chooser.setApproveButtonToolTipText("Click me to open the selected file.!");

File temp=null;
do
{
if(chooser.showOpenDialog(this.npd.f)!=JFileChooser.APPROVE_OPTION)
return;
temp=chooser.getSelectedFile();

if(temp.exists()) break;

 JOptionPane.showMessageDialog(this.npd.f,
 ""+temp.getName()+""
file not found.
"+
"Please verify the correct file name was given.",
"Open", JOptionPane.INFORMATION_MESSAGE);

} while(true);

this.npd.ta.setText("");

if(!openFile(temp))
{
fileName="Untitled"; saved=true;
this.npd.f.setTitle(fileName+" - "+applicationTitle);
}
if(!temp.canWrite())
newFileFlag=true;

}
///////////

```

```

void updateStatus(File temp,boolean saved)
{
if(saved)
{
this.saved=true;
fileName=new String(temp.getName());
if(!temp.canWrite())
{fileName+=(Read only); newFileFlag=true;}
fileRef=temp;
npd.f.setTitle(fileName + " - "+applicationTitle);
npd.statusBar.setText("File : "+temp.getPath()+" saved/opened successfully.");
newFileFlag=false;
}
else
{
npd.statusBar.setText("Failed to save/open : "+temp.getPath());
}
}
///////////
boolean confirmSave()
{
String strMsg="The text in the "+fileName+" file has been changed.
"+
"Do you want to save the changes?";
if(!saved)
{
int x=JOptionPane.showConfirmDialog(this.npd.f,strMsg,applicationTitle,
JOptionPane.YES_NO_CANCEL_OPTION);

if(x==JOptionPane.CANCEL_OPTION) return false;
if(x==JOptionPane.YES_OPTION && !saveAsFile()) return false;
}
return true;
}
///////////
void newFile()
{
if(!confirmSave()) return;

this.npd.ta.setText("");
fileName=new String("Untitled");
fileRef=new File(fileName);
saved=true;
newFileFlag=true;
this.npd.f.setTitle(fileName+ " - "+applicationTitle);
}

} // end defination of class FileOperation
/****************************************/
public class Notepad implements ActionListener, MenuConstants
{

JFrame f;
JTextPane ta;
JLabel statusBar;

private String fileName="Untitled";
private boolean saved=true;
String applicationName="Javapad";

```

```

String searchString, replaceString;
int lastSearchIndex;

FileOperation fileHandler;
FontChooser fontDialog=null;
FindDialog findReplaceDialog=null;
JColorChooser bcolorChooser=null;
JColorChooser fcolorChooser=null;
JDialog backgroundDialog=null;
JDialog foregroundDialog=null;
JMenuItem cutItem,copyItem, deleteItem, findItem, findNextItem,
replaceItem, gotoItem, selectAllItem;
/*****************/
Notepad()
{
f=new JFrame(fileName+" - "+applicationName);
ta=new Jpre(30,60);
statusBar=new JLabel("|| Ln 1, Col 1 ",JLabel.RIGHT);
f.add(new JScrollPane(ta),BorderLayout.CENTER);
f.add(statusBar,BorderLayout.SOUTH);
f.add(new JLabel(" "),BorderLayout.EAST);
f.add(new JLabel(" "),BorderLayout.WEST);
createMenuBar(f);
//f.setSize(350,350);
f.pack();
f.setLocation(100,50);
f.setVisible(true);
f.setLocation(150,50);
f.setDefaultCloseOperation(JFrame.DO_NOTHING_ON_CLOSE);

fileHandler=new FileOperation(this);

///////////
ta.addCaretListener(
new CaretListener()
{
public void caretUpdate(CaretEvent e)
{
int lineNumber=0, column=0, pos=0;

try
{
pos=ta.getCaretPosition();
lineNumber=ta.getLineOfOffset(pos);
column=pos-ta.getLineStartOffset(lineNumber);
}catch(Exception excp){}
if(ta.getText().length()==0){lineNumber=0; column=0;}
statusBar.setText("|| Ln "+(lineNumber+1)+", Col "+(column+1));
}
});
///////////
DocumentListener myListener = new DocumentListener()
{
public void changedUpdate(DocumentEvent e){fileHandler.saved=false;}
public void removeUpdate(DocumentEvent e){fileHandler.saved=false;}
public void insertUpdate(DocumentEvent e){fileHandler.saved=false;}
};
ta.getDocument().addDocumentListener(myListener);
/////////

```

```

WindowListener frameClose=new WindowAdapter()
{
public void windowClosing(WindowEvent we)
{
if(fileHandler.confirmSave())System.exit(0);
}
};

f.addWindowListener(frameClose);
///////////
/*
ta.append("Hello dear hello hi");
ta.append("\nwho are u dear mister hello");
ta.append("\nhello bye hel");
ta.append("\nHello");
ta.append("\nMiss u mister hello hell");
fileHandler.saved=true;
*/
}

void goTo()
{
int lineNumber=0;
try
{
lineNumber=ta.getLineOffset(ta.getCaretPosition())+1;
String tempStr=JOptionPane.showInputDialog(f,"Enter Line Number:",""+lineNumber);
if(tempStr==null)
{return;}
lineNumber=Integer.parseInt(tempStr);
ta.setCaretPosition(ta.getLineStartOffset(lineNumber-1));
}catch(Exception e){}
}

public void actionPerformed(ActionEvent ev)
{
String cmdText=ev.getActionCommand();
if(cmdText.equals(fileNew))
fileHandler.newFile();
else if(cmdText.equals(fileOpen))
fileHandler.openFile();
else if(cmdText.equals(fileSave))
fileHandler.saveThisFile();
else if(cmdText.equals(fileSaveAs))
fileHandler.saveAsFile();
else if(cmdText.equals(fileExit))
{if(fileHandler.confirmSave())System.exit(0);}
else if(cmdText.equals(filePrint))
JOptionPane.showMessageDialog(
Notepad.this.f,
"Get ur printer repaired first! It seems u dont have one!",
"Bad Printer",
JOptionPane.INFORMATION_MESSAGE
);
else if(cmdText.equals(editCut))
}

```

```
ta.cut();
///////////
else if(cmdText.equals(editCopy))
ta.copy();
///////////
else if(cmdText.equals(editPaste))
ta.paste();
///////////
else if(cmdText.equals(editDelete))
ta.replaceSelection("");
///////////
else if(cmdText.equals(editFind))
{
if(NotePad.this.ta.getText().length()==0)
return; // text box have no text
if(findReplaceDialog==null)
findReplaceDialog=new FindDialog(NotePad.this.ta);
findReplaceDialog.showDialog(NotePad.this.f,true); //find
}
///////////
else if(cmdText.equals(editFindNext))
{
if(NotePad.this.ta.getText().length()==0)
return; // text box have no text

if(findReplaceDialog==null)
statusBar.setText("Use Find option of Edit Menu first !!!!");
else
findReplaceDialog.findNextWithSelection();
}
///////////
else if(cmdText.equals(editReplace))
{
if(NotePad.this.ta.getText().length()==0)
return; // text box have no text

if(findReplaceDialog==null)
findReplaceDialog=new FindDialog(NotePad.this.ta);
findReplaceDialog.showDialog(NotePad.this.f,false); //replace
}
///////////
else if(cmdText.equals(editGoTo))
{
if(NotePad.this.ta.getText().length()==0)
return; // text box have no text
goTo();
}
///////////
else if(cmdText.equals(editSelectAll))
ta.selectAll();
///////////
else if(cmdText.equals(editTimeDate))
ta.insert(new Date().toString(),ta.getSelectionStart());
///////////
else if(cmdText.equals(formatWordWrap))
{
JCheckBoxMenuItem temp=(JCheckBoxMenuItem)ev.getSource();
ta.setLineWrap(temp.isSelected());
}
///////////
```

```

else if(cmdText.equals(formatFont))
{
if(fontDialog==null)
fontDialog=new FontChooser(ta.getFont());

if(fontDialog.showDialog(Notepad.this.f,"Choose a font"))
Notepad.this.ta.setFont(fontDialog.createFont());
}

///////////////////////
else if(cmdText.equals(formatForeground))
showForegroundColorDialog();
///////////////////////
else if(cmdText.equals(formatBackground))
showBackgroundColorDialog();
///////////////////////

else if(cmdText.equals(viewStatusBar))
{
JCheckBoxMenuItem temp=(JCheckBoxMenuItem)ev.getSource();
statusBar.setVisible(temp.isSelected());
}
///////////////////////
else if(cmdText.equals(helpAboutNotepad))
{
 JOptionPane.showMessageDialog(Notepad.this.f,aboutText,"Dedicated 2 u!",
JOptionPane.INFORMATION_MESSAGE);
}
else
statusBar.setText("This "+cmdText+" command is yet to be implemented");
}//action Performed
///////////////////////
void showBackgroundColorDialog()
{
if(bcolorChooser==null)
bcolorChooser=new JColorChooser();
if(backgroundDialog==null)
backgroundDialog=JColorChooser.createDialog
(Notepad.this.f,
formatBackground,
false,
bcolorChooser,
new ActionListener()
{public void actionPerformed(ActionEvent evvv){
Notepad.this.ta.setBackground(bcolorChooser.getColor());},
null);

backgroundDialog.setVisible(true);
}

///////////////////////
void showForegroundColorDialog()
{
if(fcolorChooser==null)
fcolorChooser=new JColorChooser();
if(foregroundDialog==null)
foregroundDialog=JColorChooser.createDialog
(Notepad.this.f,
formatForeground,
false,
fcolorChooser,
new ActionListener()

```

```

{public void actionPerformed(ActionEvent evvv){
    Notepad.this.ta.setForeground(fcolorChooser.getColor());}},  

    null);  
  

foregroundDialog.setVisible(true);  

}  
  

//////////  

JMenuItem createMenuItem(String s, int key,JMenu toMenu,ActionListener al)  

{  

JMenuItem temp=new JMenuItem(s,key);  

temp.addActionListener(al);  

toMenu.add(temp);  
  

return temp;  

}  

//////////  

JMenuItem createMenuItem(String s, int key,JMenu toMenu,int aclKey,ActionListener al)  

{  

JMenuItem temp=new JMenuItem(s,key);  

temp.addActionListener(al);  

temp.setAccelerator(KeyStroke.getKeyStroke(aclKey,ActionEvent.CTRL_MASK));  

toMenu.add(temp);  
  

return temp;  

}  

//////////  

JCheckBoxMenuItem createCheckBoxMenuItem(String s,  

    int key,JMenu toMenu,ActionListener al)  

{  

JCheckBoxMenuItem temp=new JCheckBoxMenuItem(s);  

temp.setMnemonic(key);  

temp.addActionListener(al);  

temp.setSelected(false);  

toMenu.add(temp);  
  

return temp;  

}  

//////////  

JMenu createMenu(String s,int key,JMenuBar toMenuBar)  

{  

JMenu temp=new JMenu(s);  

temp.setMnemonic(key);  

toMenuBar.add(temp);  

return temp;  

}  

*****  

void createMenuBar(JFrame f)  

{  

JMenuBar mb=new JMenuBar();  

JMenuItem temp;  
  

JMenu fileMenu=createMenu(fileText,KeyEvent.VK_F,mb);  

JMenu editMenu=createMenu(editText,KeyEvent.VK_E,mb);  

JMenu formatMenu=createMenu(formatText,KeyEvent.VK_O,mb);  

JMenu viewMenu=createMenu(viewText,KeyEvent.VK_V,mb);  

JMenu helpMenu=createMenu(helpText,KeyEvent.VK_H,mb);  
  

createMenuItem(fileNew,KeyEvent.VK_N,fileMenu,KeyEvent.VK_N,this);  

createMenuItem(fileOpen,KeyEvent.VK_O,fileMenu,KeyEvent.VK_O,this);

```

```

createMenuItem(fileSave,KeyEvent.VK_S,fileMenu,KeyEvent.VK_S,this);
createMenuItem(fileSaveAs,KeyEvent.VK_A,fileMenu,this);
fileMenu.addSeparator();
temp=createMenuItem(filePageSetup,KeyEvent.VK_U,fileMenu,this);
temp.setEnabled(false);
createMenuItem(filePrint,KeyEvent.VK_P,fileMenu,KeyEvent.VK_P,this);
fileMenu.addSeparator();
createMenuItem(fileExit,KeyEvent.VK_X,fileMenu,this);

temp=createMenuItem(editUndo,KeyEvent.VK_U,editMenu,KeyEvent.VK_Z,this);
temp.setEnabled(false);
editMenu.addSeparator();
cutItem=createMenuItem(editCut,KeyEvent.VK_T,editMenu,KeyEvent.VK_X,this);
copyItem=createMenuItem(editCopy,KeyEvent.VK_C,editMenu,KeyEvent.VK_C,this);
createMenuItem(editPaste,KeyEvent.VK_P,editMenu,KeyEvent.VK_V,this);
deleteItem=createMenuItem(editDelete,KeyEvent.VK_L,editMenu,this);
deleteItem.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_DELETE,0));
editMenu.addSeparator();
findItem=createMenuItem(editFind,KeyEvent.VK_F,editMenu,KeyEvent.VK_F,this);
findNextItem=createMenuItem(editFindNext,KeyEvent.VK_N,editMenu,this);
findNextItem.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_F3,0));
replaceItem=createMenuItem(editReplace,KeyEvent.VK_R,editMenu,KeyEvent.VK_H,this);
gotoItem=createMenuItem(editGoTo,KeyEvent.VK_G,editMenu,KeyEvent.VK_G,this);
editMenu.addSeparator();
selectAllItem=createMenuItem(editSelectAll,KeyEvent.VK_A,editMenu,KeyEvent.VK_A,this);
createMenuItem(editTimeDate,KeyEvent.VK_D,editMenu,this)
.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_F5,0));

createCheckBoxMenuItem(formatWordWrap,KeyEvent.VK_W,formatMenu,this);

createMenuItem(formatFont,KeyEvent.VK_F,formatMenu,this);
formatMenu.addSeparator();
createMenuItem(formatForeground,KeyEvent.VK_T,formatMenu,this);
createMenuItem(formatBackground,KeyEvent.VK_P,formatMenu,this);

createCheckBoxMenuItem(viewStatusBar,KeyEvent.VK_S,viewMenu,this).setSelected(true);
*****For Look and Feel***/
LookAndFeelMenu.createLookAndFeelMenuItem(viewMenu,this.f);

temp=createMenuItem(helpHelpTopic,KeyEvent.VK_H,helpMenu,this);
temp.setEnabled(false);
helpMenu.addSeparator();
createMenuItem(helpAboutNotepad,KeyEvent.VK_A,helpMenu,this);

MenuListener editMenuListener=new MenuListener()
{
    public void menuSelected(MenuEvent evvvv)
    {
if(Notepad.this.ta.getText().length()==0)
{
    findItem.setEnabled(false);
    findNextItem.setEnabled(false);
    replaceItem.setEnabled(false);
    selectAllItem.setEnabled(false);
    gotoItem.setEnabled(false);
}
else
{
    findItem.setEnabled(true);
}
    }
}

```

```
findNextItem.setEnabled(true);
replaceItem.setEnabled(true);
selectAllItem.setEnabled(true);
gotoItem.setEnabled(true);
}
if(Notepad.this.ta.getSelectionStart()==ta.getSelectionEnd())
{
cutItem.setEnabled(false);
copyItem.setEnabled(false);
deleteItem.setEnabled(false);
}
else
{
cutItem.setEnabled(true);
copyItem.setEnabled(true);
deleteItem.setEnabled(true);
}
}

public void menuDeselected(MenuEvent evvvv){}
public void menuCanceled(MenuEvent evvvv){}
};

editMenu.addMenuListener(editMenuListener);
f.setJMenuBar(mb);
}

*****Constructor*****
///////////////////////////////
public static void main(String[] s)
{
new Notepad();
}

//public
interface MenuConstants
{
final String fileText="File";
final String editText="Edit";
final String formatText="Format";
final String viewText="View";
final String helpText="Help";

final String fileNew="New";
final String fileOpen="Open...";
final String fileSave="Save";
final String fileSaveAs="Save As...";
final String filePageSetup="Page Setup...";
final String filePrint="Print";
final String fileExit="Exit";

final String editUndo="Undo";
final String editCut="Cut";
final String editCopy="Copy";
final String editPaste="Paste";
final String editDelete="Delete";
final String editFind="Find...";
final String editFindNext="Find Next";
final String editReplace="Replace";
final String editGoTo="Go To...";
final String editSelectAll="Select All";
final String editTimeDate="Time/Date";
```

```

final String formatWordWrap="Word Wrap";
final String formatFont="Font...";
final String formatForeground="Set Text color...";
final String formatBackground="Set Pad color...";

final String viewStatusBar="Status Bar";

final String helpHelpTopic="Help Topic";
final String helpAboutNotepad="About Javapad";

final String aboutText="Your Javapad";
}

```

[download this example](#)

## Calculator in Java with Source Code

**Calculator in Java with Source Code:** We can develop calculator in java with the help of AWT/Swing with event handling. Let's see the code of creating calculator in java.

```

*****
Save this file as MyCalculator.java
to compile it use
javac MyCalculator.java
to use the calcuator do this
java MyCalculator

*****
import java.awt.*;
import java.awt.event.*;
*****

public class MyCalculator extends Frame
{
    public boolean setClear=true;
    double number, memValue;
    char op;

    String digitButtonText[] = {"7", "8", "9", "4", "5", "6", "1", "2", "3", "0", "+/-", "." };
    String operatorButtonText[] = {"/", "sqrt", "*", "%", "-", "1/X", "+", "=" };
    String memoryButtonText[] = {"MC", "MR", "MS", "M+" };
    String specialButtonText[] = {"Backspc", "C", "CE" };

    MyDigitButton digitButton[ ]=new MyDigitButton[digitButtonText.length];
    MyOperatorButton operatorButton[ ]=new MyOperatorButton[operatorButtonText.length];
    MyMemoryButton memoryButton[ ]=new MyMemoryButton[memoryButtonText.length];
    MySpecialButton specialButton[ ]=new MySpecialButton[specialButtonText.length];

    Label displayLabel=new Label("0",Label.RIGHT);
    Label memLabel=new Label(" ",Label.RIGHT);

```

```

final int FRAME_WIDTH=325,FRAME_HEIGHT=325;
final int HEIGHT=30, WIDTH=30, H_SPACE=10,V_SPACE=10;
final int TOPX=30, TOPY=50;
///////////////////////////////
MyCalculator(String frameText)//constructor
{
super(frameText);

int tempX=TOPX, y=TOPY;
displayLabel.setBounds(tempX,y,240,HEIGHT);
displayLabel.setBackground(Color.BLUE);
displayLabel.setForeground(Color.WHITE);
add(displayLabel);

memLabel.setBounds(TOPX, TOPY+HEIGHT+ V_SPACE,WIDTH, HEIGHT);
add(memLabel);

// set Co-ordinates for Memory Buttons
tempX=TOPX;
y=TOPY+2*(HEIGHT+V_SPACE);
for(int i=0; i<0)
resText=resText.substring(0,resText.length()-2);
return resText;
}
///////////////////////////////
public static void main(String []args)
{
new MyCalculator("Calculator - JavaTpoint");
}
}

*****



class MyDigitButton extends Button implements ActionListener
{
MyCalculator cl;

///////////////////////////////
MyDigitButton(int x,int y, int width,int height,String cap, MyCalculator clc)
{
super(cap);
setBounds(x,y,width,height);
this.cl=clc;
this.cl.add(this);
addActionListener(this);
}
///////////////////////////////
static boolean isInString(String s, char ch)
{
for(int i=0; i<s.length(); i++)
if(s.charAt(i)==ch)
return true;
return false;
}
}

```

[download this example](#)

## IP Finder in Java with Source Code

**IP Finder in Java with Source Code:** We can develop IP Finder in java with the help of Networking, AWT/Swing with event handling. Let's

see the code of creating IP Finder in java.

```
String url="www.javatpoint.com";
InetAddress ia=InetAddress.getByName(url);
String ip=ia.getHostAddress();

Let's see the Swing code to find IP address.

import javax.swing.*;
import java.awt.event.*;
import java.net.*;
public class IPFinder extends JFrame implements ActionListener{
JLabel l;
JTextField tf;
JButton b;
IPFinder(){
super("IP Finder Tool - Javatpoint");
l=new JLabel("Enter URL:");
l.setBounds(50,70,150,20);
tf=new JTextField();
tf.setBounds(50,100,200,20);

b=new JButton("Find IP");
b.setBounds(50,150,80,30);
b.addActionListener(this);
add(l);
add(tf);
add(b);
setSize(300,300);
setLayout(null);
setVisible(true);
}
public void actionPerformed(ActionEvent e){
String url=tf.getText();
try {
InetAddress ia=InetAddress.getByName(url);
String ip=ia.getHostAddress();
 JOptionPane.showMessageDialog(this,ip);
} catch (UnknownHostException e1) {
JOptionPane.showMessageDialog(this,e1.toString());
}
}
public static void main(String[] args) {
new IPFinder();
}
}
```

## Word Character Counter in Java with Source Code

**Word Character Counter in Java with Source Code:** We can develop Word Character Counter in java with the help of string, AWT/Swing with event handling. Let's see the code of creating Word Character Counter in java.

```
String text="hello javatpoint this is wcc tool";
String words[]=text.split("\\s");
int length=words.length;//returns total number of words
int clength=text.length();//returns total number of characters with space
```

Let's see the swing code to count word and character.

```

import java.awt.event.*;
import javax.swing.*;
public class WCC extends JFrame implements ActionListener{
Jpre ta;
JButton b1,b2;
WCC(){
super("Word Character Counter - JavaTpoint");
ta=new Jpre();
ta.setBounds(50,50,300,200);

b1=new JButton("Word");
b1.setBounds(50,300,100,30);

b2=new JButton("Character");
b2.setBounds(180,300,100,30);

b1.addActionListener(this);
b2.addActionListener(this);
add(b1);add(b2);add(ta);
setSize(400,400);
setLayout(null);
setVisible(true);
}
public void actionPerformed(ActionEvent e){
String text=ta.getText();
if(e.getSource()==b1){
String words[] =text.split("\\s");
 JOptionPane.showMessageDialog(this,"Total words: "+words.length());
}
if(e.getSource()==b2){
 JOptionPane.showMessageDialog(this,"Total Characters with space: "+text.length());
}
}
public static void main(String[] args) {
new WCC();
}
}

```

## URL Source Code Generator in Java with Source Code

**URL Source Code Generator in Java with Source Code:** We can develop URL Source Code Generator in java with the help of networking, AWT/Swing with event handling. Let's see the code of creating URL Source Code Generator in java.

```

URL u=new URL("https://www.facebook.com");//change the URL
URLConnection uc=u.openConnection();
InputStream is=uc.getInputStream();
int i;
StringBuilder sb=new StringBuilder();
while((i=is.read())!=-1){
sb.append((char)i);
}
String source=sb.toString();

```

Let's see the swing code to generate Source Code of URL.

```

import java.awt.*;
import java.awt.event.*;

```

```

import java.io.InputStream;
import java.net.*;
public class SourceGetter extends Frame implements ActionListener{
    TextField tf;
    pre ta;
    Button b;
    Label l;
    SourceGetter(){
        super("Source Getter Tool - Javatpoint");
        l=new Label("Enter URL:");
        l.setBounds(50,50,50,20);

        tf=new TextField();
        tf.setBounds(120,50,250,20);

        b=new Button("Get Source Code");
        b.setBounds(120, 100,120,30);
        b.addActionListener(this);

        ta=new pre();
        ta.setBounds(120,150,250,150);

        add(l);add(tf);add(b);add(ta);
        setSize(400,400);
        setLayout(null);
        setVisible(true);
    }
    public void actionPerformed(ActionEvent e){
        String s=tf.getText();
        if(s==null){}
        else{
            try{
                URL u=new URL(s);
                URLConnection uc=u.openConnection();

                InputStream is=uc.getInputStream();
                int i;
                StringBuilder sb=new StringBuilder();
                while((i=is.read())!=-1){
                    sb.append((char)i);
                }
                String source=sb.toString();
                ta.setText(source);
            }catch(Exception ex){System.out.println(e);}
        }
    }
    public static void main(String[] args) {
        new SourceGetter();
    }
}

```

## Folder Explorer in Java with Source Code

**Folder Explorer in Java with Source Code:** We can develop Folder Explorer in java with the help of IO Stream, AWT/Swing with event handling. Let's see the code of creating Folder Explorer in java.

```
import java.io.*;
```

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.tree.*;
/*****************/
class Explorer extends JPanel implements ActionListener
{
JTextField jtf;
Jpre jta;
JTree tree;
JButton refresh;
JTable jtb;
JScrollPane jsp;
JScrollPane jspTable;

String currDirectory=null;

final String[] colHeads={"File Name","SIZE(in Bytes)","Read Only","Hidden"};
String[][]data={{","","","","","",""}};

///////////////////////
Explorer(String path)
{

jtf=new JTextField();
jta=new Jpre(5,30);
refresh=new JButton("Refresh");

File temp=new File(path);
DefaultMutableTreeNode top=createTree(temp);

//if(top!=null)

tree=new JTree(top);

jsp=new JScrollPane(tree);

final String[] colHeads={"File Name","SIZE(in Bytes)","Read Only","Hidden"};
String[][]data={{","","","","","",""}};
jtb=new JTable(data, colHeads);
jspTable=new JScrollPane(jtb);

setLayout(new BorderLayout());
add(jtf,BorderLayout.NORTH);
add(jsp,BorderLayout.WEST);
add(jspTable,BorderLayout.CENTER);
add(refresh,BorderLayout.SOUTH);

tree.addMouseListener(
new MouseAdapter()
{
public void mouseClicked(MouseEvent me)
{
doMouseClicked(me);
}
});
jtf.addActionListener(this);
refresh.addActionListener(this);
}

///////////////////////

```

```

public void actionPerformed(ActionEvent ev)
{
File temp=new File(jtf.getText());
DefaultMutableTreeNode newtop=createTree(temp);
if(newtop!=null)
tree=new JTree(newtop);
remove(jsp);
jsp=new JScrollPane(tree);
setVisible(false);
add(jsp,BorderLayout.WEST);
tree.addMouseListener(
new MouseAdapter()
{
public void mouseClicked(MouseEvent me)
{
doMouseClicked(me);
}
});
setVisible(true);
}
///////////
DefaultMutableTreeNode createTree(File temp)
{
DefaultMutableTreeNode top=new DefaultMutableTreeNode(temp.getPath());
if(!(temp.exists() && temp.isDirectory()))
return top;

fillTree(top,temp.getPath());

return top;
}
///////////
void fillTree(DefaultMutableTreeNode root, String filename)
{
File temp=new File(filename);

if(!(temp.exists() && temp.isDirectory()))
return;
//System.out.println(filename);
File[] filelist=temp.listFiles();

for(int i=0; i

```

[download this example](#)

## Puzzle Game in Java

**Puzzle Game in Java with Source Code:** We can develop Puzzle Game in java with the help of AWT/Swing with event handling. Let's see the code of creating Puzzle Game in java.



```
import java.awt.*;
import javax.swing.*;
import java.awt.event.*;
public class puzzle extends JFrame implements ActionListener{
JButton b1,b2,b3,b4,b5,b6,b7,b8,b9,next;
puzzle(){
super("Puzzle Game - JavaTpoint");
b1=new JButton("1");
b2=new JButton(" ");
b3=new JButton("3");
b4=new JButton("4");
b5=new JButton("5");
b6=new JButton("6");
b7=new JButton("7");
b8=new JButton("8");
b9=new JButton("2");
next=new JButton("next");

b1.setBounds(10,30,50,40);
b2.setBounds(70,30,50,40);
b3.setBounds(130,30,50,40);
b4.setBounds(10,80,50,40);
b5.setBounds(70,80,50,40);
b6.setBounds(130,80,50,40);
b7.setBounds(10,130,50,40);
b8.setBounds(70,130,50,40);
b9.setBounds(130,130,50,40);
next.setBounds(70,200,100,40);

add(b1);add(b2);add(b3);add(b4);add(b5);add(b6);add(b7);add(b8);add(b9); add(next);
b1.addActionListener(this);
b2.addActionListener(this);
b3.addActionListener(this);
b4.addActionListener(this);
b5.addActionListener(this);
b6.addActionListener(this);
b7.addActionListener(this);
b8.addActionListener(this);
b9.addActionListener(this);
next.addActionListener(this);

next.setBackground(Color.black);
```

```

next.setForeground(Color.green);
setSize(250,300);
setLayout(null);
setVisible(true);
setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
}//end of constructor

public void actionPerformed(ActionEvent e){
if(e.getSource()==next){
String s=b4.getLabel();
b4.setLabel(b9.getLabel());
b9.setLabel(s);
s=b1.getLabel();
b1.setLabel(b5.getLabel());
b5.setLabel(s);
s=b2.getLabel();
b2.setLabel(b7.getLabel());
b7.setLabel(s);
}
if(e.getSource()==b1){
String s=b1.getLabel();
if(b2.getLabel().equals(" ")){ b2.setLabel(s); b1.setLabel(" ");}
else if(b4.getLabel().equals(" ")){ b4.setLabel(s); b1.setLabel(" ");}
}//end of if

if(e.getSource()==b3){
String s=b3.getLabel();
if(b2.getLabel().equals(" ")){ b2.setLabel(s); b3.setLabel(" ");}
else if(b6.getLabel().equals(" ")){ b6.setLabel(s); b3.setLabel(" ");}
}//end of if

if(e.getSource()==b2){
String s=b2.getLabel();
if(b1.getLabel().equals(" ")){ b1.setLabel(s); b2.setLabel(" ");}
else if(b3.getLabel().equals(" ")){ b3.setLabel(s); b2.setLabel(" ");}
else if(b5.getLabel().equals(" ")){ b5.setLabel(s); b2.setLabel(" ");}
}//end of if

if(e.getSource()==b4){
String s=b4.getLabel();
if(b1.getLabel().equals(" ")){ b1.setLabel(s); b4.setLabel(" ");}
else if(b7.getLabel().equals(" ")){ b7.setLabel(s); b4.setLabel(" ");}
else if(b5.getLabel().equals(" ")){ b5.setLabel(s); b4.setLabel(" ");}
}//end of if

if(e.getSource()==b5){
String s=b5.getLabel();
if(b2.getLabel().equals(" ")){ b2.setLabel(s); b5.setLabel(" ");}
else if(b4.getLabel().equals(" ")){ b4.setLabel(s); b5.setLabel(" ");}
else if(b6.getLabel().equals(" ")){ b6.setLabel(s); b5.setLabel(" ");}
else if(b8.getLabel().equals(" ")){ b8.setLabel(s); b5.setLabel(" ");}
}//end of if

if(e.getSource()==b6){
String s=b6.getLabel();
if(b9.getLabel().equals(" ")){ b9.setLabel(s); b6.setLabel(" ");}
else if(b3.getLabel().equals(" ")){ b3.setLabel(s); b6.setLabel(" ");}
else if(b5.getLabel().equals(" ")){ b5.setLabel(s); b6.setLabel(" ");}
}

```

```

} //end of if

if(e.getSource()==b7){
String s=b7.getLabel();
if(b4.getLabel().equals(" ")){ b4.setLabel(s); b7.setLabel(" ");}
else if(b8.getLabel().equals(" ")){ b8.setLabel(s); b7.setLabel(" ");}

}//end of if

if(e.getSource()==b8){
String s=b8.getLabel();
if(b7.getLabel().equals(" ")){ b7.setLabel(s); b8.setLabel(" ");}
else if(b9.getLabel().equals(" ")){ b9.setLabel(s); b8.setLabel(" ");}
else if(b5.getLabel().equals(" ")){ b5.setLabel(s); b8.setLabel(" ");}

}//end of if

if(e.getSource()==b9){
String s=b9.getLabel();
if(b6.getLabel().equals(" ")){ b6.setLabel(s); b9.setLabel(" ");}
else if(b8.getLabel().equals(" ")){ b8.setLabel(s); b9.setLabel(" ");}
if(b1.getLabel().equals("1")&&b2.getLabel();().equals("2")&&b3.getLabel();()
>equals("3")&&b4.getLabel();().equals("4")&&b5.getLabel();().equals("5")
&&b6.getLabel();().equals("6")&&b7.getLabel();().equals("7")&&b8.getLabel();()
.equals("8")&&b9.getLabel();().equals(" ")){
JOptionPane.showMessageDialog(puzzle.this,"!!!you won!!!");
}
}//end of if

}//end of actionPerformed

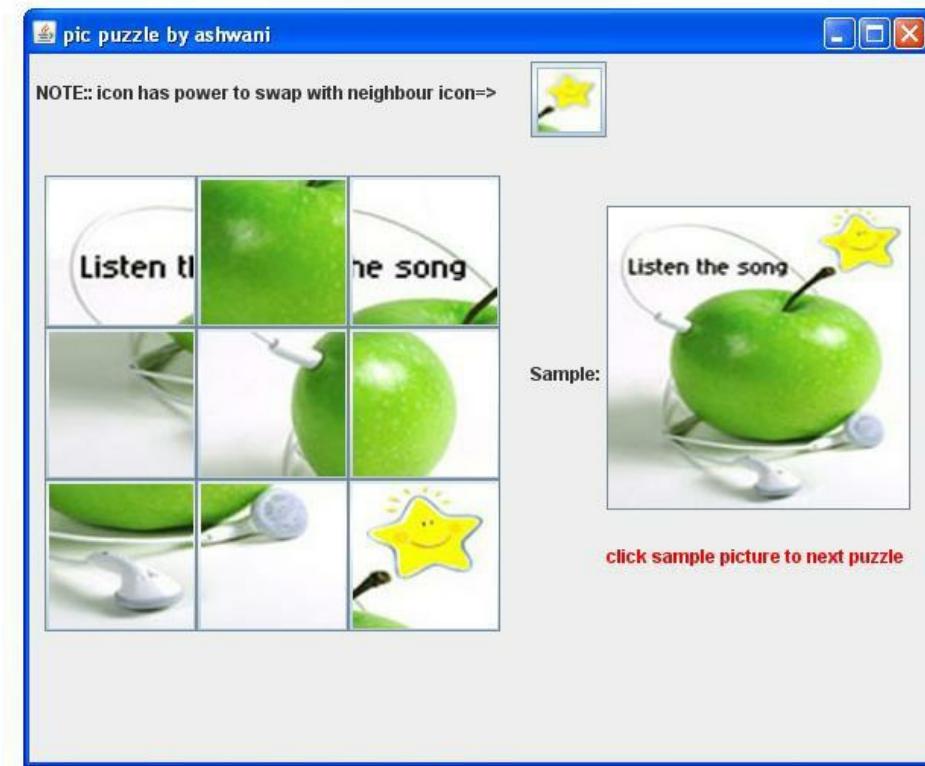
public static void main(String[] args){
new puzzle();
}//end of main

}//end of class

```

[download this example](#)

## Example of Pic Puzzle Game



```

import java.awt.event.*;
import java.awt.*;
import javax.swing.*;
class picpuzzle2 extends JFrame implements ActionListener{
JButton b1,b2,b3,b4,b5,b6,b7,b8,b9,sample,starB;
Icon star;
Icon ic0=new ImageIcon("pic/starB0.jpg");
Icon ic10=new ImageIcon("pic/starB10.jpg");
Icon ic20=new ImageIcon("pic/starB20.jpg");
Icon samicon1=new ImageIcon("pic/main.jpg");
Icon samicon2=new ImageIcon("pic/main2.jpg");
Icon samicon3=new ImageIcon("pic/main3.jpg");
Icon ic1=new ImageIcon("pic/1.jpg");
Icon ic2=new ImageIcon("pic/5.jpg");
Icon ic3=new ImageIcon("pic/2.jpg");
Icon ic4=new ImageIcon("pic/7.jpg");
Icon ic5=new ImageIcon("pic/4.jpg");
Icon ic6=new ImageIcon("pic/6.jpg");
Icon ic7=new ImageIcon("pic/8.jpg");
Icon ic8=new ImageIcon("pic/9.jpg");
Icon ic9=new ImageIcon("pic/3.jpg");

Icon ic11=new ImageIcon("pic/12.jpg");
Icon ic12=new ImageIcon("pic/13.jpg");
Icon ic13=new ImageIcon("pic/16.jpg");
Icon ic14=new ImageIcon("pic/11.jpg");
Icon ic15=new ImageIcon("pic/14.jpg");
Icon ic16=new ImageIcon("pic/19.jpg");
Icon ic17=new ImageIcon("pic/17.jpg");
Icon ic18=new ImageIcon("pic/15.jpg");
Icon ic19=new ImageIcon("pic/18.jpg");

Icon ic21=new ImageIcon("pic/24.jpg");
Icon ic22=new ImageIcon("pic/25.jpg");
Icon ic23=new ImageIcon("pic/21.jpg");

```

```

Icon ic24=new ImageIcon("pic/27.jpg");
Icon ic25=new ImageIcon("pic/23.jpg");
Icon ic26=new ImageIcon("pic/29.jpg");
Icon ic27=new ImageIcon("pic/28.jpg");
Icon ic28=new ImageIcon("pic/22.jpg");
Icon ic29=new ImageIcon("pic/26.jpg");

picpuzzle2(){

super("pic puzzle");

b1=new JButton(ic1);
b1.setBounds(10,80,100,100);
b2=new JButton(ic2);
b2.setBounds(110,80,100,100);
b3=new JButton(ic3);
b3.setBounds(210,80,100,100);
b4=new JButton(ic4);
b4.setBounds(10,180,100,100);
b5=new JButton(ic5);
b5.setBounds(110,180,100,100);
b6=new JButton(ic6);
b6.setBounds(210,180,100,100);
b7=new JButton(ic7);
b7.setBounds(10,280,100,100);
b8=new JButton(ic8);
b8.setBounds(110,280,100,100);
b9=new JButton(ic9);
b9.setBounds(210,280,100,100);
sample=new JButton(samicon1);
sample.setBounds(380,100,200,200);

JLabel l1=new JLabel("Sample:");
l1.setBounds(330,200,70,20);
JLabel l2=new JLabel("NOTE:
icon has power to swap with neighbour icon=");
l2.setBounds(5,15,500,20);
JLabel l3=new JLabel("click sample picture to next puzzle");
l3.setBounds(380,320,200,20);
l3.setForeground(Color.red);

starB=new JButton(ic0);
starB.setBounds(330,5,50,50);
star=starB.getIcon();

add(b1);add(b2);add(b3);add(b4);add(b5);add(b6);add(b7);add(b8);
add(b9);add(sample);add(l1);add(l2);add(starB);add(l3);
b1.addActionListener(this); b2.addActionListener(this);
b3.addActionListener(this); b4.addActionListener(this);
b5.addActionListener(this); b6.addActionListener(this);
b7.addActionListener(this); b8.addActionListener(this);
b9.addActionListener(this);

sample.addActionListener(this);
setLayout(null);
setSize(600,500);
setVisible(true);
setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
}

```

```

public void actionPerformed(ActionEvent e){
    if(e.getSource()==b1){
        Icon s1=b1.getIcon();
        if(b2.getIcon()==star){
            b2.setIcon(s1);
            b1.setIcon(star);
        } else if(b4.getIcon()==star){
            b4.setIcon(s1);
            b1.setIcon(star);
        }
    }//end of if

    if(e.getSource()==b2){
        Icon s1=b2.getIcon();
        if(b1.getIcon()==star){
            b1.setIcon(s1);
            b2.setIcon(star);
        } else if(b5.getIcon()==star){
            b5.setIcon(s1);
            b2.setIcon(star);
        }
        else if(b3.getIcon()==star){
            b3.setIcon(s1);
            b2.setIcon(star);
        }
    }//end of if

    if(e.getSource()==b3){
        Icon s1=b3.getIcon();
        if(b2.getIcon()==star){
            b2.setIcon(s1);
            b3.setIcon(star);
        } else if(b6.getIcon()==star){
            b6.setIcon(s1);
            b3.setIcon(star);
        }
    }//end of if

    if(e.getSource()==b4){
        Icon s1=b4.getIcon();
        if(b1.getIcon()==star){
            b1.setIcon(s1);
            b4.setIcon(star);
        } else if(b5.getIcon()==star){
            b5.setIcon(s1);
            b4.setIcon(star);
        }
        else if(b7.getIcon()==star){
            b7.setIcon(s1);
            b4.setIcon(star);
        }
    }//end of if

    if(e.getSource()==b5){
        Icon s1=b5.getIcon();
        if(b2.getIcon()==star){
            b2.setIcon(s1);
            b5.setIcon(star);
        } else if(b4.getIcon()==star){
            b4.setIcon(s1);
        }
    }
}

```

```

        b5.setIcon(star);
    }

    else if(b6.getIcon()==star){
        b6.setIcon(s1);
        b5.setIcon(star);
    }

    else if(b8.getIcon()==star){
        b8.setIcon(s1);
        b5.setIcon(star);
    }

}

//end of if

if(e.getSource()==b6){

    Icon s1=b6.getIcon();

    if(b3.getIcon()==star){

        b3.setIcon(s1);
        b6.setIcon(star);

    } else if(b5.getIcon()==star){

        b5.setIcon(s1);
        b6.setIcon(star);

    }

    else if(b9.getIcon()==star){

        b9.setIcon(s1);
        b6.setIcon(star);

    }

}

//end of if

if(e.getSource()==b7){

    Icon s1=b7.getIcon();

    if(b4.getIcon()==star){

        b4.setIcon(s1);
        b7.setIcon(star);

    } else if(b8.getIcon()==star){

        b8.setIcon(s1);
        b7.setIcon(star);

    }

}

//end of if

if(e.getSource()==b8){

    Icon s1=b8.getIcon();

    if(b7.getIcon()==star){

        b7.setIcon(s1);
        b8.setIcon(star);

    } else if(b5.getIcon()==star){

        b5.setIcon(s1);
        b8.setIcon(star);

    }

    else if(b9.getIcon()==star){

        b9.setIcon(s1);
        b8.setIcon(star);

    }

}

//end of if

if(e.getSource()==b9){

    Icon s1=b9.getIcon();

    if(b8.getIcon()==star){

        b8.setIcon(s1);
        b9.setIcon(star);

    } else if(b6.getIcon()==star){

}

```

```

        b6.setIcon(s1);
        b9.setIcon(star);
    }
}//end of if

if(e.getSource()==sample){
Icon s1=sample.getIcon();
if(s1==samicon3){
sample.setIcon(samicon1);
b1.setIcon(ic1);
b2.setIcon(ic2);
b3.setIcon(ic3);
b4.setIcon(ic4);
b5.setIcon(ic5);
b6.setIcon(ic6);
b7.setIcon(ic7);
b8.setIcon(ic8);
b9.setIcon(ic9);
star=b9.getIcon();
starB.setIcon(ic0);
}//eof if
else if(s1==samicon1){
sample.setIcon(samicon2);
b1.setIcon(ic11);
b2.setIcon(ic12);
b3.setIcon(ic13);
b4.setIcon(ic14);
b5.setIcon(ic15);
b6.setIcon(ic16);
b7.setIcon(ic17);
b8.setIcon(ic18);
b9.setIcon(ic19);
star=b6.getIcon();
starB.setIcon(ic10);
}//eof else
else{
sample.setIcon(samicon3);
b1.setIcon(ic21);
b2.setIcon(ic22);
b3.setIcon(ic23);
b4.setIcon(ic24);
b5.setIcon(ic25);
b6.setIcon(ic26);
b7.setIcon(ic27);
b8.setIcon(ic28);
b9.setIcon(ic29);
star=b6.getIcon();
starB.setIcon(ic20);
}//eof else

}
}//end of actionPerformed

public static void main(String args[]){
new picpuzzle2();
}//end of main
}//end of class

```

[download this example](#)

# Example of Tic Tac Toe Game in Swing

In this example, we are going to see the example of tic tac toe game (also known as 0 and x (cross)).

It can be developed only through AWT api, but we are using here swing framework.

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
class TTT1 extends JFrame implements ItemListener, ActionListener{
int i,j,ii,jj,x,y,yesnull;
int a[][]={{10,1,2,3,11},{10,1,4,7,11},{10,1,5,9,11},{10,2,5,8,11},
{10,3,5,7,11},{10,3,6,9,11},{10,4,5,6,11},
{10,7,8,9,11} };
int a1[][]={{10,1,2,3,11},{10,1,4,7,11},{10,1,5,9,11},{10,2,5,8,11},
{10,3,5,7,11},{10,3,6,9,11},{10,4,5,6,11},{10,7,8,9,11} };

boolean state,type,set;

Icon ic1,ic2,icon,ic11,ic22;
Checkbox c1,c2;
JLabel l1,l2;
 JButton b[]={new JButton[9];
 JButton reset;

public void showButton(){

x=10; y=10;j=0;
for(i=0;i<=8;i++,x+=100,y++) {
b[i]=new JButton();
if(j==3)
{j=0; y+=100; x=10;}
b[i].setBounds(x,y,100,100);
add(b[i]);
b[i].addActionListener(this);
}//eof for

reset=new JButton("RESET");
reset.setBounds(100,350,100,50);
add(reset);
reset.addActionListener(this);

}//eof showButton

/****************************************/
public void check(int num1){
for(ii=0;ii<=7;ii++){
for(jj=1;jj<=3;jj++){
if(a[ii][jj]==num1){ a[ii][4]=11; }

}//eof for jj

}//eof for ii
}//eof check
/****************************************/

/****************************************/
```

```

public void complogic(int num){

    for(i=0;i<=7;i++){
        for(j=1;j<=3;j++){
            if(a[i][j]==num){ a[i][0]=11; a[i][4]=10;      }
        }
    }

    for(i=0;i<=7;i++) { // for 1
        set=true;
        if(a[i][4]==10){ //if 1
            int count=0;
            for(j=1;j<=3;j++){ //for 2
                if(b[(a[i][j]-1)].getIcon()!=null){ //if 2
                    count++;
                }
                //eof if 2
            }
            else{ yesnull=a[i][j]; }
        }
        //eof for 2
        if(count==2){ //if 2
            b[yesnull-1].setIcon(ic2);
            this.check(yesnull); set=false;break;
        }
        //eof if 1
        else
        if(a[i][0]==10){
            for(j=1;j<=3;j++){ //for2
                if(b[(a[i][j]-1)].getIcon()==null){ //if 1
                    b[(a[i][j]-1)].setIcon(ic2);
                    this.check(a[i][j]);
                    set=false;
                }
            }
            //eof if1
        }
        //eof for 2
        if(set==false)
            break;
    }
    //eof elseif
}

if(set==false)
    break;
}//eof for 1

}//eof complogic

```

```

 ****
TTT1(){
super("tic tac toe by ashwani");

CheckboxGroup cbg=new CheckboxGroup();
c1=new Checkbox("vs computer",cbg,false);
c2=new Checkbox("vs friend",cbg,false);
c1.setBounds(120,80,100,40);
c2.setBounds(120,150,100,40);
add(c1); add(c2);
c1.addItemListener(this);
c2.addItemListener(this);

```

```

state=true;type=true;set=true;
ic1=new ImageIcon("ic1.jpg");
ic2=new ImageIcon("ic2.jpg");
ic11=new ImageIcon("ic11.jpg");
ic22=new ImageIcon("ic22.jpg");

setLayout(null);
setSize(330,450);
setVisible(true);
setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
}//eof constructor

/****************************************/
public void itemStateChanged(ItemEvent e){
if(c1.getState())
{
type=false;
}

else if(c2.getState())
{ type=true;
}

remove(c1);remove(c2);
repaint(0,0,330,450);
showButton();
}//eof itemstate
/****************************************/

public void actionPerformed(ActionEvent e){
/****************************************/
if(type==true)//logicfriend
{
if(e.getSource()==reset){
for(i=0;i<=8;i++){
b[i].setIcon(null);
}//eof for
}
else{
for(i=0;i<=8;i++){
if(e.getSource()==b[i]){
if(b[i].getIcon()==null){
if(state==true){ icon=ic2;
state=false;} else{ icon=ic1; state=true; }
b[i].setIcon(icon);
}
}
}
}//eof for
}//eof else
}//eof logicfriend
else if(type==false){ // complogic
if(e.getSource()==reset){
for(i=0;i<=8;i++){
b[i].setIcon(null);
}//eof for
for(i=0;i<=7;i++)
for(j=0;j<=4;j++)
a[i][j]=a1[i][j]; //again initialsing array
}
else{ //complogic
}
}
}

```

```

        for(i=0;i<=8;i++){
            if(e.getSource()==b[i]){
                if(b[i].getIcon()==null){
                    b[i].setIcon(ic1);
                    if(b[4].getIcon()==null){
                        b[4].setIcon(ic2);
                    }
                }
            }
        }
    } //eof for
} //eof complogic

for(i=0;i<=7;i++){

Icon icon1=b[(a[i][1]-1)].getIcon();
Icon icon2=b[(a[i][2]-1)].getIcon();
Icon icon3=b[(a[i][3]-1)].getIcon();
if((icon1==icon2)&&(icon2==icon3)&&(icon1!=null)){
    if(icon1==ic1){
        b[(a[i][1]-1)].setIcon(ic11);
        b[(a[i][2]-1)].setIcon(ic11);
        b[(a[i][3]-1)].setIcon(ic11);
        JOptionPane.showMessageDialog(TTT1.this,"!!!YOU won!!! click reset");
        break;
    }
    else if(icon1==ic2){
        b[(a[i][1]-1)].setIcon(ic22);
        b[(a[i][2]-1)].setIcon(ic22);
        b[(a[i][3]-1)].setIcon(ic22);
        JOptionPane.showMessageDialog(TTT1.this,"won! click reset");
        break;
    }
}
}

} //eof actionPerformed
/********************************************/


public static void main(String []args){
new TTT1();
} //eof main
} //eof class

```

[download this example](#)

## Online Exam Project in Java Swing without database

In this project, there are given 10 questions to play. User can bookmark any question for the reconsideration while going to result.

We are using here java array to store the questions, options and answers not database. You can use collection framework or database in place of array.

```
/*Online Java Paper Test*/

import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

class OnlineTest extends JFrame implements ActionListener
{
    JLabel l;
    JRadioButton jb[] = new JRadioButton[5];
    JButton b1,b2;
    ButtonGroup bg;
    int count=0,current=0,x=1,y=1,now=0;
    int m[] = new int[10];
    OnlineTest(String s)
    {
        super(s);
        l=new JLabel();
        l=new JLabel();
        add(l);
        bg=new ButtonGroup();
        for(int i=0;i<5;i++)
        {
            jb[i]=new JRadioButton();
            add(jb[i]);
            bg.add(jb[i]);
        }
        b1=new JButton("Next");
        b2=new JButton("Bookmark");
        b1.addActionListener(this);
        b2.addActionListener(this);
        add(b1);add(b2);
        set();
        l.setBounds(30,40,450,20);
        jb[0].setBounds(50,80,100,20);
        jb[1].setBounds(50,110,100,20);
        jb[2].setBounds(50,140,100,20);
        jb[3].setBounds(50,170,100,20);
        b1.setBounds(100,240,100,30);
        b2.setBounds(270,240,100,30);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setLayout(null);
        setLocation(250,100);
        setVisible(true);
        setSize(600,350);
    }

    public void actionPerformed(ActionEvent e)
    {
        if(e.getSource()==b1)
        {
            if(check())
                count=count+1;
            current++;
            set();
            if(current==9)
            {
                b1.setEnabled(false);
                b2.setText("Result");
            }
        }
        if(e.getActionCommand().equals("Bookmark"))

```

```

{
JButton bk=new JButton("Bookmark"+x);
bk.setBounds(480,20+30*x,100,30);
add(bk);
bk.addActionListener(this);
m[x]=current;
x++;
current++;
set();
if(current==9)
    b2.setText("Result");
setVisible(false);
setVisible(true);
}
for(int i=0,y=1;i<x;i++,y++)
{
if(e.getActionCommand().equals("Bookmark"+y))
{
if(check())
    count=count+1;
now=current;
current=m[y];
set();
((JButton)e.getSource()).setEnabled(false);
current=now;
}
}

if(e.getActionCommand().equals("Result"))
{
if(check())
    count=count+1;
current++;
//System.out.println("correct ans="+count);
 JOptionPane.showMessageDialog(this,"correct ans="+count);
System.exit(0);
}
}

void set()
{
jb[4]. setSelected(true);
if(current==0)
{
l.setText("Que1: Which one among these is not a primitive datatype?");
jb[0].setText("int");jb[1].setText("Float");jb[2].setText("boolean");jb[3].setText("char");
}
if(current==1)
{
l.setText("Que2: Which class is available to all the class automatically?");
jb[0].setText("Swing");jb[1].setText("Applet");jb[2].setText("Object");jb[3].setText("ActionEvent");
}
if(current==2)
{
l.setText("Que3: Which package is directly available to our class without importing it?");
jb[0].setText("swing");jb[1].setText("applet");jb[2].setText("net");jb[3].setText("lang");
}
if(current==3)
{
l.setText("Que4: String class is defined in which package?");
jb[0].setText("lang");jb[1].setText("Swing");jb[2].setText("Applet");jb[3].setText("awt");
}
}

```

```

}

if(current==4)
{
    l.setText("Que5: Which institute is best for java coaching?");
    jb[0].setText("Utek");jb[1].setText("Aptech");jb[2].setText("SSS IT");jb[3].setText("jtek");
}
if(current==5)
{
    l.setText("Que6: Which one among these is not a keyword?");
    jb[0].setText("class");jb[1].setText("int");jb[2].setText("get");jb[3].setText("if");
}
if(current==6)
{
    l.setText("Que7: Which one among these is not a class? ");
    jb[0].setText("Swing");jb[1].setText("Actionperformed");jb[2].setText("ActionEvent");
        jb[3].setText("Button");
}
if(current==7)
{
    l.setText("Que8: which one among these is not a function of Object class?");
    jb[0].setText("toString");jb[1].setText("finalize");jb[2].setText("equals");
        jb[3].setText("getDocumentBase");
}
if(current==8)
{
    l.setText("Que9: which function is not present in Applet class?");
    jb[0].setText("init");jb[1].setText("main");jb[2].setText("start");jb[3].setText("destroy");
}
if(current==9)
{
    l.setText("Que10: Which one among these is not a valid component?");
    jb[0].setText("JButton");jb[1].setText("JList");jb[2].setText("JButtonGroup");
        jb[3].setText("Jpre");
}
l.setBounds(30,40,450,20);
for(int i=0,j=0;i<=90;i+=30,j++)
    jb[j].setBounds(50,80+i,200,20);
}
boolean check()
{
    if(current==0)
        return(jb[1].isSelected());
    if(current==1)
        return(jb[2].isSelected());
    if(current==2)
        return(jb[3].isSelected());
    if(current==3)
        return(jb[0].isSelected());
    if(current==4)
        return(jb[2].isSelected());
    if(current==5)
        return(jb[2].isSelected());
    if(current==6)
        return(jb[1].isSelected());
    if(current==7)
        return(jb[3].isSelected());
    if(current==8)
        return(jb[1].isSelected());
    if(current==9)
        return(jb[2].isSelected());
}

```

```

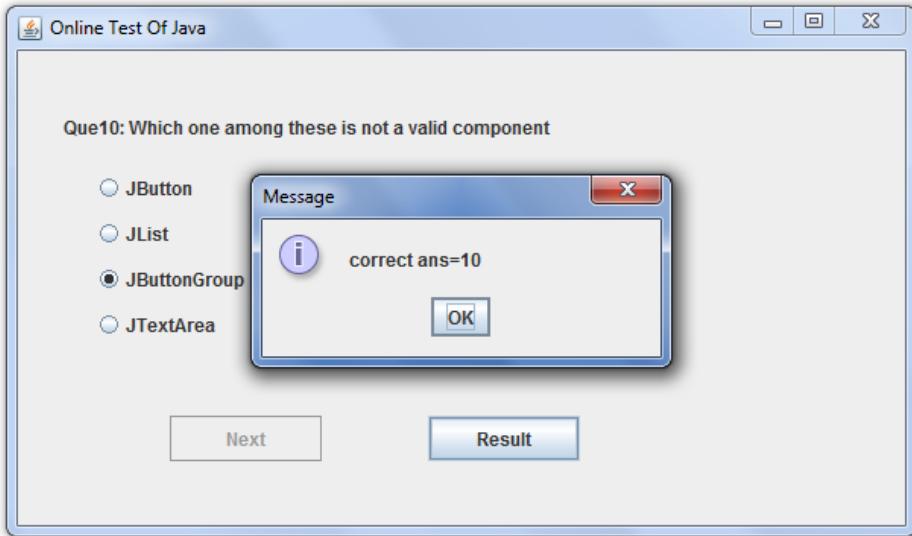
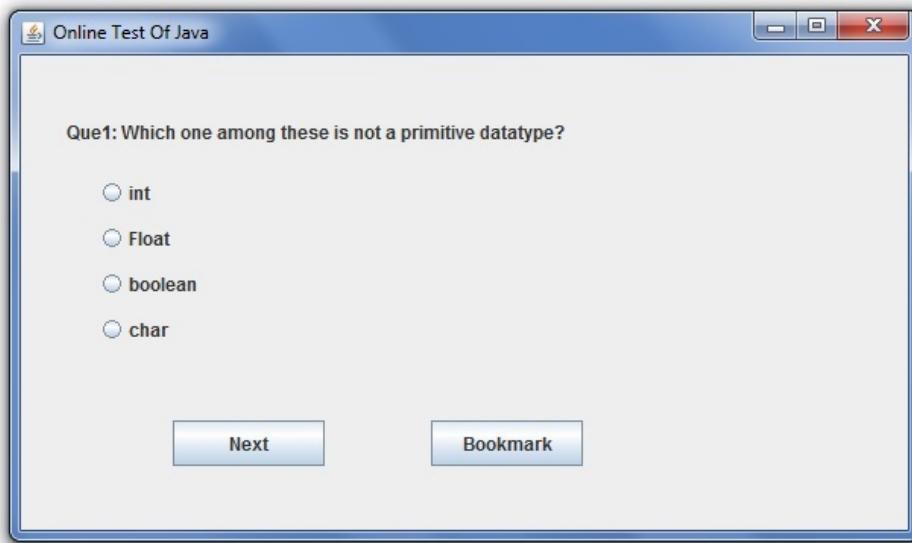
        return false;
    }
    public static void main(String s[])
    {
        new OnlineTest("Online Test Of Java");
    }
}

```

[download this mini project](#)

---

## Output



## BorderLayout (LayoutManagers):

---

### LayoutManagers:

The LayoutManagers are used to arrange components in a particular manner. LayoutManager is an interface that is implemented by all the classes of layout managers. There are following classes that represents the layout managers:

1. `java.awt.BorderLayout`
2. `java.awt.FlowLayout`
3. `java.awt.GridLayout`

4. `java.awt.CardLayout`
  5. `java.awt.GridBagLayout`
  6. `javax.swing.BoxLayout`
  7. `javax.swing.GroupLayout`
  8. `javax.swing.ScrollPaneLayout`
  9. `javax.swing.SpringLayout` etc.
- 

## BorderLayout:

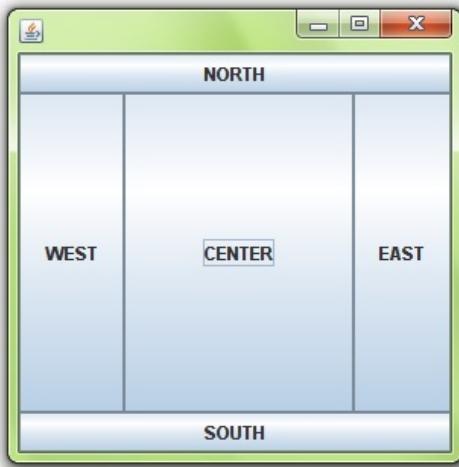
The BorderLayout is used to arrange the components in five regions: north, south, east, west and center. Each region (area) may contain one component only. It is the default layout of frame or window. The BorderLayout provides five constants for each region:

1. `public static final int NORTH`
2. `public static final int SOUTH`
3. `public static final int EAST`
4. `public static final int WEST`
5. `public static final int CENTER`

## Constructors of BorderLayout class:

- `BorderLayout()`: creates a border layout but with no gaps between the components.
  - `JBorderLayout(int hgap, int vgap)`: creates a border layout with the given horizontal and vertical gaps between the components.
- 

## Example of BorderLayout class:



```
import java.awt.*;
import javax.swing.*;

public class Border {
JFrame f;
Border(){
f=new JFrame();

JButton b1=new JButton("NORTH");
JButton b2=new JButton("SOUTH");
JButton b3=new JButton("EAST");
JButton b4=new JButton("WEST");
JButton b5=new JButton("CENTER");

f.add(b1,BorderLayout.NORTH);
f.add(b2,BorderLayout.SOUTH);
f.add(b3,BorderLayout.EAST);
f.add(b4,BorderLayout.WEST);
```

```

f.add(b5,BorderLayout.CENTER);

f.setSize(300,300);
f.setVisible(true);
}

public static void main(String[] args) {
    new Border();
}
}

```

[download this example](#)

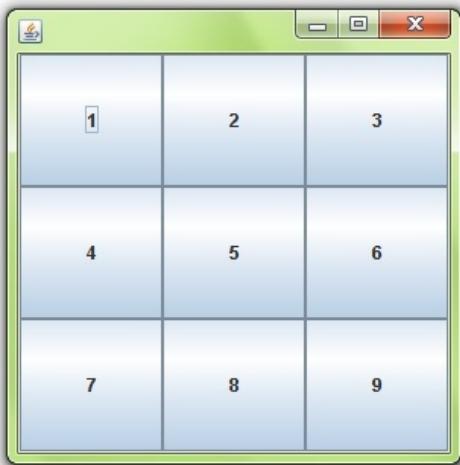
## GridLayout

The GridLayout is used to arrange the components in rectangular grid. One component is displayed in each rectangle.

Constructors of GridLayout class:

1. **GridLayout():** creates a grid layout with one column per component in a row.
2. **GridLayout(int rows, int columns):** creates a grid layout with the given rows and columns but no gaps between the components.
3. **GridLayout(int rows, int columns, int hgap, int vgap):** creates a grid layout with the given rows and columns alongwith given horizontal and vertical gaps.

Example of GridLayout class:



```

import java.awt.*;
import javax.swing.*;

public class MyGridLayout{
JFrame f;
MyGridLayout(){
f=new JFrame();

JButton b1=new JButton("1");
JButton b2=new JButton("2");
JButton b3=new JButton("3");
JButton b4=new JButton("4");

```

```

JButton b5=new JButton("5");
JButton b6=new JButton("6");
JButton b7=new JButton("7");
JButton b8=new JButton("8");
JButton b9=new JButton("9");

f.add(b1);f.add(b2);f.add(b3);f.add(b4);f.add(b5);
f.add(b6);f.add(b7);f.add(b8);f.add(b9);

f.setLayout(new GridLayout(3,3));
//setting grid layout of 3 rows and 3 columns

f.setSize(300,300);
f.setVisible(true);
}

public static void main(String[] args) {
new MyGridLayout();
}
}

```

[download this example](#)

## FlowLayout

The FlowLayout is used to arrange the components in a line, one after another (in a flow). It is the default layout of applet or panel.

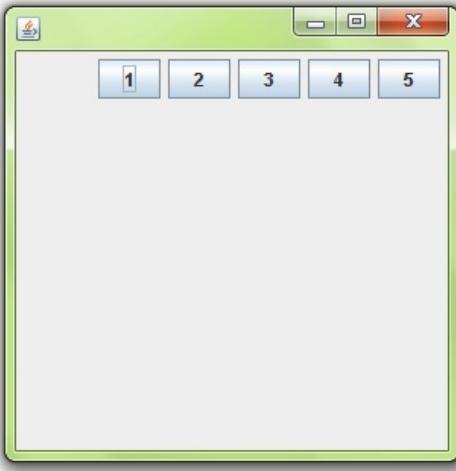
Fields of FlowLayout class:

1. **public static final int LEFT**
2. **public static final int RIGHT**
3. **public static final int CENTER**
4. **public static final int LEADING**
5. **public static final int TRAILING**

Constructors of FlowLayout class:

1. **FlowLayout():** creates a flow layout with centered alignment and a default 5 unit horizontal and vertical gap.
2. **FlowLayout(int align):** creates a flow layout with the given alignment and a default 5 unit horizontal and vertical gap.
3. **FlowLayout(int align, int hgap, int vgap):** creates a flow layout with the given alignment and the given horizontal and vertical gap.

Example of FlowLayout class:



```
import java.awt.*;
import javax.swing.*;

public class MyFlowLayout{
JFrame f;
MyFlowLayout(){
f=new JFrame();

JButton b1=new JButton("1");
JButton b2=new JButton("2");
JButton b3=new JButton("3");
JButton b4=new JButton("4");
JButton b5=new JButton("5");

f.add(b1);f.add(b2);f.add(b3);f.add(b4);f.add(b5);

f.setLayout(new FlowLayout(FlowLayout.RIGHT));
//setting flow layout of right alignment

f.setSize(300,300);
f.setVisible(true);
}

public static void main(String[] args) {
new MyFlowLayout();
}
}
```

[download this example](#)

## BoxLayout class:

The BoxLayout is used to arrange the components either vertically or horizontally. For this purpose, BoxLayout provides four constants. They are as follows:

Note: BoxLayout class is found in `javax.swing` package.

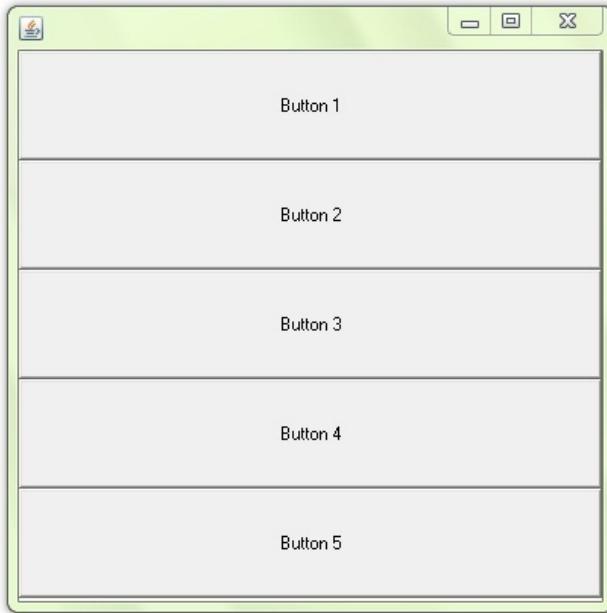
### Fields of BoxLayout class:

1. **public static final int X\_AXIS**
2. **public static final int Y\_AXIS**
3. **public static final int LINE\_AXIS**
4. **public static final int PAGE\_AXIS**

## Constructor of BoxLayout class:

1. **BoxLayout(Container c, int axis):** creates a box layout that arranges the components with the given axis.

## Example of BoxLayout class with Y-AXIS:



```

import java.awt.*;
import javax.swing.*;

public class BoxLayoutExample1 extends Frame {
    Button buttons[];

    public BoxLayoutExample1 () {
        buttons = new Button [5];

        for (int i = 0;i<5;i++) {
            buttons[i] = new Button ("Button " + (i + 1));
            add (buttons[i]);
        }

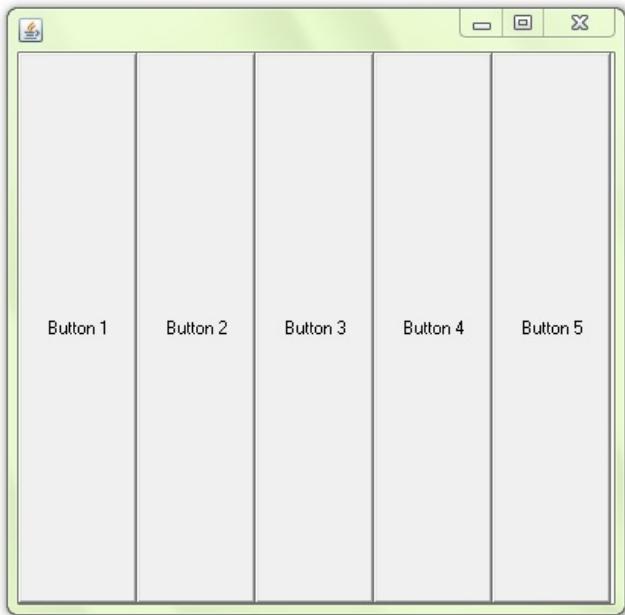
        setLayout (new BoxLayout (this, BoxLayout.Y_AXIS));
        setSize(400,400);
        setVisible(true);
    }

    public static void main(String args[]){
        BoxLayoutExample1 b=new BoxLayoutExample1();
    }
}

```

[download this example](#)

### Example of BoxLayout class with X-AXIS:



```
import java.awt.*;
import javax.swing.*;

public class BoxLayoutExample2 extends Frame {
    Button buttons[];

    public BoxLayoutExample2() {
        buttons = new Button [5];

        for (int i = 0;i<5;i++) {
            buttons[i] = new Button ("Button " + (i + 1));
            add (buttons[i]);
        }

        setLayout (new BoxLayout(this, BoxLayout.X_AXIS));
        setSize(400,400);
        setVisible(true);
    }

    public static void main(String args[]){
        BoxLayoutExample2 b=new BoxLayoutExample2();
    }
}
```

[download this example](#)

## CardLayout class

The CardLayout class manages the components in such a manner that only one component is visible at a time. It treats each component as a card that is why it is known as CardLayout.

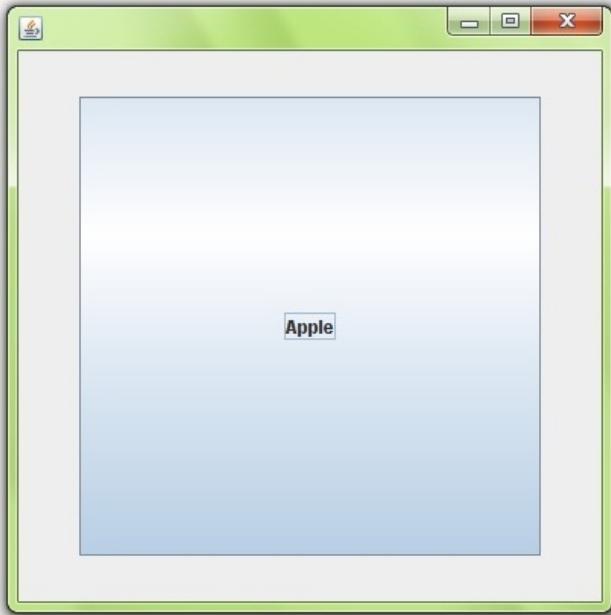
## Constructors of CardLayout class:

1. **CardLayout():** creates a card layout with zero horizontal and vertical gap.
2. **CardLayout(int hgap, int vgap):** creates a card layout with the given horizontal and vertical gap.

## Commonly used methods of CardLayout class:

- **public void next(Container parent):** is used to flip to the next card of the given container.
- **public void previous(Container parent):** is used to flip to the previous card of the given container.
- **public void first(Container parent):** is used to flip to the first card of the given container.
- **public void last(Container parent):** is used to flip to the last card of the given container.
- **public void show(Container parent, String name):** is used to flip to the specified card with the given name.

## Example of CardLayout class:



```
import java.awt.*;
import java.awt.event.*;

import javax.swing.*;

public class CardLayoutExample extends JFrame implements ActionListener{
CardLayout card;
JButton b1,b2,b3;
Container c;
CardLayoutExample(){
    c=getContentPane();
    card=new CardLayout(40,30);
    //create CardLayout object with 40 hor space and 30 ver space
    c.setLayout(card);

    b1=new JButton("Apple");
    b2=new JButton("Boy");
    b3=new JButton("Cat");
    b1.addActionListener(this);
    b2.addActionListener(this);
    b3.addActionListener(this);
}
```

```
c.add("a",b1);c.add("b",b2);c.add("c",b3);

}

public void actionPerformed(ActionEvent e) {
card.next(c);
}

public static void main(String[] args) {
CardLayoutExample cl=new CardLayoutExample();
cl.setSize(400,400);
cl.setVisible(true);
cl.setDefaultCloseOperation(EXIT_ON_CLOSE);
}
}
```

[download this example](#)

## Java Applet

Applet is a special type of program that is embedded in the webpage to generate the dynamic content. It runs inside the browser and works at client side.

### Advantage of Applet

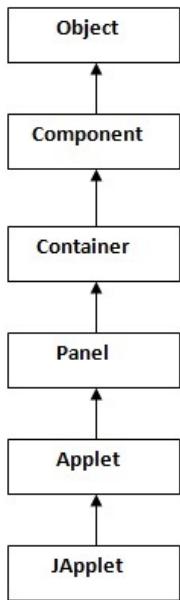
There are many advantages of applet. They are as follows:

- It works at client side so less response time.
- Secured
- It can be executed by browsers running under many platforms, including Linux, Windows, Mac Os etc.

### Drawback of Applet

- Plugin is required at client browser to execute applet.

### Hierarchy of Applet



As displayed in the above diagram, Applet class extends Panel. Panel class extends Container which is the subclass of Component.

---

## Lifecycle of Java Applet

1. Applet is initialized.
  2. Applet is started.
  3. Applet is painted.
  4. Applet is stopped.
  5. Applet is destroyed.
- 

## Lifecycle methods for Applet:

The `java.applet.Applet` class provides 4 life cycle methods and `java.awt.Component` class provides 1 life cycle method for an applet.

### `java.applet.Applet` class

For creating any applet `java.applet.Applet` class must be inherited. It provides 4 life cycle methods of applet.

1. **`public void init()`:** is used to initialize the Applet. It is invoked only once.
2. **`public void start()`:** is invoked after the `init()` method or browser is maximized. It is used to start the Applet.
3. **`public void stop()`:** is used to stop the Applet. It is invoked when Applet is stop or browser is minimized.
4. **`public void destroy()`:** is used to destroy the Applet. It is invoked only once.

### `java.awt.Component` class

The `Component` class provides 1 life cycle method of applet.

1. **`public void paint(Graphics g)`:** is used to paint the Applet. It provides `Graphics` class object that can be used for drawing oval, rectangle, arc etc.
- 

## Who is responsible to manage the life cycle of an applet?

Java Plug-in software.

---

## How to run an Applet?

There are two ways to run an applet

1. By html file.
2. By `appletViewer` tool (for testing purpose).

## Simple example of Applet by html file:

To execute the applet by html file, create an applet and compile it. After that create an html file and place the applet code in html file. Now click the html file.

```
//First.java
import java.applet.Applet;
import java.awt.Graphics;
public class First extends Applet{

public void paint(Graphics g){
g.drawString("welcome",150,150);
}

}
```

 **Note:** class must be public because its object is created by Java Plugin software that resides on the browser.

### myapplet.html

```
<html>
<body>
<applet code="First.class" width="300" height="300">
</applet>
</body>
</html>
```

## Simple example of Applet by appletviewer tool:

To execute the applet by appletviewer tool, create an applet that contains applet tag in comment and compile it. After that run it by: appletviewer First.java. Now Html file is not required but it is for testing purpose only.

```
//First.java
import java.applet.Applet;
import java.awt.Graphics;
public class First extends Applet{

public void paint(Graphics g){
g.drawString("welcome to applet",150,150);
}

}

/*
<applet code="First.class" width="300" height="300">
</applet>
*/
```

To execute the applet by appletviewer tool, write in command prompt:

```
c:\>javac First.java
c:\>appletviewer First.java
```

# Displaying Graphics in Applet

java.awt.Graphics class provides many methods for graphics programming.

## Commonly used methods of Graphics class:

1. **public abstract void drawString(String str, int x, int y):** is used to draw the specified string.
2. **public void drawRect(int x, int y, int width, int height):** draws a rectangle with the specified width and height.
3. **public abstract void fillRect(int x, int y, int width, int height):** is used to fill rectangle with the default color and specified width and height.
4. **public abstract void drawOval(int x, int y, int width, int height):** is used to draw oval with the specified width and height.
5. **public abstract void fillOval(int x, int y, int width, int height):** is used to fill oval with the default color and specified width and height.
6. **public abstract void drawLine(int x1, int y1, int x2, int y2):** is used to draw line between the points(x1, y1) and (x2, y2).
7. **public abstract boolean drawImage(Image img, int x, int y, ImageObserver observer):** is used draw the specified image.
8. **public abstract void drawArc(int x, int y, int width, int height, int startAngle, int arcAngle):** is used draw a circular or elliptical arc.
9. **public abstract void fillArc(int x, int y, int width, int height, int startAngle, int arcAngle):** is used to fill a circular or elliptical arc.
10. **public abstract void setColor(Color c):** is used to set the graphics current color to the specified color.
11. **public abstract void setFont(Font font):** is used to set the graphics current font to the specified font.

## Example of Graphics in applet:

```
import java.applet.Applet;
import java.awt.*;

public class GraphicsDemo extends Applet{

    public void paint(Graphics g){
        g.setColor(Color.red);
        g.drawString("Welcome",50, 50);
        g.drawLine(20,30,20,300);
        g.drawRect(70,100,30,30);
        g.fillRect(170,100,30,30);
        g.drawOval(70,200,30,30);

        g.setColor(Color.pink);
        g.fillOval(170,200,30,30);
        g.drawArc(90,150,30,30,30,270);
        g.fillArc(270,150,30,30,0,180);

    }
}
```

myapplet.html

```
<html>
<body>
<applet code="GraphicsDemo.class" width="300" height="300">
</applet>
</body>
</html>
```

[download this example.](#)

# Displaying Image in Applet

Applet is mostly used in games and animation. For this purpose image is required to be displayed. The `java.awt.Graphics` class provide a method `drawImage()` to display the image.

## Syntax of `drawImage()` method:

1. **public abstract boolean drawImage(Image img, int x, int y, ImageObserver observer):** is used draw the specified image.

## How to get the object of Image:

The `java.applet.Applet` class provides `getImage()` method that returns the object of `Image`. Syntax:

```
public Image getImage(URL u, String image){}
```

## Other required methods of Applet class to display image:

1. **public URL getDocumentBase():** is used to return the URL of the document in which applet is embedded.
2. **public URL getCodeBase():** is used to return the base URL.

---

## Example of displaying image in applet:

```
import java.awt.*;
import java.applet.*;

public class DisplayImage extends Applet {

    Image picture;

    public void init() {
        picture = getImage(getDocumentBase(),"sonoo.jpg");
    }

    public void paint(Graphics g) {
        g.drawImage(picture, 30,30, this);
    }
}
```

In the above example, `drawImage()` method of `Graphics` class is used to display the image. The 4th argument of `drawImage()` method of is `ImageObserver` object. The `Component` class implements `ImageObserver` interface. So current class object would also be treated as `ImageObserver` because `Applet` class indirectly extends the `Component` class.

### myapplet.html

```
<html>
<body>
<applet code="DisplayImage.class" width="300" height="300">
</applet>
</body>
</html>
```

[download this example.](#)

## Animation in Applet

Applet is mostly used in games and animation. For this purpose image is required to be moved.

### Example of animation in applet:

```
import java.awt.*;
import java.applet.*;
public class AnimationExample extends Applet {

    Image picture;

    public void init() {
        picture =getImage(getDocumentBase(),"bike_1.gif");
    }

    public void paint(Graphics g) {
        for(int i=0;i<500;i++){
            g.drawImage(picture, i, 30, this);

            try{Thread.sleep(100);}catch(Exception e){}
        }
    }
}
```

In the above example, `drawImage()` method of `Graphics` class is used to display the image. The 4th argument of `drawImage()` method of is `ImageObserver` object. The `Component` class implements `ImageObserver` interface. So current class object would also be treated as `ImageObserver` because `Applet` class indirectly extends the `Component` class.

### myapplet.html

```
<html>
<body>
<applet code="DisplayImage.class" width="300" height="300">
</applet>
</body>
</html>
```

[download this example.](#)

## EventHandling in Applet

As we perform event handling in AWT or Swing, we can perform it in applet also. Let's see the simple example of event handling in applet that prints a message by click on the button.

### Example of EventHandling in applet:

```

import java.applet.*;
import java.awt.*;
import java.awt.event.*;
public class EventApplet extends Applet implements ActionListener{
Button b;
TextField tf;

public void init(){
tf=new TextField();
tf.setBounds(30,40,150,20);

b=new Button("Click");
b.setBounds(80,150,60,50);

add(b);add(tf);
b.addActionListener(this);

setLayout(null);
}

public void actionPerformed(ActionEvent e){
tf.setText("Welcome");
}
}

```

In the above example, we have created all the controls in init() method because it is invoked only once.

### myapplet.html

```

<html>
<body>
<applet code="EventApplet.class" width="300" height="300">
</applet>
</body>
</html>

```

[download this example.](#)

## JApplet class in Applet

As we prefer Swing to AWT. Now we can use JApplet that can have all the controls of swing. The JApplet class extends the Applet class.

### Example of EventHandling in JApplet:

```

import java.applet.*;
import javax.swing.*;
import java.awt.event.*;
public class EventJApplet extends JApplet implements ActionListener{
JButton b;
JTextField tf;
public void init(){

tf=new JTextField();

```

```

tf.setBounds(30,40,150,20);

b=new JButton("Click");
b.setBounds(80,150,70,40);

add(b);add(tf);
b.addActionListener(this);

setLayout(null);
}

public void actionPerformed(ActionEvent e){
tf.setText("Welcome");
}
}

```

In the above example, we have created all the controls in init() method because it is invoked only once.

### myapplet.html

```

<html>
<body>
<applet code="EventJApplet.class" width="300" height="300">
</applet>
</body>
</html>

```

[download this example.](#)

## Painting in Applet

We can perform painting operation in applet by the mouseDragged() method of MouseMotionListener.

### Example of Painting in Applet:

```

import java.awt.*;
import java.awt.event.*;
import java.applet.*;
public class MouseDrag extends Applet implements MouseMotionListener{

public void init(){
addMouseMotionListener(this);
setBackground(Color.red);
}

public void mouseDragged(MouseEvent me){
Graphics g=getGraphics();
g.setColor(Color.white);
g.fillOval(me.getX(),me.getY(),5,5);
}
public void mouseMoved(MouseEvent me){}
}

```

In the above example, `getX()` and `getY()` method of `MouseEvent` is used to get the current x-axis and y-axis. The `getGraphics()` method of `Component` class returns the object of `Graphics`.

## myapplet.html

```
<html>
<body>
<applet code="MouseDrag.class" width="300" height="300">
</applet>
</body>
</html>
```

[download this example.](#)

## Digital clock in Applet

Digital clock can be created by using the `Calendar` and `SimpleDateFormat` class. Let's see the simple example:

### Example of Digital clock in Applet:

```
import java.applet.*;
import java.awt.*;
import java.util.*;
import java.text.*;

public class DigitalClock extends Applet implements Runnable {

    Thread t = null;
    int hours=0, minutes=0, seconds=0;
    String timeString = "";

    public void init() {
        setBackground( Color.green);
    }

    public void start() {
        t = new Thread( this );
        t.start();
    }

    public void run() {
        try {
            while (true) {

                Calendar cal = Calendar.getInstance();
                hours = cal.get( Calendar.HOUR_OF_DAY );
                if ( hours > 12 ) hours -= 12;
                minutes = cal.get( Calendar.MINUTE );
                seconds = cal.get( Calendar.SECOND );

                SimpleDateFormat formatter = new SimpleDateFormat("hh:mm:ss");
                Date date = cal.getTime();


```

```

        timeString = formatter.format( date );

        repaint();
        t.sleep( 1000 ); // interval given in milliseconds
    }
}

catch (Exception e) { }

}

public void paint( Graphics g ) {
    g.setColor( Color.blue );
    g.drawString( timeString, 50, 50 );
}

}

```

In the above example, `getX()` and `getY()` method of `MouseEvent` is used to get the current x-axis and y-axis. The `getGraphics()` method of `Component` class returns the object of `Graphics`.

## [myapplet.html](#)

```

<html>
<body>
<applet code="DigitalClock.class" width="300" height="300">
</applet>
</body>
</html>

```

[download this example.](#)

## Analog clock in Applet

Analog clock can be created by using the `Math` class. Let's see the simple example:

### Example of Analog clock in Applet:

```

import java.applet.*;
import java.awt.*;
import java.util.*;
import java.text.*;

public class MyClock extends Applet implements Runnable {

    int width, height;
    Thread t = null;
    boolean threadSuspended;
    int hours=0, minutes=0, seconds=0;
    String timeString = "";

    public void init() {
        width = getSize().width;
        height = getSize().height;
        setBackground( Color.black );

```

```

}

public void start() {
    if ( t == null ) {
        t = new Thread( this );
        t.setPriority( Thread.MIN_PRIORITY );
        threadSuspended = false;
        t.start();
    }
    else {
        if ( threadSuspended ) {
            threadSuspended = false;
            synchronized( this ) {
                notify();
            }
        }
    }
}

public void stop() {
    threadSuspended = true;
}

public void run() {
    try {
        while (true) {

            Calendar cal = Calendar.getInstance();
            hours = cal.get( Calendar.HOUR_OF_DAY );
            if ( hours > 12 ) hours -= 12;
            minutes = cal.get( Calendar.MINUTE );
            seconds = cal.get( Calendar.SECOND );

            SimpleDateFormat formatter
                = new SimpleDateFormat( "hh:mm:ss", Locale.getDefault() );
            Date date = cal.getTime();
            timeString = formatter.format( date );

            // Now the thread checks to see if it should suspend itself
            if ( threadSuspended ) {
                synchronized( this ) {
                    while ( threadSuspended ) {
                        wait();
                    }
                }
            }
            repaint();
            t.sleep( 1000 ); // interval specified in milliseconds
        }
    }
    catch (Exception e) { }
}

void drawHand( double angle, int radius, Graphics g ) {
    angle -= 0.5 * Math.PI;
    int x = (int)( radius*Math.cos(angle) );
    int y = (int)( radius*Math.sin(angle) );
    g.drawLine( width/2, height/2, width/2 + x, height/2 + y );
}

```

```

void drawWedge( double angle, int radius, Graphics g ) {
    angle -= 0.5 * Math.PI;
    int x = (int)( radius*Math.cos(angle) );
    int y = (int)( radius*Math.sin(angle) );
    angle += 2*Math.PI/3;
    int x2 = (int)( 5*Math.cos(angle) );
    int y2 = (int)( 5*Math.sin(angle) );
    angle += 2*Math.PI/3;
    int x3 = (int)( 5*Math.cos(angle) );
    int y3 = (int)( 5*Math.sin(angle) );
    g.drawLine( width/2+x2, height/2+y2, width/2 + x, height/2 + y );
    g.drawLine( width/2+x3, height/2+y3, width/2 + x, height/2 + y );
    g.drawLine( width/2+x2, height/2+y2, width/2 + x3, height/2 + y3 );
}

public void paint( Graphics g ) {
    g.setColor( Color.gray );
    drawWedge( 2*Math.PI * hours / 12, width/5, g );
    drawWedge( 2*Math.PI * minutes / 60, width/3, g );
    drawHand( 2*Math.PI * seconds / 60, width/2, g );
    g.setColor( Color.white );
    g.drawString( timeString, 10, height-10 );
}
}

```

## myapplet.html

```

<html>
<body>
<applet code="MyClock.class" width="300" height="300">
</applet>
</body>
</html>

```

[download this example.](#)

## Parameter in Applet

We can get any information from the HTML file as a parameter. For this purpose, Applet class provides a method named `getParameter()`.  
Syntax:

```
public String getParameter(String parameterName)
```

## Example of using parameter in Applet:

```

import java.applet.Applet;
import java.awt.Graphics;

public class UseParam extends Applet{

    public void paint(Graphics g){
        String str=getParameter("msg");
        g.drawString(str,50, 50);
    }
}

```

```
}
```

```
}
```

## myapplet.html

```
<html>
<body>
<applet code="UseParam.class" width="300" height="300">
<param name="msg" value="Welcome to applet">
</applet>
</body>
</html>
```

download this example.

[<<prev](#)[next>>](#)

## Applet Communication

java.applet.AppletContext class provides the facility of communication between applets. We provide the name of applet through the HTML file. It provides getApplet() method that returns the object of Applet. Syntax:

```
public Applet getApplet(String name){}
```

## Example of Applet Communication

```
import java.applet.*;
import java.awt.*;
import java.awt.event.*;
public class ContextApplet extends Applet implements ActionListener{
Button b;

public void init(){
b=new Button("Click");
b.setBounds(50,50,60,50);

add(b);
b.addActionListener(this);
}

public void actionPerformed(ActionEvent e){

AppletContext ctx=getAppletContext();
Applet a=ctx.getApplet("app2");
a.setBackground(Color.yellow);
}
}
```

## myapplet.html

```

<html>
<body>
<applet code="ContextApplet.class" width="150" height="150" name="app1">
</applet>

<applet code="First.class" width="150" height="150" name="app2">
</applet>
</body>
</html>

```

[download this example.](#)

## Java Reflection API

**Java Reflection** is a process of examining or modifying the run time behavior of a class at run time .

The **java.lang.Class** class provides many methods that can be used to get metadata, examine and change the run time behavior of a class.

The java.lang and java.lang.reflect packages provide classes for java reflection.

### Where it is used

The Reflection API is mainly used in:

- IDE (Integrated Development Environment) e.g. Eclipse, MyEclipse, NetBeans etc.
- Debugger
- Test Tools etc.

## java.lang.Class class

The java.lang.Class class performs mainly two tasks:

- provides methods to get the metadata of a class at run time.
- provides methods to examine and change the run time behavior of a class.

### Commonly used methods of Class class:

Method	Description
1) public String getName()	returns the class name
2) public static Class forName(String className)throws ClassNotFoundException	loads the class and returns the reference of Class class.
3) public Object newInstance()throws InstantiationException,IllegalAccessException	creates new instance.
4) public boolean isInterface()	checks if it is interface.
5) public boolean isArray()	checks if it is array.
6) public boolean isPrimitive()	checks if it is primitive.
7) public Class getSuperclass()	returns the superclass class reference.
8) public Field[] getDeclaredFields()throws SecurityException	returns the total number of fields of this class.
9) public Method[] getDeclaredMethods()throws SecurityException	returns the total number of methods of this class.

10) public Constructor[] getDeclaredConstructors()throws SecurityException	returns the total number of constructors of this class.
11) public Method getDeclaredMethod(String name,Class[] parameterTypes)throws NoSuchMethodException,SecurityException	returns the method class instance.

## How to get the object of Class class?

There are 3 ways to get the instance of Class class. They are as follows:

- forName() method of Class class
- getClass() method of Object class
- the .class syntax

### 1) forName() method of Class class

- is used to load the class dynamically.
- returns the instance of Class class.
- It should be used if you know the fully qualified name of class. This cannot be used for primitive types.

Let's see the simple example of forName() method.

```
class Simple{}

class Test{
    public static void main(String args[]){
        Class c=Class.forName("Simple");
        System.out.println(c.getName());
    }
}
```

Output:Simple

### 2) getClass() method of Object class

It returns the instance of Class class. It should be used if you know the type. Moreover, it can be used with primitives.

```
class Simple{}

class Test{
    void printName(Object obj){
        Class c=obj.getClass();
        System.out.println(c.getName());
    }
    public static void main(String args[]){
        Simple s=new Simple();

        Test t=new Test();
        t.printName(s);
    }
}
```

Output:Simple

### 3) The .class syntax

If a type is available but there is no instance then it is possible to obtain a Class by appending ".class" to the name of the type. It can be used for primitive data type also.

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

```

Class c = boolean.class;
System.out.println(c.getName());

Class c2 = Test.class;
System.out.println(c2.getName());
}

}

Output:boolean
Test

```

---

## Determining the class object

Following methods of Class class is used to determine the class object:

- 1) public boolean isInterface():** determines if the specified Class object represents an interface type.
- 2) public boolean isArray():** determines if this Class object represents an array class.
- 3) public boolean isPrimitive():** determines if the specified Class object represents a primitive type.

Let's see the simple example of reflection api to determine the object type.

```

class Simple{}
interface My{}


class Test{
    public static void main(String args[]){
        try{
            Class c=Class.forName("Simple");
            System.out.println(c.isInterface());


            Class c2=Class.forName("My");
            System.out.println(c2.isInterface());


        }catch(Exception e){System.out.println(e);}
    }
}

```

---

Output:false  
true

## Next Topics of Reflection API Tutorial

[newInstance\(\) method](#)

[Understanding javap tool](#)

[creating javap tool](#)

[creating appletviewer tool](#)

[Call private method from another class](#)

## newInstance() method

The **newInstance()** method of **Class** class and **Constructor** class is used to create a new instance of the class.

The newInstance() method of Class class can invoke zero-argument constructor whereas newInstance() method of Constructor class can invoke any number of arguments. So Constructor class is preferred over Class class.

## Syntax of newInstance() method of Class class

```
public T newInstance()throws InstantiationException,IllegalAccessException
```

Here T is the generic version. You can think it like Object class. You will learn about generics later.

## Example of newInstance() method

Let's see the simple example to use newInstance() method.

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

```
class Test{  
    public static void main(String args[]){  
        try{  
            Class c=Class.forName("Simple");  
            Simple s=(Simple)c.newInstance();  
            s.message();  
  
        }catch(Exception e){System.out.println(e);}  
  
    }  
}
```

Output:Hello java

## Understanding javap tool

The **javap command** disassembles a class file. The javap command displays information about the fields,constructors and methods present in a class file.

### Syntax to use javap tool

Let's see how to use javap tool or command.

```
javap fully_class_name
```

### Example to use javap tool

```
javap java.lang.Object
```

#### Output:

```
Compiled from "Object.java"  
public class java.lang.Object {  
    public java.lang.Object();  
    public final native java.lang.Class<?> getClass();  
    public native int hashCode();  
    public boolean equals(java.lang.Object);  
    protected native java.lang.Object clone() throws java.lang.CloneNotSupportedException;  
    public java.lang.String toString();  
    public final native void notify();  
    public final native void notifyAll();  
    public final native void wait(long) throws java.lang.InterruptedException;  
    public final void wait(long, int) throws java.lang.InterruptedException;  
    public final void wait() throws java.lang.InterruptedException;  
    protected void finalize() throws java.lang.Throwable;
```

```
static {};
}
```

---

## Another example to use javap tool for your class

Let's use the javap command for our java file.

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

Now let's use the javap tool to disassemble the class file.

```
javap Simple
```

### Output:

```
Compiled from ".java"
class Simple {
    Simple();
    public static void main(java.lang.String[]);
}
```

---

## javap -c command

You can use the javap -c command to see disassembled code. The code that reflects the java bytecode.

```
javap -c Simple
```

### Output:

```
Compiled from ".java"
class Simple {
    Simple();
    Code:
        0: aload_0
        1: invokespecial #1                  // Method java/lang/Object."<init>":()V
        4: return

    public static void main(java.lang.String[]);
    Code:
        0: getstatic      #2                // Field java/lang/System.out:Ljava/io/PrintStream;
        3: ldc           #3                // String hello java
        5: invokevirtual #4                // Method java/io/PrintStream.println:(Ljava/lang/String;)V
        8: return
}
```

---

## Options of javap tool

The important options of javap tool are as follows.

Option	Description
-help	prints the help message.
-l	prints line number and local variable
-c	disassembles the code
-s	prints internal type signature
-sysinfo	shows system info (path, size, date, MD5 hash)

-constants	shows static final constants
-version	shows version information

## Creating a program that works as javap tool

Following methods of **java.lang.Class** class can be used to display the metadata of a class.

Method	Description
public Field[] getDeclaredFields()throws SecurityException	returns an array of Field objects reflecting all the fields declared by the class or interface represented by this Class object.
public Constructor[] getDeclaredConstructors()throws SecurityException	returns an array of Constructor objects reflecting all the constructors declared by the class represented by this Class object.
public Method[] getDeclaredMethods()throws SecurityException	returns an array of Method objects reflecting all the methods declared by the class or interface represented by this Class object.

### Example of creating javap tool

Let's create a program that works like javap tool.

```
import java.lang.reflect.*;

public class MyJavap{
    public static void main(String[] args)throws Exception {
Class c=Class.forName(args[0]);

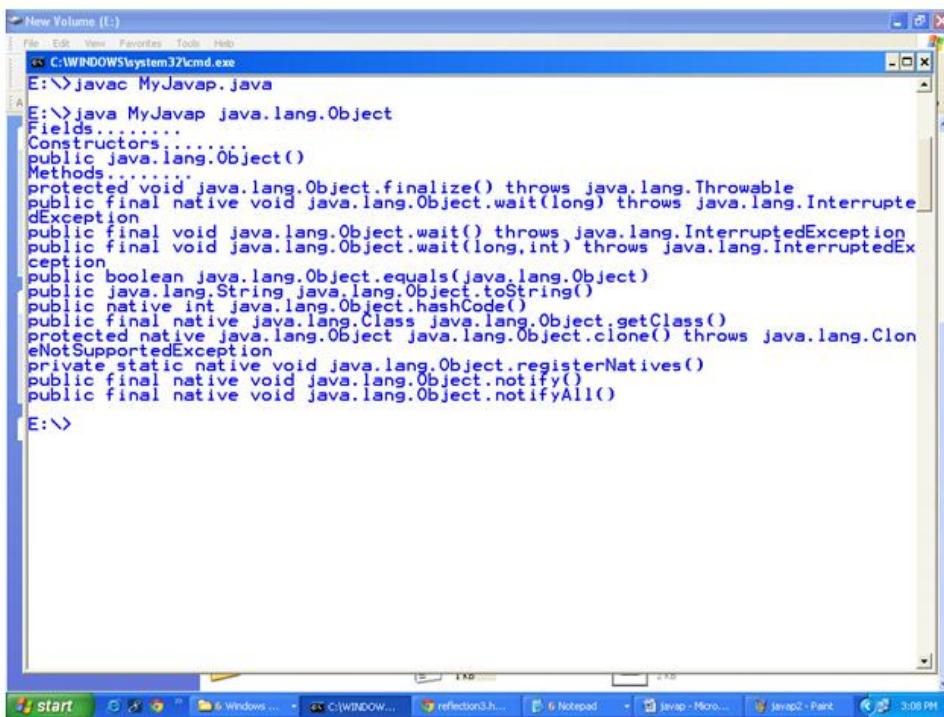
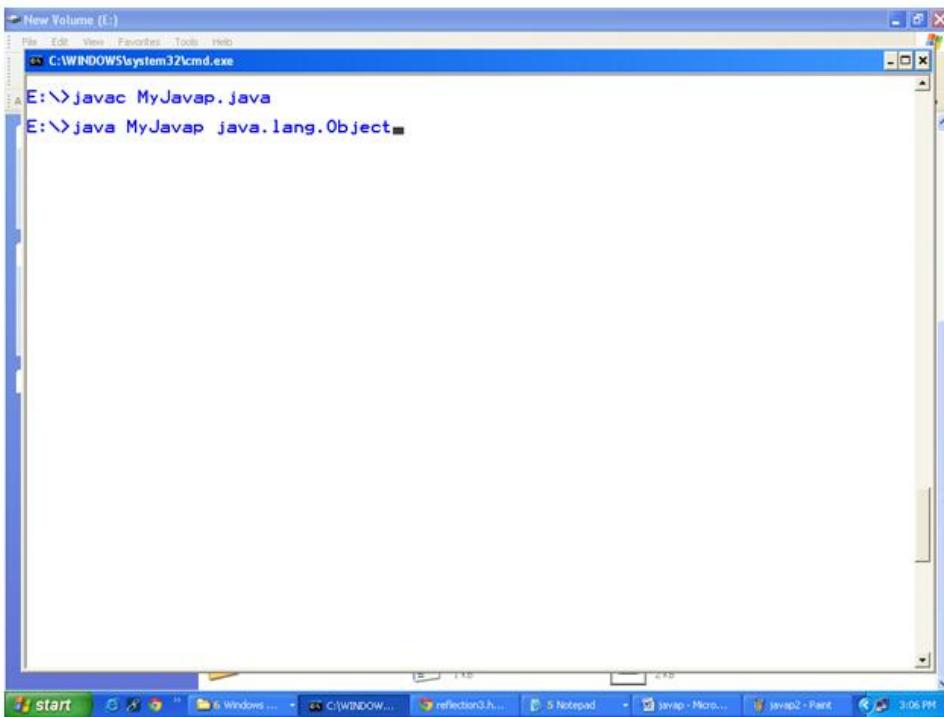
System.out.println("Fields.....");
Field f[]=c.getDeclaredFields();
for(int i=0;i<f.length;i++)
System.out.println(f[i]);

System.out.println("Constructors.....");
Constructor con[]=c.getDeclaredConstructors();
for(int i=0;i<con.length;i++)
System.out.println(con[i]);

System.out.println("Methods.....");
Method m[]=c.getDeclaredMethods();
for(int i=0;i<m.length;i++)
System.out.println(m[i]);
    }
}
```

At runtime, you can get the details of any class, it may be user-defined or pre-defined class.

### Output:



## Creating your own appletviewer

As you know well that appletviewer tool creates a frame and displays the output of applet in the frame. You can also create your frame and display the applet output.

### Example that works like appletviewer tool

Let's see the simple example that works like appletviewer tool. This example displays applet on the frame.

```
import java.applet.Applet;
import java.awt.Frame;
import java.awt.Graphics;

public class MyViewer extends Frame{
    public static void main(String[] args) throws Exception{
        Class c=Class.forName(args[0]);
```

```

MyViewer v=new MyViewer();
v.setSize(400,400);
v.setLayout(null);
v.setVisible(true);

Applet a=(Applet)c.newInstance();
a.start();
Graphics g=v.getGraphics();
a.paint(g);
a.stop();

}

}

//simple program of applet

import java.applet.Applet;
import java.awt.Graphics;

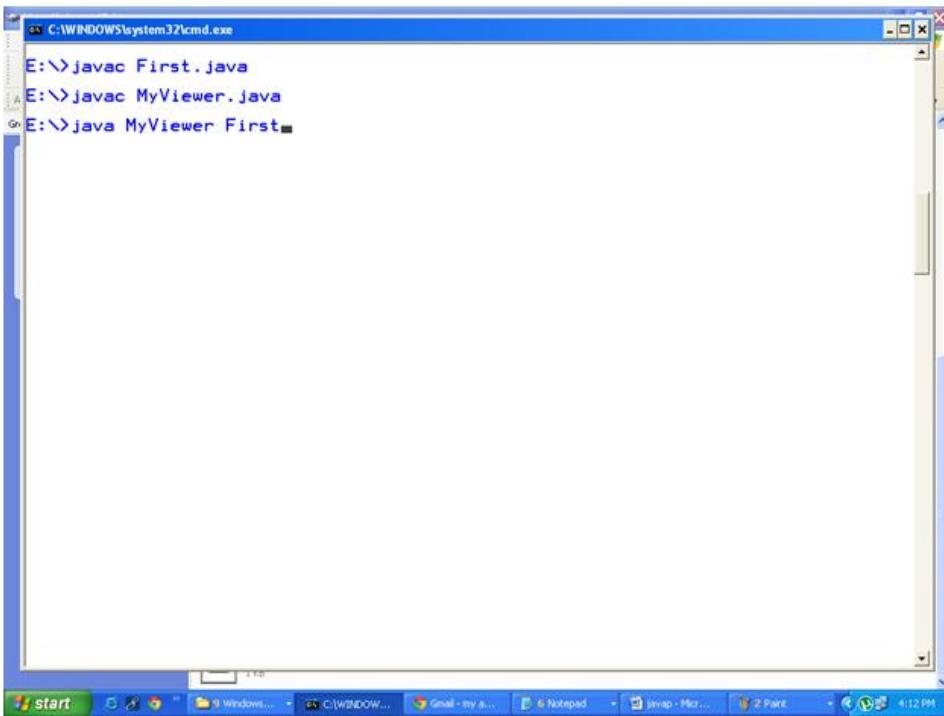
public class First extends Applet{

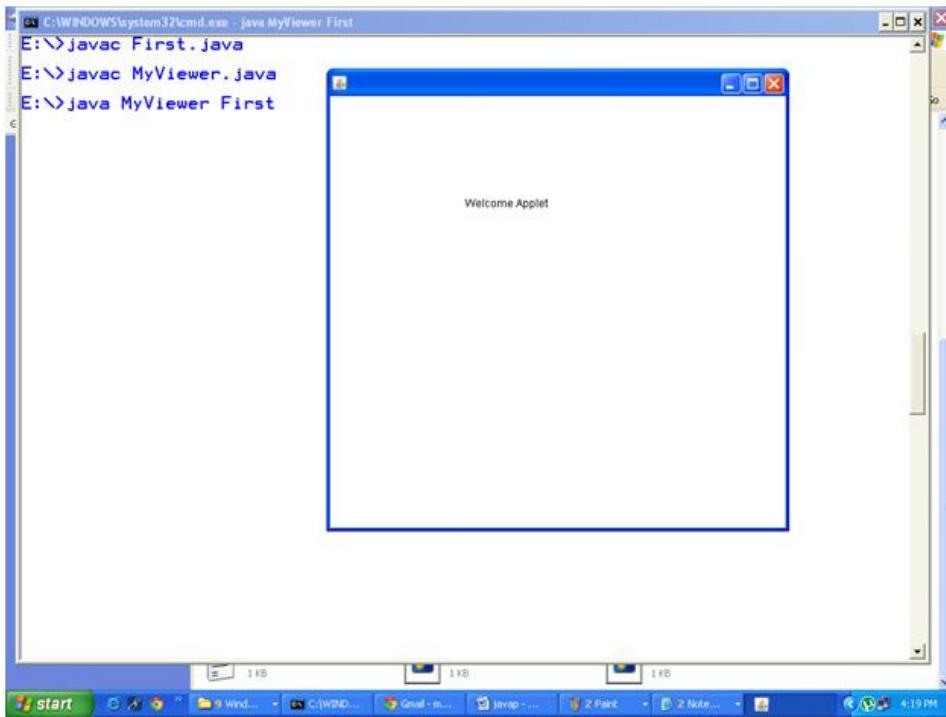
public void paint(Graphics g){g.drawString("Welcome",50, 50);}

}

```

## Output:





## How to call private method from another class in java

You can call the private method from outside the class by changing the runtime behaviour of the class.

By the help of **java.lang.Class** class and **java.lang.reflect.Method** class, we can call private method from any other class.

## Required methods of Method class

**1) public void setAccessible(boolean status) throws SecurityException** sets the accessibility of the method.

**2) public Object invoke(Object method, Object... args) throws IllegalAccessException, IllegalArgumentException, InvocationTargetException** is used to invoke the method.

Required method of Class class

**1) public Method getDeclaredMethod(String name,Class[] parameterTypes) throws**

**NoSuchMethodException, SecurityException:** returns a Method object that reflects the specified declared method of the class or interface represented by this Class object.

Example of calling private method from another class

Let's see the simple example to call private method from another class.

File: A.java

```
public class A {  
    private void message(){System.out.println("hello java"); }  
}
```

File: MethodCall.java

```
import java.lang.reflect.Method;
public class MethodCall{
public static void main(String[] args) throws Exception{

    Class c = Class.forName("A");
    Object o= c.newInstance();
    System.out.println(o);
}
}
```

```

Method m =c.getDeclaredMethod("message", null);
m.setAccessible(true);
m.invoke(o, null);
}
}

Output:hello java

```

## Another example to call parameterized private method from another class

Let's see the example to call parameterized private method from another class

*File: A.java*

```

class A{
private void cube(int n){System.out.println(n*n*n);}
}

```

*File: M.java*

```

import java.lang.reflect.*;
class M{
public static void main(String args[])throws Exception{
Class c=A.class;
Object obj=c.newInstance();

Method m=c.getDeclaredMethod("cube",new Class[]{int.class});
m.setAccessible(true);
m.invoke(obj,4);
}}

```

Output:64

## Java Date

The `java.util`, `java.sql` and `java.text` packages contains classes for representing date and time. Following classes are important for dealing with date in java.

- `java.util.Date` class
- `java.sql.Date` class
- `java.text.DateFormat` class
- `java.text.SimpleDateFormat` class
- `java.util.Calendar` class
- `java.util.GregorianCalendar` class
- `java.sql.Time` class
- `java.sql.Timestamp` class
- `java.util.TimeZone` class

## Get Current Date

There are 4 ways to print current date in java.

### 1st way:

```

java.util.Date date=new java.util.Date();
System.out.println(date);

```

Output:

Wed Mar 27 08:22:02 IST 2015

### 2nd way:

```

long millis=System.currentTimeMillis();
java.util.Date date=new java.util.Date(millis);
System.out.println(date);

```

Output:

Wed Mar 27 08:22:02 IST 2015

### **3rd way:**

```

long millis=System.currentTimeMillis();
java.sql.Date date=new java.sql.Date(millis);
System.out.println(date);

```

Output:

2015-03-27

### **4th way:**

```

Date date=java.util.Calendar.getInstance().getTime();
System.out.println(date);

```

Output:

Wed Mar 27 08:22:02 IST 2015

## java.util.Date

The `java.util.Date` class represents date and time in java. It provides constructors and methods to deal with date and time in java.

The `java.util.Date` class implements `Serializable`, `Cloneable` and `Comparable<Date>` interface. It is inherited by `java.sql.Date`, `java.sql.Time` and `java.sql.Timestamp` interfaces.

After `Calendar` class, most of the constructors and methods of `java.util.Date` class has been deprecated. Here, we are not giving list of any deprecated constructor and method.

### java.util.Date Constructors

No.	Constructor	Description
1)	<code>Date()</code>	Creates a date object representing current date and time.
2)	<code>Date(long milliseconds)</code>	Creates a date object for the given milliseconds since January 1, 1970, 00:00:00 GMT.

### java.util.Date Methods

No.	Method	Description
1)	<code>boolean after(Date date)</code>	tests if current date is after the given date.
2)	<code>boolean before(Date date)</code>	tests if current date is before the given date.
3)	<code>Object clone()</code>	returns the clone object of current date.
4)	<code>int compareTo(Date date)</code>	compares current date with given date.
5)	<code>boolean equals(Date date)</code>	compares current date with given date for equality.
6)	<code>static Date from(Instant instant)</code>	returns an instance of Date object from Instant date.
7)	<code>long getTime()</code>	returns the time represented by this date object.
8)	<code>int hashCode()</code>	returns the hash code value for this date object.
9)	<code>void setTime(long time)</code>	changes the current date and time to given time.

10)	Instant toInstant()	converts current date into Instant object.
11)	String toString()	converts this date into Instant object.

## java.util.Date Example

Let's see the example to print date in java using java.util.Date class.

### 1st way:

```
java.util.Date date=new java.util.Date();
System.out.println(date);
```

Output:

```
Wed Mar 27 08:22:02 IST 2015
```

### 2nd way:

```
long millis=System.currentTimeMillis();
java.util.Date date=new java.util.Date(millis);
System.out.println(date);
```

Output:

```
Wed Mar 27 08:22:02 IST 2015
```

## java.sql.Date

The java.sql.Date class represents only date in java. It inherits java.util.Date class.

The java.sql.Date instance is widely used in JDBC because it represents the date that can be stored in database.

Some constructors and methods of java.sql.Date class has been deprecated. Here, we are not giving list of any deprecated constructor and method.

## java.sql.Date Constructor

No.	Constructor	Description
1)	Date(long milliseconds)	Creates a sql date object for the given milliseconds since January 1, 1970, 00:00:00 GMT.

## java.sql.Date Methods

No.	Method	Description
1)	void setTime(long time)	changes the current sql date to given time.
2)	Instant toInstant()	converts current sql date into Instant object.
3)	LocalDate toLocalDate()	converts current sql date into LocalDate object.
4)	String toString()	converts this sql date object to a string.
5)	static Date valueOf(LocalDate date)	returns sql date object for the given LocalDate.
6)	static Date valueOf(String date)	returns sql date object for the given String.

## java.sql.Date Example: get current date

Let's see the example to **print date in java** using java.sql.Date class.

```
public class SQLDateExample {
    public static void main(String[] args) {
        long millis=System.currentTimeMillis();
```

```
    java.sql.Date date=new java.sql.Date(millis);
    System.out.println(date);
}
}
```

Output:

2015-03-30

## Java String to java.sql.Date Example

Let's see the example to **convert string into java.sql.Date** using valueOf() method.

```
import java.sql.Date;
public class StringToSQLDateExample {
public static void main(String[] args) {
String str="2015-03-31";
Date date=Date.valueOf(str);//converting string into sql date
System.out.println(date);
}
}
```

Output:

2015-03-31

## Java Date Format

There are two classes for formatting date in java: DateFormat and SimpleDateFormat.

The `java.text.DateFormat` class provides various methods to format and parse date and time in java in language independent manner. The `DateFormat` class is an abstract class. `java.text.Format` is the parent class and `java.text.SimpleDateFormat` is the subclass of `java.text.DateFormat` class.

In java, converting date into string is called formatting and vice-versa parsing. In other words, *formatting means date to string and parsing means string to date*.

## java.text.DateFormat Fields

```
protected Calendar calendar
protected NumberFormat numberFormat
public static final int ERA_FIELD
public static final int YEAR_FIELD
public static final int MONTH_FIELD
public static final int DATE_FIELD
public static final int HOUR_OF_DAY1_FIELD
public static final int HOUR_OF_DAY0_FIELD
public static final int MINUTE_FIELD
public static final int SECOND_FIELD
public static final int MILLISECOND_FIELD
public static final int DAY_OF_WEEK_FIELD
public static final int DAY_OF_YEAR_FIELD
public static final int DAY_OF_WEEK_IN_MONTH_FIELD
public static final int WEEK_OF_YEAR_FIELD
public static final int WEEK_OF_MONTH_FIELD
public static final int AM_PM_FIELD
public static final int HOUR1_FIELD
public static final int HOUR0_FIELD
public static final int TIMEZONE_FIELD
public static final int FULL
```

```

public static final int LONG
public static final int MEDIUM
public static final int SHORT
public static final int DEFAULT

```

## java.text.DateFormat Methods

No. Public Method	Description
1) final String format(Date date)	converts given Date object into string.
2) Date parse(String source)throws ParseException	converts string into Date object.
3) static final DateFormat getTimeInstance()	returns time formatter with default formatting style for the default locale.
4) static final DateFormat getTimeInstance(int style)	returns time formatter with the given formatting style for the default locale.
5) static final DateFormat getTimeInstance(int style, Locale locale)	returns time formatter with the given formatting style for the given locale.
6) static final DateFormat getDateInstance()	returns date formatter with default formatting style for the default locale.
7) static final DateFormat getDateInstance(int style)	returns date formatter with the given formatting style for the default locale.
8) static final DateFormat getDateInstance(int style, Locale locale)	returns date formatter with the given formatting style for the given locale.
9) static final DateFormat getDateTimeInstance()	returns date/time formatter with default formatting style for the default locale.
10) static final DateFormat getDateTimeInstance(int dateStyle,int timeStyle)	returns date/time formatter with the given date formatting style and time formatting style for the default locale.
11) static final DateFormat getDateTimeInstance(int dateStyle, int timeStyle, Locale locale)	returns date/time formatter with the given date formatting style and time formatting style for the given locale.
12) static final DateFormat getInstance()	returns date/time formatter with short formatting style for date and time.
13) static Locale[] getAvailableLocales()	returns an array of available locales.
14) Calendar getCalendar()	returns an instance of Calendar for this DateFormat instance.
15) NumberFormat getNumberFormat()	returns an instance of NumberFormat for this DateFormat instance.
16) TimeZone getTimeZone()	returns an instance of TimeZone for this DateFormat instance.

## Java DateFormat Example: Date to String

Let's see the simple example to **format date and time in java** using java.text.DateFormat class.

```

import java.text.DateFormat;
import java.util.Date;
public class DateFormatExample {
    public static void main(String[] args) {
        Date currentDate = new Date();
        System.out.println("Current Date: "+currentDate);
        String dateToStr = DateFormat.getInstance().format(currentDate);
        System.out.println("Date Format using getInstance(): "+dateToStr);
    }
}

```

Output:

```

Current Date: Tue Mar 31 14:37:23 IST 2015
Date Format using getInstance(): 31/3/15 2:37 PM

```

Let's see the full example to **format date and time in java** using java.text.DateFormat class.

```

import java.text.DateFormat;
import java.util.Date;
public class DateFormatExample2 {
    public static void main(String[] args) {
        Date currentDate = new Date();
        System.out.println("Current Date: "+currentDate);

        String dateToStr = DateFormat.getInstance().format(currentDate);
        System.out.println("Date Format using getInstance(): "+dateToStr);

        dateToStr = DateFormat.getDateInstance().format(currentDate);
        System.out.println("Date Format using getDateInstance(): "+dateToStr);

        dateToStr = DateFormat.getTimeInstance().format(currentDate);
        System.out.println("Date Format using getTimeInstance(): "+dateToStr);

        dateToStr = DateFormat.getDateTimeInstance().format(currentDate);
        System.out.println("Date Format using getDateTimeInstance(): "+dateToStr);

        dateToStr = DateFormat.getTimeInstance(DateFormat.SHORT).format(currentDate);
        System.out.println("Date Format using getTimeInstance(DateFormat.SHORT): "+dateToStr);

        dateToStr = DateFormat.getTimeInstance(DateFormat.MEDIUM).format(currentDate);
        System.out.println("Date Format using getTimeInstance(DateFormat.MEDIUM): "+dateToStr);

        dateToStr = DateFormat.getTimeInstance(DateFormat.LONG).format(currentDate);
        System.out.println("Date Format using getTimeInstance(DateFormat.LONG): "+dateToStr);

        dateToStr = DateFormat.getDateTimeInstance(DateFormat.LONG,DateFormat.SHORT).format(currentDate);
        System.out.println("Date Format using getDateTimeInstance(DateFormat.LONG,DateFormat.SHORT): "+dateToStr);
    }
}

```

Output:

```

Current Date: Tue Mar 31 14:37:23 IST 2015
Date Format using getInstance(): 31/3/15 2:37 PM
Date Format using getDateInstance(): 31 Mar, 2015
Date Format using getTimeInstance(): 2:37:23 PM
Date Format using getDateTimeInstance(): 31 Mar, 2015 2:37:23 PM
Date Format using getTimeInstance(DateFormat.SHORT): 2:37 PM
Date Format using getTimeInstance(DateFormat.MEDIUM): 2:37:23 PM
Date Format using getTimeInstance(DateFormat.LONG): 2:37:23 PM IST
Date Format using getDateTimeInstance(DateFormat.LONG,DateFormat.SHORT): 31 March, 2015 2:37 PM

```

## Java DateFormat Example: String to Date

Let's see the simple example to **convert string into date** using `java.text.DateFormat` class.

```

import java.text.DateFormat;
import java.util.Date;
public class DateFormatExample3 {
    public static void main(String[] args) throws Exception {
        Date d = DateFormat.getDateInstance().parse("31 Mar, 2015");
        System.out.println("Date is: "+d);
    }
}

```

Output:

```
Date is: Tue Mar 31 00:00:00 IST 2015
```

# Java SimpleDateFormat

The `java.text.SimpleDateFormat` class provides methods to format and parse date and time in java. The `SimpleDateFormat` is a concrete class for formatting and parsing date which inherits `java.text.DateFormat` class.

Notice that *formatting means converting date to string* and *parsing means converting string to date*.

## Java SimpleDateFormat Example: Date to String

Let's see the simple example to **format date in java** using `java.text.SimpleDateFormat` class.

```
import java.text.SimpleDateFormat;
import java.util.Date;
public class SimpleDateFormatExample {
public static void main(String[] args) {
Date date = new Date();
SimpleDateFormat formatter = new SimpleDateFormat("dd/MM/yyyy");
String strDate= formatter.format(date);
System.out.println(strDate);
}
}
```

**Test it Now**

Output:

13/04/2015

 **Note:** M (capital M) represents month and m (small m) represents minute in java.

Let's see the full example to **format date and time in java** using `java.text.SimpleDateFormat` class.

```
import java.text.ParseException;
import java.text.SimpleDateFormat;
import java.util.Date;
import java.util.Locale;
public class SimpleDateFormatExample2 {
public static void main(String[] args) {
Date date = new Date();
SimpleDateFormat formatter = new SimpleDateFormat("MM/dd/yyyy");
String strDate = formatter.format(date);
System.out.println("Date Format with MM/dd/yyyy : "+strDate);

formatter = new SimpleDateFormat("dd-M-yyyy hh:mm:ss");
strDate = formatter.format(date);
System.out.println("Date Format with dd-M-yyyy hh:mm:ss : "+strDate);

formatter = new SimpleDateFormat("dd MMMM yyyy");
strDate = formatter.format(date);
System.out.println("Date Format with dd MMMM yyyy : "+strDate);

formatter = new SimpleDateFormat("dd MMMM yyyy zzzz");
strDate = formatter.format(date);
System.out.println("Date Format with dd MMMM yyyy zzzz : "+strDate);

formatter = new SimpleDateFormat("E, dd MMM yyyy HH:mm:ss z");
strDate = formatter.format(date);
System.out.println("Date Format with E, dd MMM yyyy HH:mm:ss z : "+strDate);
}
```

```
}
```

**Test it Now**

Output:

```
Date Format with MM/dd/yyyy : 04/13/2015
Date Format with dd-M-yyyy hh:mm:ss : 13-4-2015 10:59:26
Date Format with dd MMMM yyyy : 13 April 2015
Date Format with dd MMMM yyyy zzzz : 13 April 2015 India Standard Time
Date Format with E, dd MMM yyyy HH:mm:ss z : Mon, 13 Apr 2015 22:59:26 IST
```

## Java SimpleDateFormat Example: String to Date

Let's see the simple example to **convert string into date** using `java.text.SimpleDateFormat` class.

```
import java.text.ParseException;
import java.text.SimpleDateFormat;
import java.util.Date;
public class SimpleDateFormatExample3 {
public static void main(String[] args) {
SimpleDateFormat formatter = new SimpleDateFormat("dd/MM/yyyy");
try {
Date date = formatter.parse("31/03/2015");
System.out.println("Date is: "+date);
} catch (ParseException e) {e.printStackTrace();}
}
}
```

**Test it Now**

Output:

```
Date is: Tue Mar 31 00:00:00 IST 2015
```

## Java String to int

We can convert **String to int in java** using `Integer.parseInt()` method.

### Scenario

It is generally used if we have to perform mathematical operations on the string that contains number. Whenever we get data from textfield or pre, entered data is received as a string. If entered data is integer, we need to convert string to int. To do so, we use `Integer.parseInt()` method.

### Signature

The `parseInt()` is the static method of `Integer` class. The **signature** of `parseInt()` method is given below:

```
public static int parseInt(String s)
```

## Java String to int Example

Let's see the simple code to convert string to int in java.

```
int i=Integer.parseInt("200");
```

Let's see the simple example of converting String to int in java.

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

```
String s="200";
int i=Integer.parseInt(s);
System.out.println(s+100);//200100 because + is string concatenation operator
System.out.println(i+100);//300 because + is binary plus operator
}}
```

**Test it Now**

Output:

```
200100
300
```

## Java int to String

We can convert **int to String in java** using *String.valueOf()* and *Integer.toString()* methods.

### Scenario

It is generally used if we have to display number in textfield because everything is displayed as a string in form.

### 1) String.valueOf()

The *String.valueOf()* method converts int to String. The *valueOf()* is the static method of *String* class. The **signature** of *valueOf()* method is given below:

```
public static String valueOf(int i)
```

### Java int to String Example using String.valueOf()

Let's see the simple code to convert int to String in java.

```
int i=10;
String s=String.valueOf(i);//Now it will return "10"
```

Let's see the simple example of converting String to int in java.

```
public class IntToStringExample1{
public static void main(String args[]){
int i=200;
String s=String.valueOf(i);
System.out.println(i+100);//300 because + is binary plus operator
System.out.println(s+100);//200100 because + is string concatenation operator
}}
```

**Test it Now**

Output:

```
300
200100
```

### 2) Integer.toString()

The *Integer.toString()* method converts int to String. The *toString()* is the static method of *Integer* class. The **signature** of *toString()* method is given below:

```
public static String toString(int i)
```

### Java int to String Example using Integer.toString()

Let's see the simple code to convert int to String in java using *Integer.toString()* method.

```
int i=10;
String s=Integer.toString(i); //Now it will return "10"
```

Let's see the simple example of converting String to int in java.

```
public class IntToStringExample2{
public static void main(String args[]){
int i=200;
String s=Integer.toString(i);
System.out.println(i+100); //300 because + is binary plus operator
System.out.println(s+100); //200100 because + is string concatenation operator
}}
```

**Test it Now**

Output:

```
300
200100
```

## Java String to long

We can convert **String to long in java** using *Long.parseLong()* method.

### Scenario

It is generally used if we have to perform mathematical operations on the string that contains long number. Whenever we get data from textfield or pre, entered data is received as a string. If entered data is long, we need to convert string to long. To do so, we use *Long.parseLong()* method.

### Signature

The *parseLong()* is the static method of *Long* class. The **signature** of *parseLong()* method is given below:

```
public static long parseLong(String s)
```

## Java String to long Example

Let's see the simple code to convert string to long in java.

```
long l=Long.parseLong("200");
```

Let's see the simple example of converting String to long in java.

```
public class StringToLongExample{
public static void main(String args[]){
String s="9990449935";
long l=Long.parseLong(s);
System.out.println(l);
}}
```

**Test it Now**

Output:

```
9990449935
```

## Java long to String

We can convert **long to String in java** using *String.valueOf()* and *Long.toString()* methods.

## Scenario

It is generally used if we have to display long number in textfield because everything is displayed as a string in form.

### 1) String.valueOf()

The *String.valueOf()* is an overloaded method. It can be used to convert long to String. The *valueOf()* is the static method of *String* class. The **signature** of *valueOf()* method is given below:

```
public static String valueOf(long i)
```

### Java long to String Example using String.valueOf()

Let's see the simple code to convert int to String in java.

```
long i=9993939399L;//L is the suffix for long  
String s=String.valueOf(i);//Now it will return "9993939399"
```

Let's see the simple example of converting String to int in java.

```
public class LongToStringExample1{  
    public static void main(String args[]){  
        long i=9993939399L;  
        String s=String.valueOf(i);  
        System.out.println(s);  
    }  
}
```

**Test it Now**

Output:

9993939399

### 2) Long.toString()

The *Long.toString()* method converts long to String. The *toString()* is the static method of *Long* class. The **signature** of *toString()* method is given below:

```
public static String toString(long i)
```

### Java long to String Example using Long.toString()

Let's see the simple code to convert int to String in java using *Long.toString()* method.

```
long i=9993939399L;  
String s=Long.toString(i);//Now it will return "9993939399"
```

Let's see the simple example of converting String to int in java.

```
public class LongToStringExample2{  
    public static void main(String args[]){  
        long i=9993939399L;  
        String s=Long.toString(i);  
        System.out.println(s);  
    }  
}
```

**Test it Now**

Output:

9993939399

# Java String to float

We can convert **String to float in java** using *Float.parseFloat()* method.

## Scenario

It is generally used if we have to perform mathematical operations on the string that contains float number. Whenever we get data from textfield or pre, entered data is received as a string. If entered data is float, we need to convert string to float. To do so, we use *Float.parseFloat()* method.

## Signature

The *parseFloat()* is the static method of *Float* class. The **signature** of *parseFloat()* method is given below:

```
public static int parseFloat(String s)
```

## Java String to float Example

Let's see the simple code to convert string to float in java.

```
float f=Float.parseFloat("23.6");
```

Let's see the simple example of converting String to float in java.

```
public class StringToFloatExample{  
    public static void main(String args[]){  
        String s="23.6";  
        float f=Float.parseFloat("23.6");  
        System.out.println(f);  
    }  
}
```

**Test it Now**

Output:

```
23.6
```

# Collections in Java

**Collections in java** is a framework that provides an architecture to store and manipulate the group of objects.

All the operations that you perform on a data such as searching, sorting, insertion, manipulation, deletion etc. can be performed by Java Collections.

Java Collection simply means a single unit of objects. Java Collection framework provides many interfaces (Set, List, Queue, Deque etc.) and classes (ArrayList, Vector, LinkedList, PriorityQueue, HashSet, LinkedHashSet, TreeSet etc.).

## What is Collection in java

Collection represents a single unit of objects i.e. a group.

## What is framework in java

- provides readymade architecture.
- represents set of classes and interface.
- is optional.

## What is Collection framework

Collection framework represents a unified architecture for storing and manipulating group of objects. It has:

1. Interfaces and its implementations i.e. classes
2. Algorithm

## Java ArrayList class

- o Java ArrayList class uses a dynamic array for storing the elements. It extends AbstractList class and implements List interface.
- o Java ArrayList class can contain duplicate elements.
- o Java ArrayList class maintains insertion order.
- o Java ArrayList class is non synchronized.
- o Java ArrayList allows random access because array works at the index basis.
- o In Java ArrayList class, manipulation is slow because a lot of shifting needs to be occurred if any element is removed from the array list.

## Java Non-generic Vs Generic Collection

Java collection framework was non-generic before JDK 1.5. Since 1.5, it is generic.

Java new generic collection allows you to have only one type of object in collection. Now it is type safe so typecasting is not required at run time.

Let's see the old non-generic example of creating java collection.

```
ArrayList al=new ArrayList(); //creating old non-generic arraylist
```

Let's see the new generic example of creating java collection.

```
ArrayList<String> al=new ArrayList(); //creating new generic arraylist
```

In generic collection, we specify the type in angular braces. Now ArrayList is forced to have only specified type of objects in it. If you try to add another type of object, it gives *compile time error*.

For more information of java generics, click here [Java Generics Tutorial](#).

## Example of Java ArrayList class

```
import java.util.*;
class TestCollection1{
    public static void main(String args[]){
        ArrayList al=new ArrayList(); //creating arraylist
        al.add("Ravi"); //adding object in arraylist
        al.add("Vijay");
        al.add("Ravi");
        al.add("Ajay");

        Iterator itr=al.iterator(); //getting Iterator from arraylist to traverse elements
        while(itr.hasNext()){
            System.out.println(itr.next());
        }
    }
}
```

**Test it Now**

```
Ravi
Vijay
Ravi
```

## Two ways to iterate the elements of collection in java

1. By Iterator interface.
2. By for-each loop.

In the above example, we have seen traversing ArrayList by Iterator. Let's see the example to traverse ArrayList elements using for-each loop.

### Iterating the elements of Collection by for-each loop

```
import java.util.*;
class TestCollection2{
    public static void main(String args[]){
        ArrayList al=new ArrayList();
        al.add("Ravi");
        al.add("Vijay");
        al.add("Ravi");
        al.add("Ajay");
        for(String obj:al)
            System.out.println(obj);
    }
}
```

**Test it Now**

```
Ravi
Vijay
Ravi
Ajay
```

## User-defined class objects in Java 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.*;
public class TestCollection3{
    public static void main(String args[]){
        //Creating user-defined class objects
        Student s1=new Student(101,"Sonoo",23);
        Student s2=new Student(102,"Ravi",21);
        Student s3=new Student(103,"Hanumat",25);

        ArrayList al=new ArrayList(); //creating arraylist
        al.add(s1); //adding Student class object
        al.add(s2);
        al.add(s3);

        Iterator itr=al.iterator();
        //traversing elements of ArrayList object
        while(itr.hasNext()){
            Student st=(Student)itr.next();
```

```
        System.out.println(st.rollno+" "+st.name+" "+st.age);
    }
}
}
```

**Test it Now**

```
101 Sonoo 23
102 Ravi 21
103 Hanumat 25
```

---

## Example of addAll(Collection c) method

```
import java.util.*;
class TestCollection4{
    public static void main(String args[]){
        ArrayList al=new ArrayList();
        al.add("Ravi");
        al.add("Vijay");
        al.add("Ajay");

        ArrayList al2=new ArrayList();
        al2.add("Sonoo");
        al2.add("Hanumat");

        al.addAll(al2);

        Iterator itr=al.iterator();
        while(itr.hasNext()){
            System.out.println(itr.next());
        }
    }
}
```

**Test it Now**

```
Ravi
Vijay
Ajay
Sonoo
Hanumat
```

---

## Example of removeAll() method

```
import java.util.*;
class TestCollection5{
    public static void main(String args++){
        ArrayList al=new ArrayList();
        al.add("Ravi");
        al.add("Vijay");
        al.add("Ajay");

        ArrayList al2=new ArrayList();
        al2.add("Ravi");
        al2.add("Hanumat");

        al.removeAll(al2);

    }
}
```

```

System.out.println("iterating the elements after removing the elements of al2...");
Iterator itr=al.iterator();
while(itr.hasNext()){
    System.out.println(itr.next());
}
}

```

**Test it Now**

```

iterating the elements after removing the elements of al2...
Vijay
Ajay

```

## Example of retainAll() method

```

import java.util.*;
class TestCollection6{
    public static void main(String args[]){
        ArrayList al=new ArrayList();
        al.add("Ravi");
        al.add("Vijay");
        al.add("Ajay");
        ArrayList al2=new ArrayList();
        al2.add("Ravi");
        al2.add("Hanumat");

        al.retainAll(al2);

        System.out.println("iterating the elements after retaining the elements of al2...");
        Iterator itr=al.iterator();
        while(itr.hasNext()){
            System.out.println(itr.next());
        }
    }
}

```

**Test it Now**

```

iterating the elements after retaining the elements of al2...
Ravi

```

## Java LinkedList class

- Java LinkedList class uses doubly linked list to store the elements. It extends the AbstractList class and implements List and Deque interfaces.
- Java LinkedList class can contain duplicate elements.
- Java LinkedList class maintains insertion order.
- Java LinkedList class is non synchronized.
- In Java LinkedList class, manipulation is fast because no shifting needs to be occurred.
- Java LinkedList class can be used as list, stack or queue.

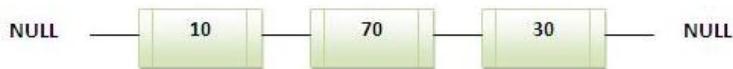


fig- doubly linked list

## Java LinkedList Example

```

import java.util.*;
public class TestCollection7{
    public static void main(String args[]){
        LinkedList al=new LinkedList();
        al.add("Ravi");
        al.add("Vijay");
        al.add("Ravi");
        al.add("Ajay");

        Iterator itr=al.iterator();
        while(itr.hasNext()){
            System.out.println(itr.next());
        }
    }
}
    
```

**Test it Now**

Output:Ravi

Vijay

Ravi

Ajay

## Difference between ArrayList and LinkedList

ArrayList and LinkedList both implements List interface and maintains insertion order. Both are non synchronized classes.

But there are many differences between ArrayList and LinkedList classes that are given below.

ArrayList	LinkedList
1) ArrayList internally uses <b>dynamic array</b> to store the elements.	LinkedList internally uses <b>doubly linked list</b> to store the elements.
2) Manipulation with ArrayList is <b>slow</b> because it internally uses array. If any element is removed from the array, all the bits are shifted in memory.	Manipulation with LinkedList is <b>faster</b> than ArrayList because it uses doubly linked list so no bit shifting is required in memory.
3) ArrayList class can <b>act as a list</b> only because it implements List only.	LinkedList class can <b>act as a list and queue</b> both because it implements List and Deque interfaces.
4) ArrayList is <b>better for storing and accessing</b> data.	LinkedList is <b>better for manipulating</b> data.

## Example of ArrayList and LinkedList in Java

Let's see a simple example where we are using ArrayList and LinkedList both.

```

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

```

List al=new ArrayList(); //creating arraylist
al.add("Ravi"); //adding object in arraylist
al.add("Vijay");
al.add("Ravi");
al.add("Ajay");

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

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

```

**Test it Now**

Output:

```

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

```

## Java List Interface

List Interface is the subinterface of Collection. It contains methods to insert and delete elements in index basis. It is a factory of ListIterator interface.

### Commonly used methods of List Interface:

1. public void add(int index, Object element);
2. public boolean addAll(int index, Collection c);
3. public Object get(int Index position);
4. public Object set(int index, Object element);
5. public Object remove(int index);
6. public ListIterator listIterator();
7. public ListIterator listIterator(int i);

## Java ListIterator Interface

ListIterator Interface is used to traverse the element in backward and forward direction.

### Commonly used methods of ListIterator Interface:

1. public boolean hasNext();
2. public Object next();
3. public boolean hasPrevious();
4. public Object previous();

### Example of ListIterator Interface:

```

import java.util.*;
public class TestCollection8{
public static void main(String args[]){
    ArrayList al=new ArrayList();
    al.add("Amit");
}

```

```

al.add("Vijay");
al.add("Kumar");
al.add(1,"Sachin");

System.out.println("element at 2nd position: "+al.get(2));

ListIterator itr=al.listIterator();

System.out.println("traversing elements in forward direction...");
while(itr.hasNext()){
System.out.println(itr.next());
}

System.out.println("traversing elements in backward direction...");
while(itr.hasPrevious()){
System.out.println(itr.previous());
}
}
}

```

#### Test it Now

```

Output:element at 2nd position: Vijay
traversing elements in forward direction...
Amit
Sachin
Vijay
Kumar
traversing elements in backward direction...
Kumar
Vijay
Sachin
Amit

```

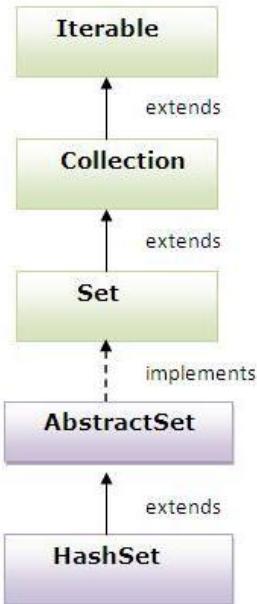
## Java HashSet class

- uses hashtable to store the elements. It extends AbstractSet class and implements Set interface.
- contains unique elements only.

### Difference between List and Set:

List can contain duplicate elements whereas Set contains unique elements only.

### Hierarchy of HashSet class:



### Example of HashSet class:

```

import java.util.*;
class TestCollection9{
    public static void main(String args[]){
        HashSet al=new HashSet();
        al.add("Ravi");
        al.add("Vijay");
        al.add("Ravi");
        al.add("Ajay");

        Iterator itr=al.iterator();
        while(itr.hasNext()){
            System.out.println(itr.next());
        }
    }
}
  
```

**Test it Now**

Output: Ajay

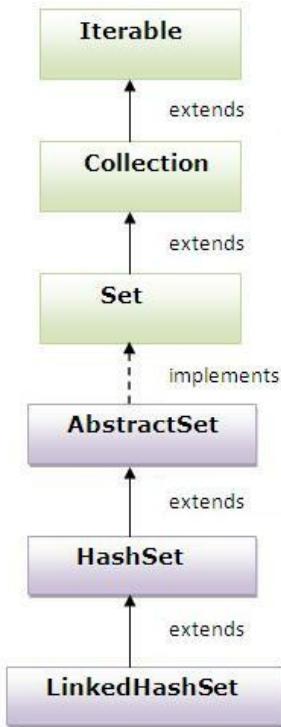
    Vijay

    Ravi

### Java LinkedHashSet class:

- contains unique elements only like HashSet. It extends HashSet class and implements Set interface.
- maintains insertion order.

### Hierarchy of LinkedHashSet class:



### Example of LinkedHashSet class:

```

import java.util.*;
class TestCollection10{
    public static void main(String args[]){
        LinkedHashSet al=new LinkedHashSet();
        al.add("Ravi");
        al.add("Vijay");
        al.add("Ravi");
        al.add("Ajay");

        Iterator itr=al.iterator();
        while(itr.hasNext()){
            System.out.println(itr.next());
        }
    }
}
    
```

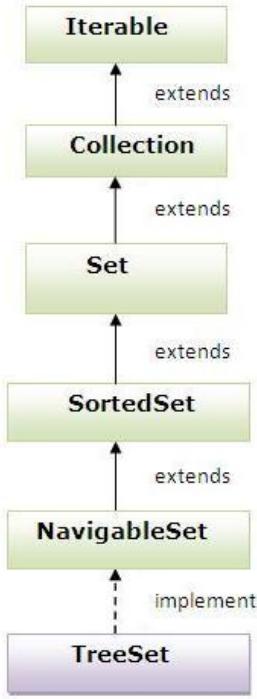
**Test it Now**

Output:>Ravi  
Vijay  
Ajay

## Java TreeSet class

- contains unique elements only like HashSet. The TreeSet class implements NavigableSet interface that extends the SortedSet interface.
- maintains ascending order.

### Hierarchy of TreeSet class:



### Example of TreeSet class:

```

import java.util.*;
class TestCollection11{
public static void main(String args[]){
    TreeSet al=new TreeSet();
    al.add("Ravi");
    al.add("Vijay");
    al.add("Ravi");
    al.add("Ajay");

    Iterator itr=al.iterator();
    while(itr.hasNext()){
        System.out.println(itr.next());
    }
}
}
  
```

**Test it Now**

Output:  
Ajay  
Ravi  
Vijay

## Java Queue Interface

The Queue interface basically orders the element in FIFO(First In First Out)manner.

### Methods of Queue Interface :

1. public boolean add(object);
2. public boolean offer(object);

3. public remove();
4. public poll();
5. public element();
6. public peek();

## PriorityQueue class:

The PriorityQueue class provides the facility of using queue. But it does not orders the elements in FIFO manner.

### Example of PriorityQueue:

```
import java.util.*;
class TestCollection12{
public static void main(String args[]){

PriorityQueue queue=new PriorityQueue();
queue.add("Amit");
queue.add("Vijay");
queue.add("Karan");
queue.add("Jai");
queue.add("Rahul");

System.out.println("head:"+queue.element());
System.out.println("head:"+queue.peek());

System.out.println("iterating the queue elements:");
Iterator itr=queue.iterator();
while(itr.hasNext()){
System.out.println(itr.next());
}

queue.remove();
queue.poll();

System.out.println("after removing two elements:");
Iterator itr2=queue.iterator();
while(itr2.hasNext()){
System.out.println(itr2.next());
}
}

}
```

#### Test it Now

```
Output:head:Amit
head:Amit
iterating the queue elements:
Amit
Jai
Karan
Vijay
Rahul
after removing two elements:
Karan
Rahul
Vijay
```

# Java Map Interface

A map contains values based on the key i.e. key and value pair. Each pair is known as an entry. Map contains only unique elements.

## Commonly used methods of Map interface:

1. **public Object put(object key, Object value):** is used to insert an entry in this map.
2. **public void putAll(Map map):** is used to insert the specified map in this map.
3. **public Object remove(object key):** is used to delete an entry for the specified key.
4. **public Object get(Object key):** is used to return the value for the specified key.
5. **public boolean containsKey(Object key):** is used to search the specified key from this map.
6. **public boolean containsValue(Object value):** is used to search the specified value from this map.
7. **public Set keySet():** returns the Set view containing all the keys.
8. **public Set entrySet():** returns the Set view containing all the keys and values.

---

## Entry

Entry is the subinterface of Map. So we will access it by Map.Entry name. It provides methods to get key and value.

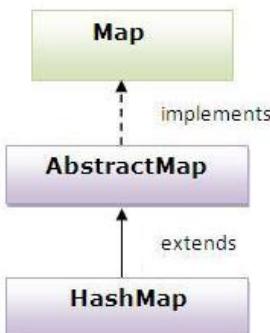
## Methods of Entry interface:

1. **public Object getKey():** is used to obtain key.
2. **public Object getValue():** is used to obtain value.

# Java HashMap class

- o A HashMap contains values based on the key. It implements the Map interface and extends AbstractMap class.
- o It contains only unique elements.
- o It may have one null key and multiple null values.
- o It maintains no order.

## Hierarchy of HashMap class:



## Example of HashMap class:

```
import java.util.*;
class TestCollection13{
    public static void main(String args[]){
        HashMap hm=new HashMap();
        hm.put(100,"Amit");
    }
}
```

```

hm.put(101,"Vijay");
hm.put(102,"Rahul");

for(Map.Entry m:hm.entrySet()){
    System.out.println(m.getKey()+" "+m.getValue());
}
}
}

```

### Test it Now

Output:102 Rahul  
100 Amit  
101 Vijay

---

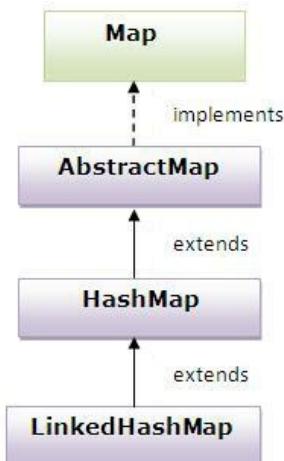
## What is difference between HashSet and HashMap?

HashSet contains only values whereas HashMap contains entry(key and value).

## Java LinkedHashMap class

- A LinkedHashMap contains values based on the key. It implements the Map interface and extends HashMap class.
- It contains only unique elements.
- It may have one null key and multiple null values.
- It is same as HashMap instead maintains insertion order.

### Hierarchy of LinkedHashMap class:



### Example of LinkedHashMap class:

```

import java.util.*;
class TestCollection14{
    public static void main(String args[]){
        LinkedHashMap hm=new LinkedHashMap();
        hm.put(100,"Amit");
    }
}

```

```

hm.put(101,"Vijay");
hm.put(102,"Rahul");

for(Map.Entry m:hm.entrySet()){
    System.out.println(m.getKey()+" "+m.getValue());
}
}
}

```

### Test it Now

Output:100 Amit

```

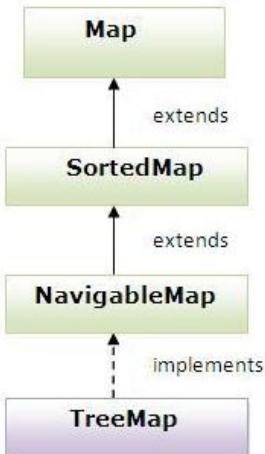
101 Vijay
103 Rahul

```

## Java TreeMap class

- A TreeMap contains values based on the key. It implements the NavigableMap interface and extends AbstractMap class.
- It contains only unique elements.
- It cannot have null key but can have multiple null values.
- It is same as HashMap instead maintains ascending order.

### Hierarchy of TreeMap class:



### Example of TreeMap class:

```

import java.util.*;
class TestCollection15{
    public static void main(String args[]){
        TreeMap hm=new TreeMap();

        hm.put(100,"Amit");
        hm.put(102,"Ravi");
        hm.put(101,"Vijay");
        hm.put(103,"Rahul");

        for(Map.Entry m:hm.entrySet()){
            System.out.println(m.getKey()+" "+m.getValue());
        }
    }
}

```

```
 }
}
```

#### Test it Now

```
Output:100 Amit
      101 Vijay
      102 Ravi
      103 Rahul
```

## What is difference between HashMap and TreeMap?

- 1) HashMap is can contain one null key. TreeMap can not contain any null key.
- 2) HashMap maintains no order. TreeMap maintains ascending order.

## Java Hashtable class

- o A Hashtable is an array of list. Each list is known as a bucket. The position of bucket is identified by calling the hashCode() method. A Hashtable contains values based on the key. It implements the Map interface and extends Dictionary class.
- o It contains only unique elements.
- o It may have not have any null key or value.
- o It is synchronized.

### Example of Hashtable:

```
import java.util.*;
class TestCollection16{
    public static void main(String args[]){
        Hashtable hm=new Hashtable();
        hm.put(100,"Amit");
        hm.put(102,"Ravi");
        hm.put(101,"Vijay");
        hm.put(103,"Rahul");
        for(Map.Entry m:hm.entrySet()){
            System.out.println(m.getKey()+" "+m.getValue());
        }
    }
}
```

#### Test it Now

```
Output:103 Rahul
      102 Ravi
      101 Vijay
      100 Amit
```

## Difference between HashMap and Hashtable

HashMap and Hashtable both are used to store data in key and value form. Both are using hashing technique to store unique keys.

But there are many differences between HashMap and Hashtable classes that are given below.

HashMap	Hashtable
1) HashMap is <b>non synchronized</b> . It is not-thread safe and can't be shared between many threads without proper synchronization code.	Hashtable is <b>synchronized</b> . It is thread-safe and can be shared with many threads.
2) HashMap <b>allows one null key and multiple null values</b>	Hashtable <b>doesn't allow any null key or value</b>
3) HashMap is a <b>new class introduced in JDK 1.2</b> .	Hashtable is a <b>legacy class</b> .
4) HashMap is <b>fast</b> .	Hashtable is <b>slow</b> .
5) We can make the HashMap as synchronized by calling this code Map m = Collections.synchronizedMap(hashMap);	Hashtable is internally synchronized and can't be unsynchronized.
6) HashMap is <b>traversed by Iterator</b> .	Hashtable is <b>traversed by Enumerator and Iterator</b> .
7) Iterator in HashMap is <b>fail-fast</b> .	Enumerator in Hashtable is <b>not fail-fast</b> .
8) HashMap inherits <b>AbstractMap</b> class.	Hashtable inherits <b>Dictionary</b> class.

## Sorting

We can sort the elements of:

1. String objects
2. Wrapper class objects
3. User-defined class objects

**Collections** class provides static methods for sorting the elements of collection. If collection elements are of Set type, we can use TreeSet. But We cannot sort the elements of List. Collections class provides methods for sorting the elements of List type elements.

### Method of Collections class for sorting List elements

**public void sort(List list):** is used to sort the elements of List. List elements must be of Comparable type.

**Note: String class and Wrapper classes implements the Comparable interface. So if you store the objects of string or wrapper classes, it will be Comparable.**

### Example of Sorting the elements of List that contains string objects

```
import java.util.*;
class TestSort1{
public static void main(String args[]){
    ArrayList al=new ArrayList();
    al.add("Viru");
    al.add("Saurav");
    al.add("Mukesh");
    al.add("Tahir");

    Collections.sort(al);
    Iterator itr=al.iterator();
    while(itr.hasNext()){
        System.out.println(itr.next());
    }
}
```

**Test it Now**

Output:Mukesh  
Saurav  
Tahir  
Viru

## Example of Sorting the elements of List that contains Wrapper class objects

```
import java.util.*;  
class TestSort2{  
public static void main(String args[]){  
  
ArrayList al=new ArrayList();  
al.add(Integer.valueOf(201));  
al.add(Integer.valueOf(101));  
al.add(230);//internally will be converted into objects as Integer.valueOf(230)  
  
Collections.sort(al);  
  
Iterator itr=al.iterator();  
while(itr.hasNext()){  
System.out.println(itr.next());  
}  
}  
}
```

**Test it Now**

Output:101  
201  
230

## Comparable interface

Comparable interface is used to order the objects of user-defined class. This interface is found in `java.lang` package and contains only one method named `compareTo(Object)`. It provides only single sorting sequence i.e. you can sort the elements based on single data member only. For instance it may be either rollno, name, age or anything else.

### Syntax:

**public int compareTo(Object obj):** is used to compare the current object with the specified object.

We can sort the elements of:

1. String objects
2. Wrapper class objects
3. User-defined class objects

**Collections** class provides static methods for sorting the elements of collection. If collection elements are of Set type, we can use `TreeSet`. But we cannot sort the elements of List. **Collections** class provides methods for sorting the elements of List type elements.

### Method of Collections class for sorting List elements

**public void sort(List list):** is used to sort the elements of List. List elements must be of Comparable type.



**Note:** String class and Wrapper classes implements the Comparable interface. So if you store the objects of string or wrapper classes, it will be Comparable.

Example of Sorting the elements of List that contains user-defined class objects on age basis

### Student.java

```
class Student implements Comparable{
int rollno;
String name;
int age;
Student(int rollno, String name, int age){
this.rollno=rollno;
this.name=name;
this.age=age;
}
public int compareTo(Object obj){
Student st=(Student)obj;
if(age==st.age)
return 0;
else if(age>st.age)
return 1;
else
return -1;
}
}
```

### Simple.java

```
import java.util.*;
import java.io.*;

class TestSort3{
public static void main(String args[]){
ArrayList al=new ArrayList();
al.add(new Student(101,"Vijay",23));
al.add(new Student(106,"Ajay",27));
al.add(new Student(105,"Jai",21));

Collections.sort(al);
Iterator itr=al.iterator();
while(itr.hasNext()){
Student st=(Student)itr.next();
System.out.println(st.rollno+" "+st.name+" "+st.age);
}
}
}
```

**Test it Now**

```
Output:105 Jai 21
      101 Vijay 23
      106 Ajay 27
```

# Comparator interface

**Comparator interface** is used to order the objects of user-defined class.

This interface is found in `java.util` package and contains 2 methods `compare(Object obj1, Object obj2)` and `equals(Object element)`.

It provides multiple sorting sequence i.e. you can sort the elements based on any data member. For instance it may be on rollno, name, age or anything else.

## Syntax of compare method

**public int compare(Object obj1, Object obj2):** compares the first object with second object.

---

**Collections** class provides static methods for sorting the elements of collection. If collection elements are of Set type, we can use `TreeSet`. But We cannot sort the elements of List. Collections class provides methods for sorting the elements of List type elements.

## Method of Collections class for sorting List elements

**public void sort(List list, Comparator c):** is used to sort the elements of List by the given comparator.

## Example of sorting the elements of List that contains user-defined class objects on the basis of age and name

In this example, we have created 4 java classes:

1. Student.java
2. AgeComparator.java
3. NameComparator.java
4. Simple.java

### **Student.java**

This class contains three fields rollno, name and age and a parameterized constructor.

```
class Student{  
    int rollno;  
    String name;  
    int age;  
    Student(int rollno, String name, int age){  
        this.rollno=rollno;  
        this.name=name;  
        this.age=age;  
    }  
}
```

### **AgeComparator.java**

This class defines comparison logic based on the age. If age of first object is greater than the second, we are returning positive value, it can be any one such as 1, 2 , 10 etc. If age of first object is less than the second object, we are returning negative value, it can be any negative value and if age of both objects are equal, we are returning 0.

```
import java.util.*;  
class AgeComparator implements Comparator{  
    public int Compare(Object o1, Object o2){  
        Student s1=(Student)o1;  
        Student s2=(Student)o2;  
  
        if(s1.age==s2.age)  
            return 0;
```

```

        else if(s1.age>s2.age)
        return 1;
    else
        return -1;
}
}

```

### **NameComparator.java**

This class provides comparison logic based on the name. In such case, we are using the compareTo() method of String class, which internally provides the comparison logic.

```

import java.util.*;
class NameComparator implements Comparator{
public int Compare(Object o1, Object o2){
Student s1=(Student)o1;
Student s2=(Student)o2;

return s1.name.compareTo(s2.name);
}
}

```

### **Simple.java**

In this class, we are printing the objects values by sorting on the basis of name and age.

```

import java.util.*;
import java.io.*;

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

ArrayList al=new ArrayList();
al.add(new Student(101,"Vijay",23));
al.add(new Student(106,"Ajay",27));
al.add(new Student(105,"Jai",21));

System.out.println("Sorting by Name...");

Collections.sort(al,new NameComparator());
Iterator itr=al.iterator();
while(itr.hasNext()){
Student st=(Student)itr.next();
System.out.println(st.rollno+" "+st.name+" "+st.age);
}

System.out.println("sorting by age...");

Collections.sort(al,new AgeComparator());
Iterator itr2=al.iterator();
while(itr2.hasNext()){
Student st=(Student)itr2.next();
System.out.println(st.rollno+" "+st.name+" "+st.age);
}

}

Output:Sorting by Name...
106 Ajay 27
105 Jai 21

```

```

101 Vijay 23
Sorting by age...
105 Jai 21
101 Vijay 23
106 Ajay 27

```

## Properties class in Java

The **properties** object contains key and value pair both as a string. It is the subclass of Hashtable.

It can be used to get property value based on the property key. The Properties class provides methods to get data from properties file and store data into properties file. Moreover, it can be used to get properties of system.

### Advantage of properties file

**Easy Maintenance:** If any information is changed from the properties file, you don't need to recompile the java class. It is mainly used to contain variable information i.e. to be changed.

## Methods of Properties class

The commonly used methods of Properties class are given below.

Method	Description
public void load(Reader r)	loads data from the Reader object.
public void load(InputStream is)	loads data from the InputStream object
public String getProperty(String key)	returns value based on the key.
public void setProperty(String key,String value)	sets the property in the properties object.
public void store(Writer w, String comment)	writers the properties in the writer object.
public void store.OutputStream os, String comment)	writes the properties in the OutputStream object.
storeToXML(OutputStream os, String comment)	writers the properties in the writer object for generating xml document.
public void storeToXML(Writer w, String comment, String encoding)	writers the properties in the writer object for generating xml document with specified encoding.

## Example of Properties class to get information from properties file

To get information from the properties file, create the properties file first.

### db.properties

```

user=system
password=oracle

```

Now, lets create the java class to read the data from the properties file.

### Test.java

```

import java.util.*;
import java.io.*;
public class Test {
public static void main(String[] args) throws Exception{
FileReader reader=new FileReader("db.properties");

Properties p=new Properties();

```

```

p.load(reader);

System.out.println(p.getProperty("user"));
System.out.println(p.getProperty("password"));
}

}

Output:system
oracle

```

Now if you change the value of the properties file, you don't need to compile the java class again. That means no maintenance problem.

---

## Example of Properties class to get all the system properties

By System.getProperties() method we can get all the properties of system. Let's create the class that gets information from the system properties.

### **Test.java**

```

import java.util.*;
import java.io.*;
public class Test {
public static void main(String[] args)throws Exception{

Properties p=System.getProperties();
Set set=p.entrySet();

Iterator itr=set.iterator();
while(itr.hasNext()){
Map.Entry entry=(Map.Entry)itr.next();
System.out.println(entry.getKey()+" = "+entry.getValue());
}

}

}

Output:
java.runtime.name = Java(TM) SE Runtime Environment
sun.boot.library.path = C:\Program Files\Java\jdk1.7.0_01\jre\bin
java.vm.version = 21.1-b02
java.vm.vendor = Oracle Corporation
java.vendor.url = http://java.oracle.com/
path.separator = ;
java.vm.name = Java HotSpot(TM) Client VM
file.encoding.pkg = sun.io
user.country = US
user.script =
sun.java.launcher = SUN_STANDARD
.....

```

---

## Example of Properties class to create properties file

Now lets write the code to create the properties file.

### **Test.java**

```

import java.util.*;
import java.io.*;
public class Test {
public static void main(String[] args)throws Exception{

Properties p=new Properties();

```

```

p.setProperty("name", "Sonoo Jaiswal");
p.setProperty("email", "sonoojaiswal@javatpoint.com");

p.store(new FileWriter("info.properties"), "Javatpoint Properties Example");

}
}

```

Let's see the generated properties file.

#### **info.properties**

```

#Javatpoint Properties Example
#Thu Oct 03 22:35:53 IST 2013
email=sonoojaiswal@javatpoint.com
name=Sonoo Jaiswal

```

## Difference between ArrayList and Vector

ArrayList and Vector both implements List interface and maintains insertion order.

But there are many differences between ArrayList and Vector classes that are given below.

ArrayList	Vector
1) ArrayList is <b>not synchronized</b> .	Vector is <b>synchronized</b> .
2) ArrayList <b>increments 50%</b> of current array size if number of element exceeds from its capacity.	Vector <b>increments 100%</b> means doubles the array size if total number of element exceeds than its capacity.
3) ArrayList is <b>not a legacy</b> class, it is introduced in JDK 1.2.	Vector is a <b>legacy</b> class.
4) ArrayList is <b>fast</b> because it is non-synchronized.	Vector is <b>slow</b> because it is synchronized i.e. in multithreading environment, it will hold the other threads in runnable or non-runnable state until current thread releases the lock of object.
5) ArrayList uses <b>Iterator</b> interface to traverse the elements.	Vector uses <b>Enumeration</b> interface to traverse the elements. But it can use Iterator also.

## Example of Java ArrayList

Let's see a simple example where we are using ArrayList to store and traverse the elements.

```

import java.util.*;
class TestArrayList21{
    public static void main(String args[]){

        List al=new ArrayList(); //creating arraylist
        al.add("Sonoo"); //adding object in arraylist
        al.add("Michael");
        al.add("James");
        al.add("Andy");
        //traversing elements using Iterator
        Iterator itr=al.iterator();
        while(itr.hasNext()){
            System.out.println(itr.next());
        }
    }
}

```

[Test it Now](#)

Output:

```
Sonoo  
Michael  
James  
Andy
```

## Example of Java Vector

Let's see a simple example of java Vector class that uses Enumeration interface.

```
import java.util.*;  
class TestVector1{  
    public static void main(String args[]){  
        Vector v=new Vector(); //creating vector  
        v.add("umesh"); //method of Collection  
        v.addElement("irfan"); //method of Vector  
        v.addElement("kumar");  
        //traversing elements using Enumeration  
        Enumeration e=v.elements();  
        while(e.hasMoreElements()){  
            System.out.println(e.nextElement());  
        }  
    }  
}
```

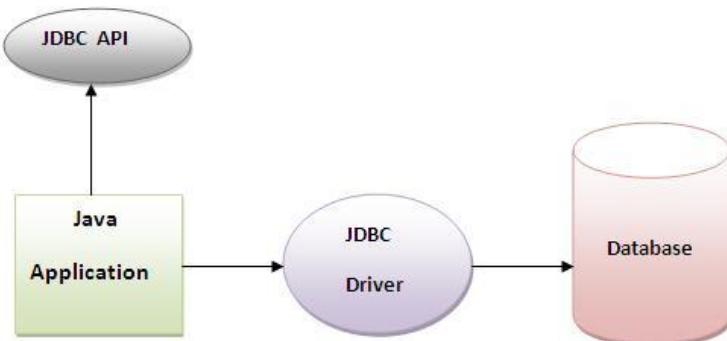
**Test it Now**

Output:

```
umesh  
irfan  
kumar
```

## Java JDBC Tutorial

Java JDBC is a java API to connect and execute query with the database. JDBC API uses jdbc drivers to connect with the database.



## Why use JDBC

Before JDBC, ODBC API was the database API to connect and execute query 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).

## What is API

API (Application programming interface) is a document that contains description of all the features of a product or software. It represents classes and interfaces that software programs can follow to communicate with each other. An API can be created for applications, libraries, operating systems, etc

---

## Topics in Java JDBC Tutorial

### 2) JDBC Drivers

In this JDBC tutorial, we will learn 4 types of JDBC drivers, their advantages and disadvantages.

---

### 3) 5 Steps to connect to the database

In this JDBC tutorial, we will see the 5 steps to connect to the database in java using JDBC.

---

### 4) Connectivity with Oracle using JDBC

In this JDBC tutorial, we will connect a simple java program with the oracle database.

---

### 5) Connectivity with MySQL using JDBC

In this JDBC tutorial, we will connect a simple java program with the mysql database.

---

### 6) Connectivity with Access without DSN

Let's connect java application with access database with and without DSN.

---

### 7) DriverManager class

In this JDBC tutorial, we will learn what does the DriverManager class and what are its methods.

---

### 8) Connection interface

In this JDBC tutorial, we will learn what is Connection interface and what are its methods.

---

### 9) Statement interface

In this JDBC tutorial, we will learn what is Statement interface and what are its methods.

---

### 10) ResultSet interface

In this JDBC tutorial, we will learn what is ResultSet interface and what are its methods. Moreover, we will learn how we can make the ResultSet scrollable.

---

### 11) PreparedStatement Interface

In this JDBC tutorial, we will learn what is benefit of PreparedStatement over Statement interface. We will see examples to insert, update or delete records using the PreparedStatement interface.

---

### 12) ResultSetMetaData interface

In this JDBC tutorial, we will learn how we can get the metadata of a table.

---

### 13) DatabaseMetaData interface

In this JDBC tutorial, we will learn how we can get the metadata of a database.

---

### 14) Storing image in Oracle

Let's learn how to store image in the oracle database using JDBC.

---

### 15) Retrieving image from Oracle

Let's see the simple example to retrieve image from the oracle database using JDBC.

---

### 16) Storing file in Oracle

Let's see the simple example to store file in the oracle database using JDBC.

---

## 17) Retrieving file from Oracle

Let's see the simple example to retrieve file from the oracle database using JDBC.

## 18) CallableStatement

Let's see the code to call stored procedures and functions using CallableStatement.

## 19) Transaction Management using JDBC

Let's see the simple example to use transaction management using JDBC.

## 20) Batch Statement using JDBC

Let's see the code to execute batch of queries.

## 21) JDBC RowSet

Let's see the working of new JDBC RowSet interface.

# JDBC Driver

JDBC Driver is a software component that enables java application to interact with the database. There are 4 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)

## 1) JDBC-ODBC bridge driver

The JDBC-ODBC bridge driver uses ODBC driver to connect to the database. The JDBC-ODBC bridge driver converts JDBC method calls into the ODBC function calls. This is now discouraged because of thin driver.

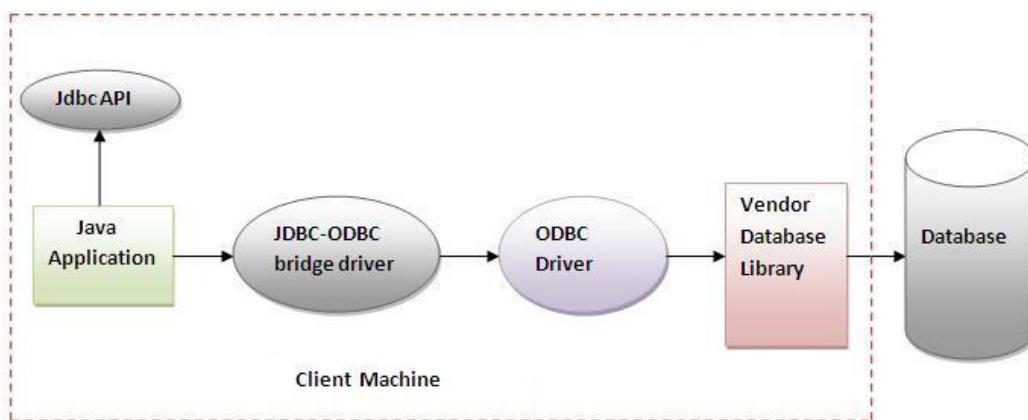


Figure- JDBC-ODBC Bridge Driver

### Advantages:

- easy to use.
- can be easily connected to any database.

### Disadvantages:

- Performance degraded because 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.

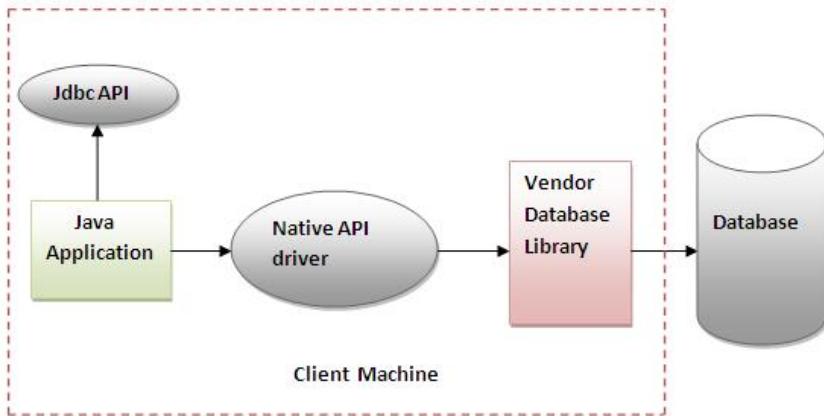


Figure- Native API Driver

### Advantage:

- performance upgraded than JDBC-ODBC bridge driver.

### Disadvantage:

- The Native driver needs to be installed on the each client machine.
- The Vendor client library needs to be installed on 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.

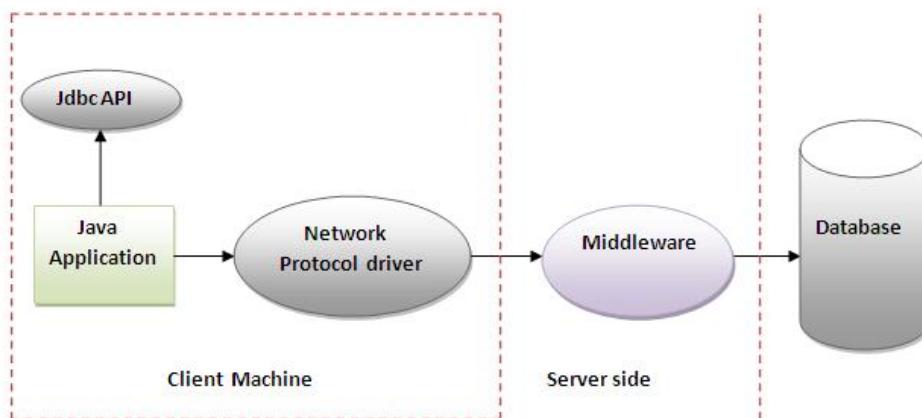


Figure- Network Protocol Driver

### Advantage:

- No client side library is required because of application server that can perform many tasks like auditing, load balancing, logging etc.

## Disadvantages:

- Network support is required on client machine.
- Requires database-specific coding to be done in the middle tier.
- Maintenance of 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 thin driver. It is fully written in Java language.

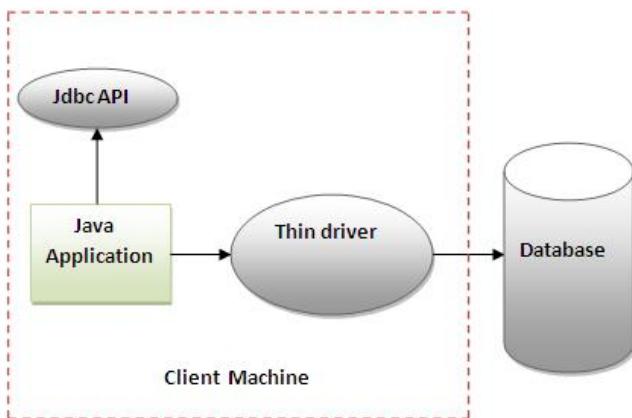


Figure- Thin Driver

## Advantage:

- Better performance than all other drivers.
- No software is required at client side or server side.

## Disadvantage:

- Drivers depends on the Database.

# 5 Steps to connect to the database in java

There are 5 steps to connect any java application with the database in java using JDBC. They are as follows:

- Register the driver class
- Creating connection
- Creating statement
- Executing queries
- Closing connection

## 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.

### Syntax of `forName()` method

```
public static void forName(String className) throws ClassNotFoundException
```

## Example to register the OracleDriver class

```
Class.forName("oracle.jdbc.driver.OracleDriver");
```

---

### 2) Create the connection object

The getConnection() method of DriverManager class is used to establish 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 name, String password)  
throws SQLException
```

### Example to establish connection with the Oracle database

```
Connection con=DriverManager.getConnection(  
"jdbc:oracle:thin:@localhost:1521:xe","system","password");
```

---

### 3) Create the Statement object

The createStatement() method of Connection interface is used to create statement. The object of statement is responsible to execute queries with the database.

#### Syntax of createStatement() method

```
public Statement createStatement() throws SQLException
```

### Example to create the statement object

```
Statement stmt=con.createStatement();
```

---

### 4) Execute the query

The executeQuery() method of Statement interface is used to execute queries to the database. This method returns the object of ResultSet that can be used to get all the records of a table.

#### Syntax of executeQuery() method

```
public ResultSet executeQuery(String sql) throws SQLException
```

### Example to execute query

```
ResultSet rs=stmt.executeQuery("select * from emp");  
  
while(rs.next()) {
```

```
System.out.println(rs.getInt(1)+" "+rs.getString(2));
}
```

---

## 5) Close the connection object

By closing connection object statement and ResultSet will be closed automatically. The close() method of Connection interface is used to close the connection.

### Syntax of close() method

```
public void close()throws SQLException
```

## Example to close connection

```
con.close();
```

## Example to connect to the Oracle database

For connecting java application with the oracle database, you need to follow 5 steps to perform database connectivity. In this example we are using Oracle10g as the database. So we need to know following informations for the oracle database:

1. **Driver class:** The driver class for the oracle database is **oracle.jdbc.driver.OracleDriver**.
2. **Connection URL:** The connection URL for the oracle10G database is **jdbc:oracle:thin:@localhost:1521:xe** where jdbc is the API, oracle is the database, thin is the driver, localhost is the server name on which oracle is running, we may also use IP address, 1521 is the port number and XE is the Oracle service name. You may get all these informations from the tnsnames.ora file.
3. **Username:** The default username for the oracle database is **system**.
4. **Password:** Password is given by the user at the time of installing the oracle database.

Let's first create a table in oracle database.

```
create table emp(id number(10),name varchar2(40),age number(3));
```

---

## Example to Connect Java Application with Oracle database

In this example, system is the username and oracle is the password of the Oracle database.

```
import java.sql.*;
class OracleCon{
public static void main(String args[]){
try{
//step1 load the driver class
Class.forName("oracle.jdbc.driver.OracleDriver");

//step2 create the connection object
Connection con=DriverManager.getConnection(

```

```

"jdbc:oracle:thin:@localhost:1521:xe","system","oracle");

//step3 create the statement object
Statement stmt=con.createStatement();

//step4 execute query
ResultSet rs=stmt.executeQuery("select * from emp");
while(rs.next())
System.out.println(rs.getInt(1)+" "+rs.getString(2)+" "+rs.getString(3));

//step5 close the connection object
con.close();

}catch(Exception e){ System.out.println(e);}

}
}

```

### [download this example](#)

The above example will fetch all the records of emp table.

---

To connect java application with the Oracle database ojdbc14.jar file is required to be loaded.

### [download the jar file ojdbc14.jar](#)

### Two ways to load the jar file:

1. paste the ojdbc14.jar file in jre/lib/ext folder
2. set classpath

#### 1) paste the ojdbc14.jar file in JRE/lib/ext folder:

Firstly, search the ojdbc14.jar file then go to JRE/lib/ext folder and paste the jar file here.

#### 2) set classpath:

There are two ways to set the classpath:

- temporary
- permanent

#### How to set the temporary classpath:

Firstly, search the ojdbc14.jar file then open command prompt and write:

```
C:>set classpath=c:\folder\ojdbc14.jar;.
```

#### How to set the permanent classpath:

Go to environment variable then click on new tab. In variable name write **classpath** and in variable value paste the path to ojdbc14.jar by appending ojdbc14.jar;.; as C:\oraclexe\app\oracle\product\10.2.0\server\jdbc\lib\ojdbc14.jar;.;

To see the slides of setting permanent path [click here](#)

## Example to connect to the mysql database

For connecting java application with the mysql database, you need to follow 5 steps to perform database connectivity.

In this example we are using MySql as the database. So we need to know following informations for the mysql database:

1. **Driver class:** The driver class for the mysql database is **com.mysql.jdbc.Driver**.
2. **Connection URL:** The connection URL for the mysql database is **jdbc:mysql://localhost:3306/sonoo** 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 sonoo is the database name. We may use any database, in such case, you need to replace the sonoo with your database name.
3. **Username:** The default username for the mysql database is **root**.
4. **Password:** Password is given by the user at the time of installing the mysql database. In this example, we are going to use root as the password.

Let's first create a table in the mysql database, but before creating table, we need to create database first.

```
create database sonoo;
use sonoo;
create table emp(id int(10),name varchar(40),age int(3));
```

---

## Example to Connect Java Application with mysql database

In this example, sonoo is the database name, root is the username and password.

```
import java.sql.*;
class MysqlCon{
public static void main(String args[]){
try{
Class.forName("com.mysql.jdbc.Driver");

Connection con=DriverManager.getConnection(
"jdbc:mysql://localhost:3306/sonoo","root","root");

//here sonoo is database name, root is username and password

Statement stmt=con.createStatement();

ResultSet rs=stmt.executeQuery("select * from emp");

while(rs.next())
System.out.println(rs.getInt(1)+" "+rs.getString(2)+" "+rs.getString(3));

con.close();

}catch(Exception e){ System.out.println(e);}

}
}
```

[download this example](#)

The above example will fetch all the records of emp table.

---

To connect java application with the mysql database mysqlconnector.jar file is required to be loaded.

[download the jar file mysql-connector.jar](#)

**Two ways to load the jar file:**

1. paste the mysqlconnector.jar file in jre/lib/ext folder
2. set classpath

## 1) paste the mysqlconnector.jar file in JRE/lib/ext folder:

Download the mysqlconnector.jar file. Go to jre/lib/ext folder and paste the jar file here.

## 2) set classpath:

There are two ways to set the classpath:

- temporary
- permanent

### How to set the temporary classpath

open command prompt and write:

```
C:>set classpath=c:\folder\mysql-connector-java-5.0.8-bin.jar;.;
```

### How to set the permanent classpath

Go to environment variable then click on new tab. In variable name write **classpath** and in variable value paste the path to the mysqlconnector.jar file by appending mysqlconnector.jar;.; as C:\folder\mysql-connector-java-5.0.8-bin.jar;.;

## Connectivity with Access without DSN

There are two ways to connect java application with the access database.

1. Without DSN (Data Source Name)
2. With DSN

Java is mostly used with Oracle, mysql, or DB2 database. So you can learn this topic only for knowledge.

### Example to Connect Java Application with access without DSN

In this example, we are going to connect the java program with the access database. In such case, we have created the login table in the access database. There is only one column in the table named name. Let's get all the name of the login table.

```
import java.sql.*;
class Test{
public static void main(String ar[]){
try{
String database="student.mdb";//Here database exists in the current directory

String url="jdbc:odbc:Driver={Microsoft Access Driver (*.mdb)};
           DBQ=" + database + ";DriverID=22;READONLY=true";

Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
Connection c=DriverManager.getConnection(url);
Statement st=c.createStatement();
ResultSet rs=st.executeQuery("select * from login");
```

```

        while(rs.next()){
            System.out.println(rs.getString(1));
        }

    }catch(Exception ee){System.out.println(ee);}

}

```

[download this example](#)

## Example to Connect Java Application with access with DSN

Connectivity with type1 driver is not considered good. To connect java application with type1 driver, create DSN first, here we are assuming your dsn name is mydsn.

```

import java.sql.*;
class Test{
public static void main(String ar[]){
try{
    String url="jdbc:odbc:mydsn";
    Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
    Connection c=DriverManager.getConnection(url);
    Statement st=c.createStatement();
    ResultSet rs=st.executeQuery("select * from login");

    while(rs.next()){
        System.out.println(rs.getString(1));
    }

}catch(Exception ee){System.out.println(ee);}

}

```

## DriverManager class:

The DriverManager class acts as an interface between user and drivers. It keeps track of the drivers that are available and handles establishing a connection between a database and the appropriate driver. The DriverManager class maintains a list of Driver classes that have registered themselves by calling the method DriverManager.registerDriver().

### Commonly used methods of DriverManager class:

- |  |  |
|--|--|
| 1) public static void registerDriver(Driver driver):                                   | is used to register the given driver with DriverManager.                                   |
| 2) public static void deregisterDriver(Driver driver):                                 | is used to deregister the given driver (drop the driver from the list) with DriverManager. |
| 3) public static Connection getConnection(String url):                                 | is used to establish the connection with the specified url.                                |
| 4) public static Connection getConnection(String url,String userName,String password): | is used to establish the connection with the specified url, username and password.         |

## Connection interface:

A Connection is the session between java application and database. The Connection interface is a factory of Statement, PreparedStatement, and DatabaseMetaData i.e. object of Connection can be used to get the object of Statement and DatabaseMetaData. The Connection interface provide many methods for transaction management like commit(), rollback() etc.

**By default, connection commits the changes after executing queries.**

### Commonly used methods of Connection interface:

- 1) public Statement createStatement():** creates a statement object that can be used to execute SQL queries.
- 2) public Statement createStatement(int resultSetType,int resultSetConcurrency):** Creates a Statement object that will generate ResultSet objects with the given type and concurrency.
- 3) public void setAutoCommit(boolean status):** is used to set the commit status. By default it is true.
- 4) public void commit():** saves the changes made since the previous commit/rollback permanent.
- 5) public void rollback():** Drops all changes made since the previous commit/rollback.
- 6) public void close():** closes the connection and Releases a JDBC resources immediately.

## Statement interface

The **Statement interface** provides methods to execute queries with the database. The statement interface is a factory of ResultSet i.e. it provides factory method to get the object of ResultSet.

### Commonly used methods of Statement interface:

The important methods of Statement interface are as follows:

- 1) public ResultSet executeQuery(String sql):** is used to execute SELECT query. It returns the object of ResultSet.
- 2) public int executeUpdate(String sql):** is used to execute specified query, it may be create, drop, insert, update, delete etc.
- 3) public boolean execute(String sql):** is used to execute queries that may return multiple results.
- 4) public int[] executeBatch():** is used to execute batch of commands.

## Example of Statement interface

Let's see the simple example of Statement interface to insert, update and delete the record.

```
import java.sql.*;
class FetchRecord{
public static void main(String args[])throws Exception{

Class.forName("oracle.jdbc.driver.OracleDriver");
Connection con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe","system","oracle");
Statement stmt=con.createStatement();

//stmt.executeUpdate("insert into emp765 values(33,'Irfan',50000)");
//int result=stmt.executeUpdate("update emp765 set name='Vimal',salary=10000 where id=33");
int result=stmt.executeUpdate("delete from emp765 where id=33");

System.out.println(result+" records affected");
con.close();
}}
```

# ResultSet interface

The object of ResultSet maintains a cursor pointing to a particular row of data. Initially, cursor points to before the first row.

 **By default, ResultSet object can be moved forward only and it is not updatable.**

But we can make this object to move forward and backward direction by passing either TYPE\_SCROLL\_INSENSITIVE or TYPE\_SCROLL\_SENSITIVE in createStatement(int,int) method as well as we can make this object as updatable by:

```
Statement stmt = con.createStatement	ResultSet.TYPE_SCROLL_INSENSITIVE,
	ResultSet.CONCUR_UPDATABLE);
```

## Commonly used methods of ResultSet interface

<b>1) public boolean next():</b>	is used to move the cursor to the one row next from the current position.
<b>2) public boolean previous():</b>	is used to move the cursor to the one row previous from the current position.
<b>3) public boolean first():</b>	is used to move the cursor to the first row in result set object.
<b>4) public boolean last():</b>	is used to move the cursor to the last row in result set object.
<b>5) public boolean absolute(int row):</b>	is used to move the cursor to the specified row number in the ResultSet object.
<b>6) public boolean relative(int row):</b>	is used to move the cursor to the relative row number in the ResultSet object, it may be positive or negative.
<b>7) public int getInt(int columnIndex):</b>	is used to return the data of specified column index of the current row as int.
<b>8) public int getInt(String columnName):</b>	is used to return the data of specified column name of the current row as int.
<b>9) public String getString(int columnIndex):</b>	is used to return the data of specified column index of the current row as String.
<b>10) public String getString(String columnName):</b>	is used to return the data of specified column name of the current row as String.

## Example of Scrollable ResultSet

Let's see the simple example of ResultSet interface to retrieve the data of 3rd row.

```
import java.sql.*;
class FetchRecord{
public static void main(String args[])throws Exception{

Class.forName("oracle.jdbc.driver.OracleDriver");
Connection con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe","system","oracle");
Statement stmt=con.createStatement	ResultSet.TYPE_SCROLL_SENSITIVE,ResultSet.CONCUR_UPDATABLE);
ResultSet rs=stmt.executeQuery("select * from emp765");

//getting the record of 3rd row
rs.absolute(3);
System.out.println(rs.getString(1)+" "+rs.getString(2)+" "+rs.getString(3));

con.close();
}}
```

# PreparedStatement interface

The PreparedStatement interface is a subinterface of Statement. It is used to execute parameterized query.

Let's see the example of parameterized query:

```
String sql="insert into emp values(?, ?, ?);
```

As you can see, we are passing parameter (?) for the values. Its value will be set by calling the setter methods of PreparedStatement.

## Why use PreparedStatement?

**Improves performance:** The performance of the application will be faster if you use PreparedStatement interface because query is compiled only once.

## How to get the instance of PreparedStatement?

The prepareStatement() method of Connection interface is used to return the object of PreparedStatement. Syntax:

```
public PreparedStatement prepareStatement(String query) throws SQLException{}
```

## Methods of PreparedStatement interface

The important methods of PreparedStatement interface are given below:

Method	Description
public void setInt(int paramInt, int value)	sets the integer value to the given parameter index.
public void setString(int paramInt, String value)	sets the String value to the given parameter index.
public void setFloat(int paramInt, float value)	sets the float value to the given parameter index.
public void setDouble(int paramInt, double value)	sets the double value to the given parameter index.
public int executeUpdate()	executes the query. It is used for create, drop, insert, update, delete etc.
public ResultSet executeQuery()	executes the select query. It returns an instance of ResultSet.

## Example of PreparedStatement interface that inserts the record

First of all create table as given below:

```
create table emp(id number(10),name varchar2(50));
```

Now insert records in this table by the code given below:

```
import java.sql.*;
class InsertPrepared{
public static void main(String args[]){
try{
Class.forName("oracle.jdbc.driver.OracleDriver");

Connection con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe","system","oracle");

PreparedStatement stmt=con.prepareStatement("insert into Emp values(?,?)");
stmt.setInt(1,101);//1 specifies the first parameter in the query
stmt.setString(2,"Ratan");
```

```
int i=stmt.executeUpdate();
System.out.println(i+" records inserted");

con.close();

}catch(Exception e){ System.out.println(e);}

}
}
```

[download this example](#)

---

## Example of PreparedStatement interface that updates the record

```
PreparedStatement stmt=con.prepareStatement("update emp set name=? where id=?");
stmt.setString(1,"Sonoo");//1 specifies the first parameter in the query i.e. name
stmt.setInt(2,101);

int i=stmt.executeUpdate();
System.out.println(i+" records updated");
```

[download this example](#)

---

## Example of PreparedStatement interface that deletes the record

```
PreparedStatement stmt=con.prepareStatement("delete from emp where id=?");
stmt.setInt(1,101);

int i=stmt.executeUpdate();
System.out.println(i+" records deleted");
```

[download this example](#)

---

## Example of PreparedStatement interface that retrieve the records of a table

```
PreparedStatement stmt=con.prepareStatement("select * from emp");
ResultSet rs=stmt.executeQuery();
while(rs.next()){
System.out.println(rs.getInt(1)+" "+rs.getString(2));
}
```

[download this example](#)

---

## Example of PreparedStatement to insert records until user press n

```
import java.sql.*;
import java.io.*;
class RS{
public static void main(String args[])throws Exception{
Class.forName("oracle.jdbc.driver.OracleDriver");
Connection con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe","system","oracle");

PreparedStatement ps=con.prepareStatement("insert into emp130 values(?, ?, ?)");

BufferedReader br=new BufferedReader(new InputStreamReader(System.in));

do{
System.out.println("enter id:");
int id=Integer.parseInt(br.readLine());
System.out.println("enter name:");
String name=br.readLine();
```

```

System.out.println("enter salary:");
float salary=Float.parseFloat(br.readLine());

ps.setInt(1,id);
ps.setString(2,name);
ps.setFloat(3,salary);
int i=ps.executeUpdate();
System.out.println(i+" records affected");

System.out.println("Do you want to continue: y/n");
String s=br.readLine();
if(s.startsWith("n")){
break;
}
}while(true);

con.close();
}}

```

## ResultSetMetaData Interface

The metadata means data about data i.e. we can get further information from the data.

If you have to get metadata of a table like total number of column, column name, column type etc. , ResultSetMetaData interface is useful because it provides methods to get metadata from the ResultSet object.

### Commonly used methods of ResultSetMetaData interface

Method	Description
public int getColumnCount()throws SQLException	it returns the total number of columns in the ResultSet object.
public String getColumnName(int index) throws SQLException	it returns the column name of the specified column index.
public String getColumnTypeName(int index) throws SQLException	it returns the column type name for the specified index.
public String getTableName(int index) throws SQLException	it returns the table name for the specified column index.

### How to get the object of ResultSetMetaData:

The getMetaData() method of ResultSet interface returns the object of ResultSetMetaData. Syntax:

```
public ResultSetMetaData getMetaData() throws SQLException
```

### Example of ResultSetMetaData interface :

```

import java.sql.*;
class Rsmd{
public static void main(String args[]){
try{
Class.forName("oracle.jdbc.driver.OracleDriver");

```

```

Connection con=DriverManager.getConnection(
"jdbc:oracle:thin:@localhost:1521:xe","system","oracle");

PreparedStatement ps=con.prepareStatement("select * from emp");
ResultSet rs=ps.executeQuery();

ResultSetMetaData rsmd=rs.getMetaData();

System.out.println("Total columns: "+rsmd.getColumnCount());
System.out.println("Column Name of 1st column: "+rsmd.getColumnName(1));
System.out.println("Column Type Name of 1st column: "+rsmd.getColumnTypeName(1));

con.close();

}catch(Exception e){ System.out.println(e);}

}
}

Output:Total columns: 2
        Column Name of 1st column: ID
        Column Type Name of 1st column: NUMBER

```

## DatabaseMetaData interface:

DatabaseMetaData interface provides methods to get meta data of a database such as database product name, database product version, driver name, name of total number of tables, name of total number of views etc.

### Commonly used methods of DatabaseMetaData interface

- **public String getDriverName()throws SQLException:** it returns the name of the JDBC driver.
- **public String getDriverVersion()throws SQLException:** it returns the version number of the JDBC driver.
- **public String getUserName()throws SQLException:** it returns the username of the database.
- **public String getDatabaseProductName()throws SQLException:** it returns the product name of the database.
- **public String getDatabaseProductVersion()throws SQLException:** it returns the product version of the database.
- **public ResultSet getTables(String catalog, String schemaPattern, String tableNamePattern, String[] types)throws SQLException:** it returns the description of the tables of the specified catalog. The table type can be TABLE, VIEW, ALIAS, SYSTEM TABLE, SYNONYM etc.

### How to get the object of DatabaseMetaData:

The getMetaData() method of Connection interface returns the object of DatabaseMetaData. Syntax:

```
public DatabaseMetaData getMetaData()throws SQLException
```

### Simple Example of DatabaseMetaData interface :

```

import java.sql.*;
class Dbmd{
public static void main(String args[]){
try{
Class.forName("oracle.jdbc.driver.OracleDriver");

```

```

Connection con=DriverManager.getConnection(
"jdbc:oracle:thin:@localhost:1521:xe","system","oracle");

DatabaseMetaData dbmd=con.getMetaData();

System.out.println("Driver Name: "+dbmd.getDriverName());
System.out.println("Driver Version: "+dbmd.getDriverVersion());
System.out.println("UserName: "+dbmd.getUserName());
System.out.println("Database Product Name: "+dbmd.getDatabaseProductName());
System.out.println("Database Product Version: "+dbmd.getDatabaseProductVersion());

con.close();

}catch(Exception e){ System.out.println(e);}

}
}

Output:Driver Name: Oracle JDBC Driver
        Driver Version: 10.2.0.1.0XE
        Database Product Name: Oracle
        Database Product Version: Oracle Database 10g Express Edition
                                Release 10.2.0.1.0 -Production

```

[download this example](#)

---

### Example of DatabaseMetaData interface that prints total number of tables :

```

import java.sql.*;
class Dbmd2{
public static void main(String args[]){
try{
Class.forName("oracle.jdbc.driver.OracleDriver");

Connection con=DriverManager.getConnection(
"jdbc:oracle:thin:@localhost:1521:xe","system","oracle");

DatabaseMetaData dbmd=con.getMetaData();
String table[]={ "TABLE"};
ResultSet rs=dbmd.getTables(null,null,null,table);

while(rs.next()){
System.out.println(rs.getString(3));
}

con.close();

}catch(Exception e){ System.out.println(e);}

}
}


```

[download this example](#)

---

### Example of DatabaseMetaData interface that prints total number of views :

```

import java.sql.*;
class Dbmd3{
public static void main(String args[]){

```

```

try{
Class.forName("oracle.jdbc.driver.OracleDriver");

Connection con=DriverManager.getConnection(
"jdbc:oracle:thin:@localhost:1521:xe","system","oracle");

DatabaseMetaData dbmd=con.getMetaData();
String table[]={"VIEW"};
ResultSet rs=dbmd.getTables(null,null,null,table);

while(rs.next()){
System.out.println(rs.getString(3));
}

con.close();

}catch(Exception e){ System.out.println(e);}

}
}

```

[download this example](#)

## Example to store image in Oracle database

You can store images in the database in java by the help of **PreparedStatement** interface.

The **setBinaryStream()** method of PreparedStatement is used to set Binary information into the parameterIndex.

### Signature of setBinaryStream method

The syntax of setBinaryStream() method is given below:

```

1) public void setBinaryStream(int paramIndex,InputStream stream)
throws SQLException
2) public void setBinaryStream(int paramIndex,InputStream stream,long length)
throws SQLException

```

For storing image into the database, BLOB (Binary Large Object) datatype is used in the table. For example:

```

CREATE TABLE "IMGTABLE"
( "NAME" VARCHAR2(4000),
"PHOTO" BLOB
)
/

```

Let's write the jdbc code to store the image in the database. Here we are using d:\\d.jpg for the location of image. You can change it according to the image location.

```

import java.sql.*;
import java.io.*;
public class InsertImage {
public static void main(String[] args) {
try{
Class.forName("oracle.jdbc.driver.OracleDriver");
Connection con=DriverManager.getConnection(
"jdbc:oracle:thin:@localhost:1521:xe","system","oracle");

PreparedStatement ps=con.prepareStatement("insert into imgtable values(?,?)");
ps.setString(1,"sonoo");

PreparedStat

```

```

FileInputStream fin=new FileInputStream("d:\\g.jpg");
ps.setBinaryStream(2,fin,fin.available());
int i=ps.executeUpdate();
System.out.println(i+" records affected");

con.close();
}catch (Exception e) {e.printStackTrace();}
}
}

```

If you see the table, record is stored in the database but image will not be shown. To do so, you need to retrieve the image from the database which we are covering in the next page.

[download this example](#)

## Example to retrieve image from Oracle database

By the help of **PreparedStatement** we can retrieve and store the image in the database.

The **getBlob()** method of PreparedStatement is used to get Binary information, it returns the instance of Blob. After calling the **getBytes()** method on the blob object, we can get the array of binary information that can be written into the image file.

### Signature of getBlob() method of PreparedStatement

```
public Blob getBlob()throws SQLException
```

### Signature of getBytes() method of Blob interface

```
public byte[] getBytes(long pos, int length)throws SQLException
```

We are assuming that image is stored in the imgtable.

```

CREATE TABLE "IMGTABLE"
(
  "NAME" VARCHAR2(4000),
  "PHOTO" BLOB
)
/

```

Now let's write the code to retrieve the image from the database and write it into the directory so that it can be displayed.

In AWT, it can be displayed by the Toolkit class. In servlet, jsp, or html it can be displayed by the img tag.

```

import java.sql.*;
import java.io.*;
public class RetrieveImage {
public static void main(String[] args) {
try{
Class.forName("oracle.jdbc.driver.OracleDriver");
Connection con=DriverManager.getConnection(
"jdbc:oracle:thin:@localhost:1521:xe","system","oracle");

PreparedStatement ps=con.prepareStatement("select * from imgtable");
ResultSet rs=ps.executeQuery();
if(rs.next()){//now on 1st row

Blob b=rs.getBlob(2);//2 means 2nd column data
byte barr[]={};b.getBytes(1,(int)b.length());//1 means first image
}
}
}

```

```

FileOutputStream fout=new FileOutputStream("d:\\sonoo.jpg");
fout.write(barr);

fout.close();
}//end of if
System.out.println("ok");

con.close();
}catch (Exception e) {e.printStackTrace(); }
}
}

```

Now if you see the d drive, sonoo.jpg image is created.

[download this example](#)

## Example to store file in Oracle database:

The setCharacterStream() method of PreparedStatement is used to set character information into the parameterIndex.

### Syntax:

- 1) public void setBinaryStream(int paramInt,InputStream stream)throws SQLException
- 2) public void setBinaryStream(int paramInt,InputStream stream,long length)throws SQLException

For storing file into the database, CLOB (Character Large Object) datatype is used in the table. For example:

```

CREATE TABLE "FILETABLE"
(
  "ID" NUMBER,
  "NAME" CLOB
)
/

```

```

import java.io.*;
import java.sql.*;

public class StoreFile {
public static void main(String[] args) {
try{
Class.forName("oracle.jdbc.driver.OracleDriver");
Connection con=DriverManager.getConnection(
"jdbc:oracle:thin:@localhost:1521:xe","system","oracle");

PreparedStatement ps=con.prepareStatement(
"insert into filetable values(?,?)");

File f=new File("d:\\myfile.txt");
FileReader fr=new FileReader(f);

ps.setInt(1,101);
ps.setCharacterStream(2,fr,(int)f.length());

```

```

int i=ps.executeUpdate();
System.out.println(i+" records affected");

con.close();

}catch (Exception e) {e.printStackTrace();}
}
}

```

[download this example](#)

## Example to retrieve file from Oracle database:

The getClob() method of PreparedStatement is used to get file information from the database.

### Syntax of getClob method

```
public Clob getClob(int columnIndex){}
```

Let's see the table structure of this example to retrieve the file.

```

CREATE TABLE "FILETABLE"
( "ID" NUMBER,
"NAME" CLOB
)
/

```

The example to retrieve the file from the Oracle database is given below.

```

import java.io.*;
import java.sql.*;

public class Retrievefile {
public static void main(String[] args) {
try{
Class.forName("oracle.jdbc.driver.OracleDriver");
Connection con=DriverManager.getConnection(
"jdbc:oracle:thin:@localhost:1521:xe","system","oracle");

PreparedStatement ps=con.prepareStatement("select * from filetable");
ResultSet rs=ps.executeQuery();
rs.next(); //now on 1st row

Clob c=rs.getBlob(2);
Reader r=c.getCharacterStream();

FileWriter fw=new FileWriter("d:\\retrivefile.txt");

int i;

```

```

while((i=r.read())!=-1)
fw.write((char)i);

fw.close();
con.close();

System.out.println("success");
}catch (Exception e) {e.printStackTrace(); }
}
}

```

[download this example](#)

## CallableStatement Interface

To call the **stored procedures and functions**, CallableStatement interface is used.

We can have business logic on the database by the use of stored procedures and functions that will make the performance better because these are precompiled.

Suppose you need to get the age of the employee based on the date of birth, you may create a function that receives date as the input and returns age of the employee as the output.

### What is the difference between stored procedures and functions.

The differences between stored procedures and functions are given below:

Stored Procedure	Function
is used to perform business logic.	is used to perform calculation.
must not have the return type.	must have the return type.
may return 0 or more values.	may return only one values.
We can call functions from the procedure.	Procedure cannot be called from function.
Procedure supports input and output parameters.	Function supports only input parameter.
Exception handling using try/catch block can be used in stored procedures.	Exception handling using try/catch can't be used in user defined functions.

### How to get the instance of CallableStatement?

The prepareCall() method of Connection interface returns the instance of CallableStatement. Syntax is given below:

```
public CallableStatement prepareCall("{ call procedurename(?,?...?) }");
```

The example to get the instance of CallableStatement is given below:

```
CallableStatement stmt=con.prepareCall("{call myprocedure(?,?)}");
```

It calls the procedure myprocedure that receives 2 arguments.

## Full example to call the stored procedure using JDBC

To call the stored procedure, you need to create it in the database. Here, we are assuming that stored procedure looks like this.

```

create or replace procedure "INSERTR"
(id IN NUMBER,
name IN VARCHAR2)
is
begin
insert into user420 values(id,name);
end;
/

```

The table structure is given below:

```
create table user420(id number(10), name varchar2(200));
```

In this example, we are going to call the stored procedure INSERTR that receives id and name as the parameter and inserts it into the table user420. Note that you need to create the user420 table as well to run this application.

```

import java.sql.*;
public class Proc {
public static void main(String[] args) throws Exception{

Class.forName("oracle.jdbc.driver.OracleDriver");
Connection con=DriverManager.getConnection(
"jdbc:oracle:thin:@localhost:1521:xe","system","oracle");

CallableStatement stmt=con.prepareCall("{call insertR(?,?)}");
stmt.setInt(1,1011);
stmt.setString(2,"Amit");
stmt.execute();

System.out.println("success");
}
}

```

Now check the table in the database, value is inserted in the user420 table.

---

## Example to call the function using JDBC

In this example, we are calling the sum4 function that receives two input and returns the sum of the given number. Here, we have used the **registerOutParameter** method of CallableStatement interface, that registers the output parameter with its corresponding type. It provides information to the CallableStatement about the type of result being displayed.

The **Types** class defines many constants such as INTEGER, VARCHAR, FLOAT, DOUBLE, BLOB, CLOB etc.

Let's create the simple function in the database first.

```

create or replace function sum4
(n1 in number,n2 in number)
return number
is
temp number(8);
begin
temp :=n1+n2;
return temp;
end;
/

```

Now, let's write the simple program to call the function.

```

import java.sql.*;

public class FuncSum {
public static void main(String[] args) throws Exception{

```

```

Class.forName("oracle.jdbc.driver.OracleDriver");
Connection con=DriverManager.getConnection(
"jdbc:oracle:thin:@localhost:1521:xe","system","oracle");

CallableStatement stmt=con.prepareCall("{?= call sum4(?,?)}");
stmt.setInt(2,10);
stmt.setInt(3,43);
stmt.registerOutParameter(1,Types.INTEGER);
stmt.execute();

System.out.println(stmt.getInt(1));

}

}

Output: 53

```

## Transaction Management in JDBC

Transaction represents **a single unit of work**.

The ACID properties describes the transaction management well. ACID stands for Atomicity, Consistency, isolation and durability.

**Atomicity** means either all successful or none.

**Consistency** ensures bringing the database from one consistent state to another consistent state.

**Isolation** ensures that transaction is isolated from other transaction.

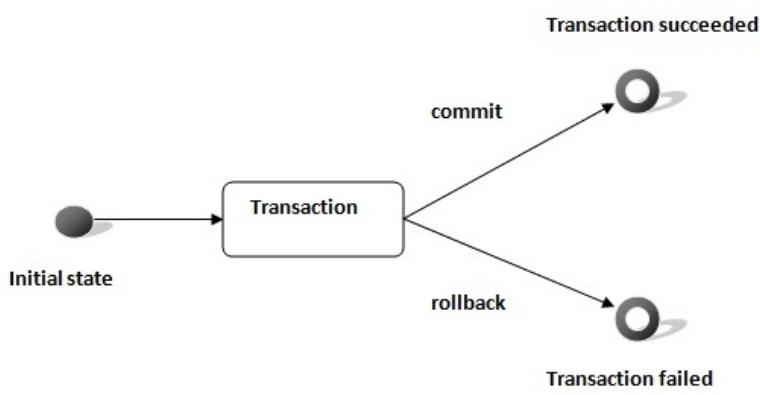
**Durability** means once a transaction has been committed, it will remain so, even in the event of errors, power loss etc.

---

### Advantage of Transaction Management

**fast performance** It makes the performance fast because database is hit at the time of commit.

---



In JDBC, **Connection interface** provides methods to manage transaction.

Method	Description
void setAutoCommit(boolean status)	It is true by default means each transaction is committed by default.
void commit()	commits the transaction.
void rollback()	cancels the transaction.

### Simple example of transaction management in jdbc using Statement

Let's see the simple example of transaction management using Statement.

```
import java.sql.*;
class FetchRecords{
public static void main(String args[])throws Exception{
Class.forName("oracle.jdbc.driver.OracleDriver");
Connection con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe","system","oracle");
con.setAutoCommit(false);

Statement stmt=con.createStatement();
stmt.executeUpdate("insert into user420 values(190,'abhi',40000)");
stmt.executeUpdate("insert into user420 values(191,'umesh',50000)");

con.commit();
con.close();
}}
```

If you see the table emp400, you will see that 2 records has been added.

## Example of transaction management in jdbc using PreparedStatement

Let's see the simple example of transaction management using PreparedStatement.

```
import java.sql.*;
import java.io.*;
class TM{
public static void main(String args[]){
try{

Class.forName("oracle.jdbc.driver.OracleDriver");
Connection con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe","system","oracle");
con.setAutoCommit(false);

PreparedStatement ps=con.prepareStatement("insert into user420 values(?, ?, ?)");

BufferedReader br=new BufferedReader(new InputStreamReader(System.in));
while(true){

System.out.println("enter id");
String s1=br.readLine();
int id=Integer.parseInt(s1);

System.out.println("enter name");
String name=br.readLine();

System.out.println("enter salary");
String s3=br.readLine();
int salary=Integer.parseInt(s3);

ps.setInt(1,id);
ps.setString(2,name);
ps.setInt(3,salary);
ps.executeUpdate();

System.out.println("commit/rollback");
String answer=br.readLine();
if(answer.equals("commit")){
con.commit();
}
if(answer.equals("rollback")){
con.rollback();
}
}
}
}
```

```

        }

System.out.println("Want to add more records y/n");
String ans=br.readLine();
if(ans.equals("n")){
break;
}

}

con.commit();
System.out.println("record successfully saved");

con.close();//before closing connection commit() is called
}catch(Exception e){System.out.println(e);}

}}

```

It will ask to add more records until you press n. If you press n, transaction is committed.

---

## Batch Processing in JDBC

Instead of executing a single query, we can execute a batch (group) of queries. It makes the performance fast.

The `java.sql.Statement` and `java.sql.PreparedStatement` interfaces provide methods for batch processing.

### Advantage of Batch Processing

Fast Performance

---

### Methods of Statement interface

The required methods for batch processing are given below:

Method	Description
<code>void addBatch(String query)</code>	It adds query into batch.
<code>int[] executeBatch()</code>	It executes the batch of queries.

---

## Example of batch processing in jdbc

Let's see the simple example of batch processing in jdbc. It follows following steps:

- Load the driver class
- Create Connection
- Create Statement
- Add query in the batch
- Execute Batch
- Close Connection

```

import java.sql.*;
class FetchRecords{
public static void main(String args[])throws Exception{
Class.forName("oracle.jdbc.driver.OracleDriver");
Connection con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe","system","oracle");
con.setAutoCommit(false);

```

```

Statement stmt=con.createStatement();
stmt.addBatch("insert into user420 values(190,'abhi',40000)");
stmt.addBatch("insert into user420 values(191,'umesh',50000)");

stmt.executeBatch();//executing the batch

con.commit();
con.close();
}}

```

If you see the table user420, two records has been added.

## Example of batch processing using PreparedStatement

```

import java.sql.*;
import java.io.*;
class BP{
public static void main(String args[]){
try{

Class.forName("oracle.jdbc.driver.OracleDriver");
Connection con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe","system","oracle");

PreparedStatement ps=con.prepareStatement("insert into user420 values(?, ?, ?)");

BufferedReader br=new BufferedReader(new InputStreamReader(System.in));
while(true){

System.out.println("enter id");
String s1=br.readLine();
int id=Integer.parseInt(s1);

System.out.println("enter name");
String name=br.readLine();

System.out.println("enter salary");
String s3=br.readLine();
int salary=Integer.parseInt(s3);

ps.setInt(1,id);
ps.setString(2,name);
ps.setInt(3,salary);

ps.addBatch();
System.out.println("Want to add more records y/n");
String ans=br.readLine();
if(ans.equals("n")){
break;
}

}

ps.executeBatch();

System.out.println("record successfully saved");

con.close();
}catch(Exception e){System.out.println(e);}

}}

```

It will add the queries into the batch until user press n. Finally it executes the batch. Thus all the added queries will be fired.

# JDBC RowSet

The instance of **RowSet** is the java bean component because it has properties and java bean notification mechanism. It is introduced since JDK 5.

It is the wrapper of ResultSet. It holds tabular data like ResultSet but it is easy and flexible to use.

The implementation classes of RowSet interface are as follows:

- JdbcRowSet
- CachedRowSet
- WebRowSet
- JoinRowSet
- FilteredRowSet

Let's see how to create and execute RowSet.

```
JdbcRowSet rowSet = RowSetProvider.newFactory().createJdbcRowSet();
rowSet.setUrl("jdbc:oracle:thin:@localhost:1521:xe");
rowSet.setUsername("system");
rowSet.setPassword("oracle");

rowSet.setCommand("select * from emp400");
rowSet.execute();
```



*It is the new way to get the instance of JdbcRowSet since JDK 7.*

## Advantage of RowSet

The advantages of using RowSet are given below:

1. It is easy and flexible to use
2. It is Scrollable and Updatable by default

## Simple example of JdbcRowSet

Let's see the simple example of JdbcRowSet without event handling code.

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.Statement;
import javax.sql.RowSetEvent;
import javax.sql.RowSetListener;
import javax.sql.rowset.JdbcRowSet;
import javax.sql.rowset.RowSetProvider;

public class RowSetExample {
    public static void main(String[] args) throws Exception {
        Class.forName("oracle.jdbc.driver.OracleDriver");

        //Creating and Executing RowSet
        JdbcRowSet rowSet = RowSetProvider.newFactory().createJdbcRowSet();
        rowSet.setUrl("jdbc:oracle:thin:@localhost:1521:xe");
        rowSet.setUsername("system");
        rowSet.setPassword("oracle");

        rowSet.setCommand("select * from emp400");
```

```

rowSet.execute();

while (rowSet.next()) {
    // Generating cursor Moved event
    System.out.println("Id: " + rowSet.getString(1));
    System.out.println("Name: " + rowSet.getString(2));
    System.out.println("Salary: " + rowSet.getString(3));
}

}

```

The output is given below:

```

Id: 55
Name: Om Bhim
Salary: 70000
Id: 190
Name: abhi
Salary: 40000
Id: 191
Name: umesh
Salary: 50000

```

---

## Full example of Jdbc RowSet with event handling

To perform event handling with JdbcRowSet, you need to add the instance of **RowSetListener** in the addRowSetListener method of JdbcRowSet.

The RowSetListener interface provides 3 method that must be implemented. They are as follows:

- 1) public void cursorMoved(RowSetEvent event);
- 2) public void rowChanged(RowSetEvent event);
- 3) public void rowSetChanged(RowSetEvent event);

Let's write the code to retrieve the data and perform some additional tasks while cursor is moved, cursor is changed or rowset is changed. The event handling operation can't be performed using ResultSet so it is preferred now.

```

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.Statement;
import javax.sql.RowSetEvent;
import javax.sql.RowSetListener;
import javax.sql.rowset.JdbcRowSet;
import javax.sql.rowset.RowSetProvider;

public class RowSetExample {
    public static void main(String[] args) throws Exception {
        Class.forName("oracle.jdbc.driver.OracleDriver");

        //Creating and Executing RowSet
        JdbcRowSet rowSet = RowSetProvider.newFactory().createJdbcRowSet();
        rowSet.setUrl("jdbc:oracle:thin:@localhost:1521:xe");
        rowSet.setUsername("system");
        rowSet.setPassword("oracle");

        rowSet.setCommand("select * from emp400");
        rowSet.execute();

        //Adding Listener and moving RowSet
        rowSet.addRowSetListener(new MyListener());
    }
}

```

```

        while (rowSet.next()) {
            // Generating cursor Moved event
            System.out.println("Id: " + rowSet.getString(1));
            System.out.println("Name: " + rowSet.getString(2));
            System.out.println("Salary: " + rowSet.getString(3));
        }

    }

class MyListener implements RowSetListener {
    public void cursorMoved(RowSetEvent event) {
        System.out.println("Cursor Moved...");
    }
    public void rowChanged(RowSetEvent event) {
        System.out.println("Cursor Changed...");
    }
    public void rowSetChanged(RowSetEvent event) {
        System.out.println("RowSet changed...");
    }
}

```

The output is as follows:

```

Cursor Moved...
Id: 55
Name: Om Bhim
Salary: 70000
Cursor Moved...
Id: 190
Name: abhi
Salary: 40000
Cursor Moved...
Id: 191
Name: umesh
Salary: 50000
Cursor Moved...

```

## New Features in Java

There are many new features that have been added in java. There are major enhancement made in Java5, Java6 and Java7 like **auto-boxing, generics, var-args, java annotations, enum, premain method** etc.

Most of the interviewers ask questions from this chapter.

## J2SE 4 Features

The important feature of J2SE 4 is assertions. It is used for testing.

- [Assertion \(Java 4\)](#)

---

## J2SE 5 Features

The important features of J2SE 5 are generics and assertions. Others are auto-boxing, enum, var-args, static import, for-each loop

(enhanced for loop etc.

- For-each loop (Java 5)
  - Varargs (Java 5)
  - Static Import (Java 5)
  - Autoboxing and Unboxing (Java 5)
  - Enum (Java 5)
  - Covariant Return Type (Java 5)
  - Annotation (Java 5)
  - Generics (Java 5)
- 

## JavaSE 6 Features

The important feature of JavaSE 6 is premain method (also known as instrumentation).

- Instrumentation (premain method) (Java 6)
- 

## JavaSE 7 Features

The important features of JavaSE 7 are try with resource, catching multiple exceptions etc.

- String in switch statement (Java 7)
- Binary Literals (Java 7)
- The try-with-resources (Java 7)
- Caching Multiple Exceptions by single catch (Java 7)
- Underscores in Numeric Literals (Java 7)

## Assertion:

Assertion is a statement in java. It can be used to test your assumptions about the program.

While executing assertion, it is believed to be true. If it fails, JVM will throw an error named AssertionError. It is mainly used for testing purpose.

## Advantage of Assertion:

It provides an effective way to detect and correct programming errors.

---

## Syntax of using Assertion:

There are two ways to use assertion. First way is:

```
assert expression;
```

and second way is:

```
assert expression1 : expression2;
```

---

## Simple Example of Assertion in java:

```
import java.util.Scanner;  
  
class AssertionExample{
```

```

public static void main( String args[] ){

Scanner scanner = new Scanner( System.in );
System.out.print("Enter ur age ");

int value = scanner.nextInt();
assert value>=18:" Not valid";

System.out.println("value is "+value);
}
}

```

[download this example](#)

If you use assertion, It will not run simply because assertion is disabled by default. To enable the assertion, **-ea** or **-enableassertions** switch of java must be used.

Compile it by: **javac AssertionExample.java**

Run it by: **java -ea AssertionExample**

---

```

Output: Enter ur age 11
        Exception in thread "main" java.lang.AssertionError: Not valid

```

## Where not to use Assertion:

There are some situations where assertion should be avoid to use. They are:

1. According to Sun Specification, assertion should not be used to check arguments in the public methods because it should result in appropriate runtime exception e.g. `IllegalArgumentException`, `NullPointerException` etc.
2. Do not use assertion, if you don't want any error in any situation.

## For-each loop (Advanced or Enhanced For loop):

The for-each loop introduced in Java5. It is mainly used to traverse array or collection elements. The advantage of for-each loop is that it eliminates the possibility of bugs and makes the code more readable.

### Advantage of for-each loop:

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

---

### Syntax of for-each loop:

```
for(data_type variable : array | collection){}
```

---

### Simple Example of for-each loop for traversing the array elements:

```

class ForEachExample1{
    public static void main(String args[]){
        int arr[]={12,13,14,44};
    }
}

```

```
for(int i:arr){  
    System.out.println(i);  
}  
  
}  
}
```

**Test it Now**

Output:12

```
13  
14  
44
```

## Simple Example of for-each loop for traversing the collection elements:

```
import java.util.*;  
class ForEachExample2{  
    public static void main(String args[]){  
        ArrayList list=new ArrayList();  
        list.add("vimal");  
        list.add("sonoo");  
        list.add("ratan");  
  
        for(String s:list){  
            System.out.println(s);  
        }  
    }  
}
```

**Test it Now**

Output:vimal

```
sonoo  
ratan
```

[download this example](#)

## Variable Argument (Varargs):

The varargs allows the method to accept zero or multiple arguments. Before varargs either we use overloaded method or take an array as the method parameter but it was not considered good because it leads to the maintenance problem. If we don't know how many argument we will have to pass in the method, varargs is the better approach.

### Advantage of Varargs:

We don't have to provide overloaded methods so less code.

### Syntax of varargs:

The varargs uses ellipsis i.e. three dots after the data type. Syntax is as follows:

```
return_type method_name(data_type... variableName){}
```

## Simple Example of Varargs in java:

```
class VarargsExample1{  
  
    static void display(String... values){  
        System.out.println("display method invoked ");  
    }  
  
    public static void main(String args[]){  
  
        display(); //zero argument  
        display("my", "name", "is", "varargs"); //four arguments  
    }  
}
```

**Test it Now**

Output: display method invoked  
display method invoked

## Another Program of Varargs in java:

```
class VarargsExample2{  
  
    static void display(String... values){  
        System.out.println("display method invoked ");  
        for(String s:values){  
            System.out.println(s);  
        }  
    }  
  
    public static void main(String args[]){  
  
        display(); //zero argument  
        display("hello"); //one argument  
        display("my", "name", "is", "varargs"); //four arguments  
    }  
}
```

**Test it Now**

Output: display method invoked  
display method invoked  
hello  
display method invoked  
my  
name  
is  
varargs

[download this example](#)

## Rules for varargs:

While using the varargs, you must follow some rules otherwise program code won't compile. The rules are as follows:

- There can be only one variable argument in the method.
- Variable argument (varargs) must be the last argument.

## Examples of varargs that fails to compile:

```
void method(String... a, int... b){}//Compile time error

void method(int... a, String b){}//Compile time error
```

## Example of Varargs that is the last argument in the method:

```
class VarargsExample3{

    static void display(int num, String... values){
        System.out.println("number is "+num);
        for(String s:values){
            System.out.println(s);
        }
    }

    public static void main(String args[]){
        display(500,"hello");//one argument
        display(1000,"my","name","is","varargs");//four arguments
    }
}
```

[Test it Now](#)

```
Output:number is 500
    hello
    number is 1000
    my
    name
    is
    varargs
```

[download this example](#)

## Static Import:

The static import feature of Java 5 facilitate the java programmer to access any static member of a class directly. There is no need to qualify it by the class name.

## Advantage of static import:

- Less coding is required if you have access any static member of a class oftenly.

## Disadvantage of static import:

- If you overuse the static import feature, it makes the program unreadable and unmaintainable.

---

## Simple Example of static import

```

import static java.lang.System.*;
class StaticImportExample{
    public static void main(String args[]){
        out.println("Hello");//Now no need of System.out
        out.println("Java");
    }
}

```

**Test it Now**

Output:Hello  
Java

## What is the difference between import and static import?

The import allows the java programmer to access classes of a package without package qualification whereas the static import feature allows to access the static members of a class without the class qualification. The import provides accessibility to classes and interface whereas static import provides accessibility to static members of the class.

## Autoboxing and Unboxing:

The automatic conversion of primitive data types into its equivalent Wrapper type is known as boxing and opposite operation is known as unboxing. This is the new feature of Java5. So java programmer doesn't need to write the conversion code.

### Advantage of Autoboxing and Unboxing:

No need of conversion between primitives and Wrappers manually so less coding is required.

---

### Simple Example of Autoboxing in java:

```

class BoxingExample1{
    public static void main(String args[]){
        int a=50;
        Integer a2=new Integer(a);//Boxing
        Integer a3=5;//Boxing
        System.out.println(a2+" "+a3);
    }
}

```

**Test it Now**

Output:50 5

[download this example](#)

---

### Simple Example of Unboxing in java:

The automatic conversion of wrapper class type into corresponding primitive type, is known as Unboxing. Let's see the example of unboxing:

```
class UnboxingExample1{
    public static void main(String args[]){
        Integer i=new Integer(50);
        int a=i;

        System.out.println(a);
    }
}
```

**Test it Now**

Output:50

---

## Autoboxing and Unboxing with comparison operators

Autoboxing can be performed with comparison operators. Let's see the example of boxing with comparison operator:

```
class UnboxingExample2{
    public static void main(String args[]){
        Integer i=new Integer(50);

        if(i<100){           //unboxing internally
            System.out.println(i);
        }
    }
}
```

**Test it Now**

Output:50

---

## Autoboxing and Unboxing with method overloading

In method overloading, boxing and unboxing can be performed. There are some rules for method overloading with boxing:

- **Widening beats boxing**
- **Widening beats varargs**
- **Boxing beats varargs**

### 1) Example of Autoboxing where widening beats boxing

If there is possibility of widening and boxing, widening beats boxing.

```
class Boxing1{
    static void m(int i){System.out.println("int");}
    static void m(Integer i){System.out.println("Integer");}

    public static void main(String args[]){
        short s=30;
        m(s);
    }
}
```

**Test it Now**

Output:int

---

## 2) Example of Autoboxing where widening beats varargs

If there is possibility of widening and varargs, widening beats var-args.

```
class Boxing2{  
    static void m(int i, int i2){System.out.println("int int");}  
    static void m(Integer... i){System.out.println("Integer...");}  
  
    public static void main(String args[]){  
        short s1=30,s2=40;  
        m(s1,s2);  
    }  
}
```

**Test it Now**

Output:int int

## 3) Example of Autoboxing where boxing beats varargs

Let's see the program where boxing beats variable argument:

```
class Boxing3{  
    static void m(Integer i){System.out.println("Integer");}  
    static void m(Integer... i){System.out.println("Integer...");}  
  
    public static void main(String args[]){  
        int a=30;  
        m(a);  
    }  
}
```

**Test it Now**

Output:Integer

## Method overloading with Widening and Boxing

Widening and Boxing can't be performed as given below:

```
class Boxing4{  
    static void m(Long l){System.out.println("Long");}  
  
    public static void main(String args[]){  
        int a=30;  
        m(a);  
    }  
}
```

**Test it Now**

Output:Compile Time Error

# Java Enum

**Enum in java** is a data type that contains fixed set of constants.

It can be used for days of the week (SUNDAY, MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY and SATURDAY) , directions (NORTH, SOUTH, EAST and WEST) etc. The java enum constants are static and final implicitly. It is available from JDK 1.5.

Java Enums can be thought of as classes that have fixed set of constants.

## Points to remember for Java Enum

- enum improves type safety
- enum can be easily used in switch
- enum can be traversed
- enum can have fields, constructors and methods
- enum may implement many interfaces but cannot extend any class because it internally extends Enum class

## Simple example of java enum

```
class EnumExample1{  
    public enum Season { WINTER, SPRING, SUMMER, FALL }  
  
    public static void main(String[] args) {  
        for (Season s : Season.values())  
            System.out.println(s);  
    }  
}
```

**Test it Now**

Output:  
WINTER  
SPRING  
SUMMER  
FALL

[download this enum example](#)

## What is the purpose of values() method in enum?

The java compiler internally adds the values() method when it creates an enum. The values() method returns an array containing all the values of the enum.

## Internal code generated by the compiler for the above example of enum type

The java compiler internally creates a static and final class that extends the Enum class as shown in the below example:

```
public static final class EnumExample1$Season extends Enum  
{  
    private EnumExample1$Season(String s, int i)  
    {  
        super(s, i);  
    }  
  
    public static EnumExample1$Season[] values()  
    {  
        return (EnumExample1$Season[])$VALUES.clone();  
    }  
  
    public static EnumExample1$Season valueOf(String s)  
    {
```

```

        return (EnumExample1$Season)Enum.valueOf(EnumExample1$Season, s);
    }

    public static final EnumExample1$Season WINTER;
    public static final EnumExample1$Season SPRING;
    public static final EnumExample1$Season SUMMER;
    public static final EnumExample1$Season FALL;
    private static final EnumExample1$Season $VALUES[];

    static
    {
        WINTER = new EnumExample1$Season("WINTER", 0);
        SPRING = new EnumExample1$Season("SPRING", 1);
        SUMMER = new EnumExample1$Season("SUMMER", 2);
        FALL = new EnumExample1$Season("FALL", 3);
        $VALUES = (new EnumExample1$Season[] {
            WINTER, SPRING, SUMMER, FALL
        });
    }
}

```

---

## Defining Java enum

The enum can be defined within or outside the class because it is similar to a class.

### Java enum example: defined outside class

```

enum Season { WINTER, SPRING, SUMMER, FALL }
class EnumExample2{
    public static void main(String[] args) {
        Season s=Season.WINTER;
        System.out.println(s);
    }
}

```

**Test it Now**

Output:WINTER

### Java enum example: defined inside class

```

class EnumExample3{
    enum Season { WINTER, SPRING, SUMMER, FALL; } //semicolon(;) is optional here
    public static void main(String[] args) {
        Season s=Season.WINTER;//enum type is required to access WINTER
        System.out.println(s);
    }
}

```

**Test it Now**

Output:WINTER

## Initializing specific values to the enum constants

The enum constants have initial value that starts from 0, 1, 2, 3 and so on. But we can initialize the specific value to the enum constants by defining fields and constructors. As specified earlier, Enum can have fields, constructors and methods.

### Example of specifying initial value to the enum constants

```
class EnumExample4{
```

```

enum Season{
    WINTER(5), SPRING(10), SUMMER(15), FALL(20);

    private int value;
    private Season(int value){
        this.value=value;
    }
}
public static void main(String args[]){
    for (Season s : Season.values())
        System.out.println(s+" "+s.value);

}}

```

### Test it Now

[download this enum example](#)

Output:WINTER 5  
 SPRING 10  
 SUMMER 15  
 FALL 20

 **Constructor of enum type is private. If you don't declare private compiler internally creates private constructor.**

```

enum Season{
    WINTER(10),SUMMER(20);
    private int value;
    Season(int value){
        this.value=value;
    }
}

```

Internal code generated by the compiler for the above example of enum type

```

final class Season extends Enum
{
    public static Season[] values()
    {
        return (Season[])$VALUES.clone();
    }
    public static Season valueOf(String s)
    {
        return (Season)Enum.valueOf(Season, s);
    }
    private Season(String s, int i, int j)
    {
        super(s, i);
        value = j;
    }
    public static final Season WINTER;
    public static final Season SUMMER;
    private int value;
    private static final Season $VALUES[];
    static
    {
        WINTER = new Season("WINTER", 0, 10);
        SUMMER = new Season("SUMMER", 1, 20);
        $VALUES = (new Season[] {
            WINTER, SUMMER
        })
    }
}

```

```
});  
}  
}
```

---

## Can we create the instance of enum by new keyword?

No, because it contains private constructors only.

---

## Can we have abstract method in enum?

Yes, ofcourse! we can have abstract methods and can provide the implementation of these methods.

---

## Java enum in switch statement

We can apply enum on switch statement as in the given example:

### Example of applying enum on switch statement

```
class EnumExample5{  
enum Day{ SUNDAY, MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY, SATURDAY}  
public static void main(String args[]){  
Day day=Day.MONDAY;  
  
switch(day){  
case SUNDAY:  
System.out.println("sunday");  
break;  
case MONDAY:  
System.out.println("monday");  
break;  
default:  
System.out.println("other day");  
}  
}}
```

[Test it Now](#)

[download this enum example](#)

Output:monday

## Java Annotations

Java **Annotation** is a tag that represents the *metadata* i.e. attached with class, interface, methods or fields to indicate some additional information which can be used by java compiler and JVM.

Annotations in java are used to provide additional information, so it is an alternative option for XML and java marker interfaces.

First, we will learn some built-in annotations then we will move on creating and using custom annotations.

---

## Built-In Java Annotations

There are several built-in annotations in java. Some annotations are applied to java code and some to other annotations.

### Built-In Java Annotations used in java code

- @Override

- @SuppressWarnings
- @Deprecated

## Built-In Java Annotations used in other annotations

- @Target
- @Retention
- @Inherited
- @Documented

# Understanding Built-In Annotations in java

Let's understand the built-in annotations first.

## @Override

@Override annotation assures that the subclass method is overriding the parent class method. If it is not so, compile time error occurs.

Sometimes, we do silly mistakes such as spelling mistakes etc. So, it is better to mark @Override annotation that provides assurance that method is overridden.

```
class Animal{  
void eatSomething(){System.out.println("eating something");}  
}  
  
class Dog extends Animal{  
@Override  
void eatsomething(){System.out.println("eating foods");}//should be eatSomething  
}  
  
class TestAnnotation1{  
public static void main(String args[]){  
Animal a=new Dog();  
a.eatSomething();  
}}
```

**Test it Now**

Output: Compile Time Error

## @SuppressWarnings

@SuppressWarnings annotation: is used to suppress warnings issued by the compiler.

```
import java.util.*;  
class TestAnnotation2{  
@SuppressWarnings("unchecked")  
public static void main(String args[]){  
ArrayList list=new ArrayList();  
list.add("sonoo");  
list.add("vimal");  
list.add("ratan");  
  
for(Object obj:list)  
System.out.println(obj);  
}}
```

**Test it Now**

Now no warning at compile time.

If you remove the @SuppressWarnings("unchecked") annotation, it will show warning at compile time because we are using non-generic collection.

## @Deprecated

@Deprecated annotation marks that this method is deprecated so compiler prints warning. It informs user that it may be removed in the future versions. So, it is better not to use such methods.

```
class A{  
void m(){System.out.println("hello m");}  
  
@Deprecated  
void n(){System.out.println("hello n");}  
}  
  
class TestAnnotation3{  
public static void main(String args[]){  
  
A a=new A();  
a.n();  
}}
```

**Test it Now**

### At Compile Time:

Note: Test.java uses or overrides a deprecated API.

Note: Recompile with -Xlint:deprecation for details.

### At Runtime:

hello n

## Custom Annotation

To create and use custom java annotation, visit the next page.

## Java Custom Annotation

**Java Custom annotations** or Java User-defined annotations are easy to create and use. The `@interface` element is used to declare an annotation. For example:

```
@interface MyAnnotation{}
```

Here, MyAnnotation is the custom annotation name.

### Points to remember for java custom annotation signature

There are few points that should be remembered by the programmer.

1. Method should not have any throws clauses
2. Method should return one of the following: primitive data types, String, Class, enum or array of these data types.
3. Method should not have any parameter.
4. We should attach `@` just before interface keyword to define annotation.
5. It may assign a default value to the method.

## Types of Annotation

There are three types of annotations.

1. Marker Annotation
2. Single-Value Annotation

### 3. Multi-Value Annotation

#### 1) Marker Annotation

An annotation that has no method, is called marker annotation. For example:

```
@interface MyAnnotation{}
```

The @Override and @Deprecated are marker annotations.

---

#### 2) Single-Value Annotation

An annotation that has one method, is called single-value annotation. For example:

```
@interface MyAnnotation{  
    int value();  
}
```

We can provide the default value also. For example:

```
@interface MyAnnotation{  
    int value() default 0;  
}
```

#### How to apply Single-Value Annotation

Let's see the code to apply the single value annotation.

```
@MyAnnotation(value=10)
```

The value can be anything.

---

#### 3) Mult-Value Annotation

An annotation that has more than one method, is called Multi-Value annotation. For example:

```
@interface MyAnnotation{  
    int value1();  
    String value2();  
    String value3();  
}
```

We can provide the default value also. For example:

```
@interface MyAnnotation{  
    int value1() default 1;  
    String value2() default "";  
    String value3() default "xyz";  
}
```

#### How to apply Multi-Value Annotation

Let's see the code to apply the multi-value annotation.

```
@MyAnnotation(value1=10,value2="Arun Kumar",value3="Ghaziabad")
```

---

### Built-in Annotations used in custom annotations in java

- @Target
- @Retention
- @Inherited
- @Documented

---

#### @Target

**@Target** tag is used to specify at which type, the annotation is used.

The `java.lang.annotation.ElementType` enum declares many constants to specify the type of element where annotation is to be applied such as TYPE, METHOD, FIELD etc. Let's see the constants of ElementType enum:

Element Types	Where the annotation can be applied
TYPE	class, interface or enumeration
FIELD	fields
METHOD	methods
CONSTRUCTOR	constructors
LOCAL_VARIABLE	local variables
ANNOTATION_TYPE	annotation type
PARAMETER	parameter

### Example to specify annoation for a class

```
@Target(ElementType.TYPE)
@interface MyAnnotation{
    int value1();
    String value2();
}
```

### Example to specify annoation for a class, methods or fields

```
@Target({ElementType.TYPE, ElementType.FIELD, ElementType.METHOD})
@interface MyAnnotation{
    int value1();
    String value2();
}
```

## @Retention

**@Retention** annotation is used to specify to what level annotation will be available.

RetentionPolicy	Availability
RetentionPolicy.SOURCE	refers to the source code, discarded during compilation. It will not be available in the compiled class.
RetentionPolicy.CLASS	refers to the .class file, available to java compiler but not to JVM . It is included in the class file.
RetentionPolicy.RUNTIME	refers to the runtime, available to java compiler and JVM .

### Example to specify the RetentionPolicy

```
@Retention(RetentionPolicy.RUNTIME)
@Target(ElementType.TYPE)
@interface MyAnnotation{
    int value1();
    String value2();
}
```

## Example of custom annotation: creating, applying and accessing annotation

Let's see the simple example of creating, applying and accessing annotation.

*File: Test.java*

```
//Creating annotation
import java.lang.annotation.*;
import java.lang.reflect.*;
```

```

@Retention(RetentionPolicy.RUNTIME)
@Target(ElementType.METHOD)
@interface MyAnnotation{
    int value();
}

//Applying annotation
class Hello{
    @MyAnnotation(value=10)
    public void sayHello(){System.out.println("hello annotation");}
}

//Accessing annotation
class TestCustomAnnotation1{
    public static void main(String args[])throws Exception{

        Hello h=new Hello();
        Method m=h.getClass().getMethod("sayHello");

        MyAnnotation manno=m.getAnnotation(MyAnnotation.class);
        System.out.println("value is: "+manno.value());
    }
}

```

**Test it Now**

Output:value is: 10

---

[download this example](#)

### How built-in annotations are used in real scenario?

In real scenario, java programmer only need to apply annotation. He/She doesn't need to create and access annotation. Creating and Accessing annotation is performed by the implementation provider. On behalf of the annotation, java compiler or JVM performs some additional operations.

---

### @Inherited

By default, annotations are not inherited to subclasses. The `@Inherited` annotation marks the annotation to be inherited to subclasses.

```

@Inherited
interface ForEveryone { } //Now it will be available to subclass also

interface ForEveryone { }
class Superclass{};

class Subclass extends Superclass{}

```

---

### @Documented

The `@Documented` Marks the annotation for inclusion in the documentation.

## Generics in Java

The **Java Generics** programming is introduced in J2SE 5 to deal with type-safe objects.

Before generics, we can store any type of objects in collection i.e. non-generic. Now generics, forces the java programmer to store specific type of objects.

## Advantage of Java Generics

There are mainly 3 advantages of generics. They are as follows:

**1) Type-safety :** We can hold only a single type of objects in generics. It doesn't allow to store other objects.

**2) Type casting is not required:** There is no need to typecast the object.

Before Generics, we need to type cast.

```
List list = new ArrayList();
list.add("hello");
String s = (String) list.get(0); //typecasting
```

After Generics, we don't need to typecast the object.

```
List<String> list = new ArrayList<String>();
list.add("hello");
String s = list.get(0);
```

**3) Compile-Time Checking:** It is checked at compile time so problem will not occur at runtime. The good programming strategy says it is far better to handle the problem at compile time than runtime.

```
List<String> list = new ArrayList<String>();
list.add("hello");
list.add(32); //Compile Time Error
```

**Syntax** to use generic collection

```
ClassOrInterface<Type>
```

**Example** to use Generics in java

```
ArrayList<String>
```

## Full Example of Generics in Java

Here, we are using the ArrayList class, but you can use any collection class such as ArrayList, LinkedList, HashSet, TreeSet, HashMap, Comparator etc.

```
import java.util.*;
class TestGenerics1{
public static void main(String args[]){
ArrayList<String> list=new ArrayList<String>();
list.add("rahul");
list.add("jai");
//list.add(32); //compile time error

String s=list.get(1); //type casting is not required
System.out.println("element is: "+s);
```

```
Iterator<String> itr=list.iterator();
while(itr.hasNext()){
System.out.println(itr.next());
}
```

**Test it Now**

```
Output:element is: jai
        rahul
        jai
```

## Example of Java Generics using Map

Now we are going to use map elements using generics. Here, we need to pass key and value. Let us understand it by a simple example:

```
import java.util.*;
class TestGenerics2{
public static void main(String args[]){
Map<Integer, String> map=new HashMap<Integer, String>();
map.put(1,"vijay");
map.put(4,"umesh");
map.put(2,"ankit");

//Now use Map.Entry for Set and Iterator
Set<Map.Entry<Integer, String>> set=map.entrySet();

Iterator<Map.Entry<Integer, String>> itr=set.iterator();
while(itr.hasNext()){
Map.Entry e=itr.next(); //no need to typecast
System.out.println(e.getKey()+" "+e.getValue());
}
}}
```

**Test it Now**

Output:  
1 vijay  
2 ankit  
4 umesh

## Generic class

A class that can refer to any type is known as generic class. Here, we are using **T** type parameter to create the generic class of specific type.

Let's see the simple example to create and use the generic class.

### Creating generic class:

```
class MyGen<T>{
T obj;
void add(T obj){this.obj=obj;}
T get(){return obj;}
}
```

The T type indicates that it can refer to any type (like String, Integer, Employee etc.). The type you specify for the class, will be used to store and retrieve the data.

### Using generic class:

Let's see the code to use the generic class.

```
class TestGenerics3{
public static void main(String args[]){
MyGen<Integer> m=new MyGen<Integer>();
m.add(2);
//m.add("vivek");//Compile time error
System.out.println(m.get());
}}

```

Output:2

## Type Parameters

The type parameters naming conventions are important to learn generics thoroughly. The commonly type parameters are as follows:

1. T - Type
2. E - Element

- 
- 3. K - Key
  - 4. N - Number
  - 5. V - Value
- 

## Generic Method

Like generic class, we can create generic method that can accept any type of argument.

Let's see a simple example of java generic method to print array elements. We are using here **E** to denote the element.

```
public class TestGenerics4{

    public static < E > void printArray(E[] elements) {
        for ( E element : elements){
            System.out.println(element );
        }
        System.out.println();
    }

    public static void main( String args[] ) {
        Integer[] intArray = { 10, 20, 30, 40, 50 };
        Character[] charArray = { 'J', 'A', 'V', 'A', 'T','P','O','I','N','T' };

        System.out.println( "Printing Integer Array" );
        printArray( intArray );

        System.out.println( "Printing Character Array" );
        printArray( charArray );
    }
}
```

**Test it Now**

Output:Printing Integer Array

```
10
20
30
40
50
Printing Character Array
J
A
V
A
T
P
O
I
N
T
```

---

## Wildcard in Java Generics

The ? (question mark) symbol represents wildcard element. It means any type. If we write `<? extends Number>`, it means any child class of Number e.g. Integer, Float, double etc. Now we can call the method of Number class through any child class object.

Let's understand it by the example given below:

```
import java.util.*;
abstract class Shape{
    abstract void draw();
}
class Rectangle extends Shape{
```

```

void draw(){System.out.println("drawing rectangle");}
}
class Circle extends Shape{
void draw(){System.out.println("drawing circle");}
}

class GenericTest{
//creating a method that accepts only child class of Shape
public static void drawShapes(List lists){
for(Shape s:lists){
s.draw(); //calling method of Shape class by child class instance
}
}
public static void main(String args[]){
List list1=new ArrayList();
list1.add(new Rectangle());

List list2=new ArrayList();
list2.add(new Circle());
list2.add(new Circle());

drawShapes(list1);
drawShapes(list2);
}}
drawing rectangle
drawing circle
drawing circle

```

## RMI (Remote Method Invocation)

The **RMI** (Remote Method Invocation) is an API that provides a mechanism to create distributed application in java. The RMI allows an object to invoke methods on an object running in another JVM.

The RMI provides remote communication between the applications using two objects *stub* and *skeleton*.

### Understanding stub and skeleton

RMI uses stub and skeleton object for communication with the remote object.

A **remote object** is an object whose method can be invoked from another JVM. Let's understand the stub and skeleton objects:

#### **stub**

The stub is an object, acts as a gateway for the client side. All the outgoing requests are routed through it. It resides at the client side and represents the remote object. When the caller invokes method on the stub object, it does the following tasks:

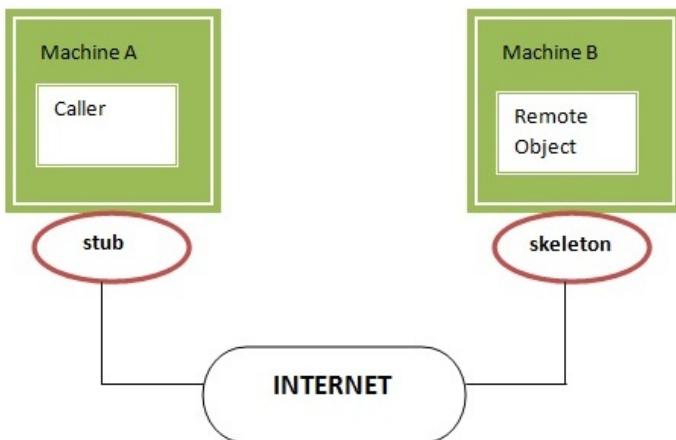
1. It initiates a connection with remote Virtual Machine (JVM),
2. It writes and transmits (marshals) the parameters to the remote Virtual Machine (JVM),
3. It waits for the result
4. It reads (unmarshals) the return value or exception, and
5. It finally, returns the value to the caller.

#### **skeleton**

The skeleton is an object, acts as a gateway for the server side object. All the incoming requests are routed through it. When the skeleton receives the incoming request, it does the following tasks:

1. It reads the parameter for the remote method
2. It invokes the method on the actual remote object, and
3. It writes and transmits (marshals) the result to the caller.

In the Java 2 SDK, a stub protocol was introduced that eliminates the need for skeletons.



## Understanding requirements for the distributed applications

If any application performs these tasks, it can be distributed application.

1. The application need to locate the remote method
2. It need to provide the communication with the remote objects, and
3. The application need to load the class definitions for the objects.

The RMI application have all these features, so it is called the distributed application.

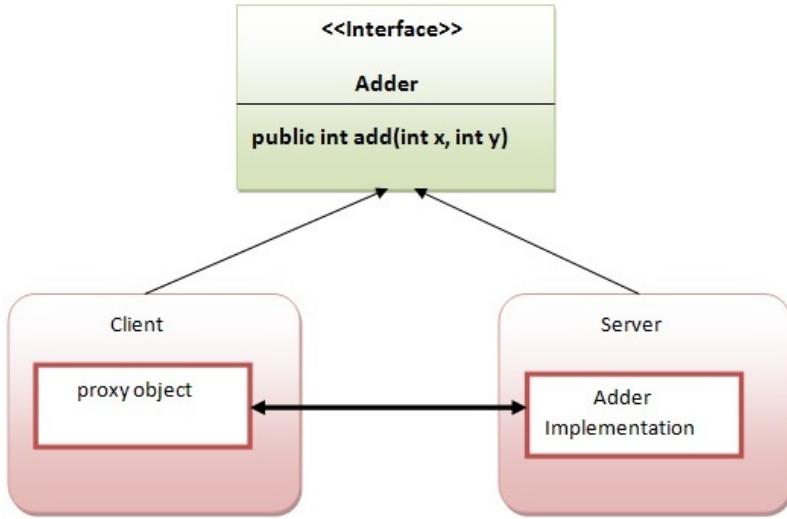
## Steps to write the RMI program

There is given the 6 steps to write the RMI program.

1. Create the remote interface
2. Provide the implementation of the remote interface
3. Compile the implementation class and create the stub and skeleton objects using the rmic tool
4. Start the registry service by rmiregistry tool
5. Create and start the remote application
6. Create and start the client application

## RMI Example

In this example, we have followed all the 6 steps to create and run the rmi application. The client application need only two files, remote interface and client application. In the rmi application, both client and server interacts with the remote interface. The client application invokes methods on the proxy object, RMI sends the request to the remote JVM. The return value is sent back to the proxy object and then to the client application.



## 1) create the remote interface

For creating the remote interface, extend the Remote interface and declare the RemoteException with all the methods of the remote interface. Here, we are creating a remote interface that extends the Remote interface. There is only one method named add() and it declares RemoteException.

```

import java.rmi.*;
public interface Adder extends Remote{
    public int add(int x,int y) throws RemoteException;
}

```

## 2) Provide the implementation of the remote interface

Now provide the implementation of the remote interface. For providing the implementation of the Remote interface, we need to

- Either extend the UnicastRemoteObject class,
- or use the exportObject() method of the UnicastRemoteObject class

In case, you extend the UnicastRemoteObject class, you must define a constructor that declares RemoteException.

```

import java.rmi.*;
import java.rmi.server.*;
public class AdderRemote extends UnicastRemoteObject implements Adder{
    AdderRemote() throws RemoteException{
        super();
    }
    public int add(int x,int y){return x+y;}
}

```

## 3) create the stub and skeleton objects using the rmic tool.

Next step is to create stub and skeleton objects using the rmi compiler. The rmic tool invokes the RMI compiler and creates stub and skeleton objects.

```
rmic AdderRemote
```

## 4) Start the registry service by the rmiregistry tool

Now start the registry service by using the rmiregistry tool. If you don't specify the port number, it uses a default port number. In this example, we are using the port number 5000.

```
rmiregistry 5000
```

## 5) Create and run the server application

Now rmi services need to be hosted in a server process. The Naming class provides methods to get and store the remote object. The Naming class provides 5 methods.

1. **public static java.rmi.Remote lookup(java.lang.String) throws java.rmi.NotBoundException, java.net.MalformedURLException, java.rmi.RemoteException;** it returns the reference of the remote object.
2. **public static void bind(java.lang.String, java.rmi.Remote) throws java.rmi.AlreadyBoundException, java.net.MalformedURLException, java.rmi.RemoteException;** it binds the remote object with the given name.
3. **public static void unbind(java.lang.String) throws java.rmi.RemoteException, java.rmi.NotBoundException, java.net.MalformedURLException;** it destroys the remote object which is bound with the given name.
4. **public static void rebind(java.lang.String, java.rmi.Remote) throws java.rmi.RemoteException, java.net.MalformedURLException;** it binds the remote object to the new name.
5. **public static java.lang.String[] list(java.lang.String) throws java.rmi.RemoteException, java.net.MalformedURLException;** it returns an array of the names of the remote objects bound in the registry.

In this example, we are binding the remote object by the name sonoo.

```
import java.rmi.*;
import java.rmi.registry.*;
public class MyServer{
public static void main(String args[]){
try{
Adder stub=new AdderRemote();
Naming.rebind("rmi://localhost:5000/sonoo",stub);
}catch(Exception e){System.out.println(e);}
}
}
```

---

## 6) Create and run the client application

At the client we are getting the stub object by the lookup() method of the Naming class and invoking the method on this object. In this example, we are running the server and client applications, in the same machine so we are using localhost. If you want to access the remote object from another machine, change the localhost to the host name (or IP address) where the remote object is located.

```
import java.rmi.*;
public class MyClient{
public static void main(String args[]){
try{
Adder stub=(Adder)Naming.lookup("rmi://localhost:5000/sonoo");
System.out.println(stub.add(34,4));
}catch(Exception e){}
}
}
```

---

[download this example of rmi](#)

For running this rmi example,

```
1) compile all the java files
javac *.java

2)create stub and skeleton object by rmic tool
rmic AdderRemote

3)start rmi registry in one command prompt
rmiregistry 5000
```

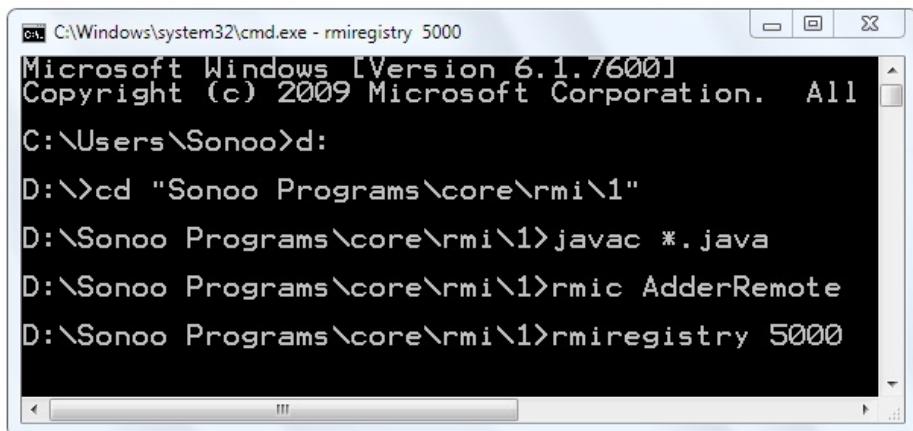
4) start the server in another command prompt

```
java MyServer
```

5) start the client application in another command prompt

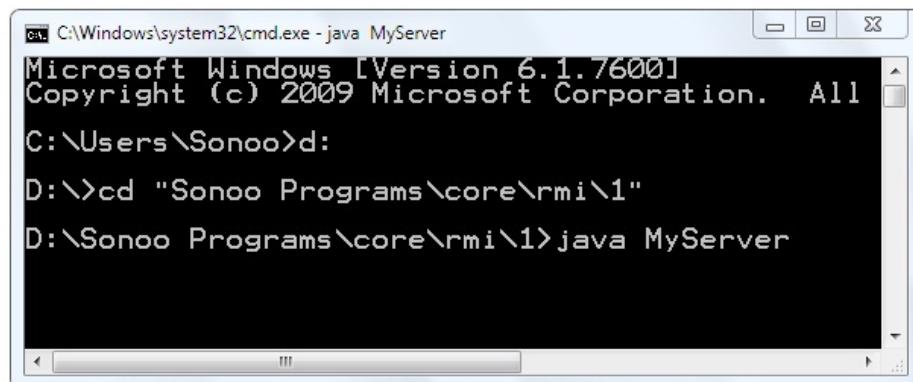
```
java MyClient
```

## Output of this RMI example



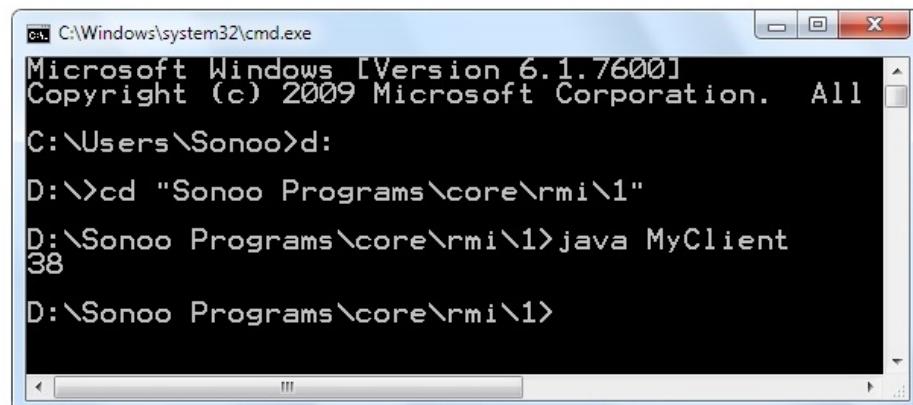
```
C:\Windows\system32\cmd.exe - rmiregistry 5000
Microsoft Windows [Version 6.1.7600]
Copyright (c) 2009 Microsoft Corporation. All rights reserved.

C:\Users\Sonoo>d:
D:\>cd "Sonoo Programs\core\rmi\1"
D:\Sonoo Programs\core\rmi\1>javac *.java
D:\Sonoo Programs\core\rmi\1>rmic AdderRemote
D:\Sonoo Programs\core\rmi\1>rmiregistry 5000
```



```
C:\Windows\system32\cmd.exe - java MyServer
Microsoft Windows [Version 6.1.7600]
Copyright (c) 2009 Microsoft Corporation. All rights reserved.

C:\Users\Sonoo>d:
D:\>cd "Sonoo Programs\core\rmi\1"
D:\Sonoo Programs\core\rmi\1>java MyServer
```



```
C:\Windows\system32\cmd.exe
Microsoft Windows [Version 6.1.7600]
Copyright (c) 2009 Microsoft Corporation. All rights reserved.

C:\Users\Sonoo>d:
D:\>cd "Sonoo Programs\core\rmi\1"
D:\Sonoo Programs\core\rmi\1>java MyClient
38
D:\Sonoo Programs\core\rmi\1>
```

## Meaningful example of RMI application with database

Consider a scenario, there are two applications running in different machines. Let's say MachineA and MachineB, machineA is located in United States and MachineB in India. MachineB want to get list of all the customers of MachineA application.

Let's develop the RMI application by following the steps.

### 1) Create the table

First of all, we need to create the table in the database. Here, we are using Oracle10 database.

CUSTOMER400					
Table	Data	Indexes	Model	Constraints	Grants
Query	Count Rows	Insert Row			Statistics
EDIT	ACC_NO	FIRSTNAME	LASTNAME	EMAIL	AMOUNT
	67539876	James	Franklin	franklin1james@gmail.com	500000
	67534876	Ravi	Kumar	ravimalik@gmail.com	98000
	67579872	Vimal	Jaiswal	jaiswalvimal32@gmail.com	9380000

## 2) Create Customer class and Remote interface

File: *Customer.java*

```
package com.javatpoint;
public class Customer implements java.io.Serializable{
    private int acc_no;
    private String firstname,lastname,email;
    private float amount;
    //getters and setters
}
```



**Note:** *Customer class must be Serializable.*

File: *Bank.java*

```
package com.javatpoint;
import java.rmi.*;
import java.util.*;
interface Bank extends Remote{
    public List<Customer> getCustomers()throws RemoteException;
}
```

## 3) Create the class that provides the implementation of Remote interface

File: *BankImpl.java*

```
package com.javatpoint;
import java.rmi.*;
import java.rmi.server.*;
import java.sql.*;
import java.util.*;
class BankImpl extends UnicastRemoteObject implements Bank{
    BankImpl()throws RemoteException{}

    public List<Customer> getCustomers(){
        List<Customer> list=new ArrayList();
        try{
            Class.forName("oracle.jdbc.driver.OracleDriver");
            Connection con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe","system","oracle");
            PreparedStatement ps=con.prepareStatement("select * from customer400");
            ResultSet rs=ps.executeQuery();

            while(rs.next()){

```

```

Customer c=new Customer();
c.setAcc_no(rs.getInt(1));
c.setFirstname(rs.getString(2));
c.setLastname(rs.getString(3));
c.setEmail(rs.getString(4));
c.setAmount(rs.getFloat(5));
list.add(c);
}

con.close();
}catch(Exception e){System.out.println(e);}
return list;
}//end of getCustomers()
}

```

**4) Compile the class rmic tool and start the registry service by rmiregistry tool**



```

C:\batch10we\rmidb>javac -d . Customer.java
C:\batch10we\rmidb>javac -d . BankImpl.java
C:\batch10we\rmidb>rmic com.javatpoint.BankImpl
C:\batch10we\rmidb>rmiregistry 6666

```

**5) Create and run the Server**

*File: MyServer.java*

```

package com.javatpoint;
import java.rmi.*;
public class MyServer{
public static void main(String args[])throws Exception{
Remote r=new BankImpl();
Naming.rebind("rmi://localhost:6666/javatpoint",r);
}}

```



```

C:\batch10we\rmidb>javac -d . MyServer.java
C:\batch10we\rmidb>java com.javatpoint.MyServer

```

**6) Create and run the Client**

*File: MyClient.java*

```

package com.javatpoint;
import java.util.*;
import java.rmi.*;

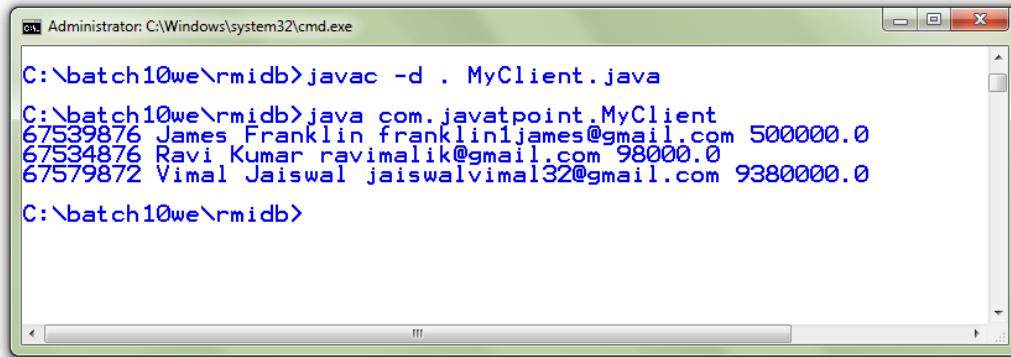
```

```

public class MyClient{
public static void main(String args[])throws Exception{
Bank b=(Bank)Naming.lookup("rmi://localhost:6666/javatpoint");

List<Customer> list=b.getCustomers();
for(Customer c:list){
System.out.println(c.getAcc_no()+" "+c.getFirstname()+" "+c.getLastname()
+" "+c.getEmail()+" "+c.getAmount());
}
}

}}
```



The screenshot shows a Windows Command Prompt window titled "Administrator: C:\Windows\system32\cmd.exe". The command entered is "javac -d . MyClient.java". The output shows three customer records being printed:

```

C:\batch10we\rmidb>javac -d . MyClient.java
C:\batch10we\rmidb>java com.javatpoint.MyClient
67539876 James Franklin franklinjames@gmail.com 500000.0
67534876 Ravi Kumar ravimalik@gmail.com 98000.0
67579872 Vimal Jaiswal jaiswalvimal32@gmail.com 9380000.0
C:\batch10we\rmidb>
```

[download this example of rmi with database](#)

## Internationalization and Localization in Java

**Internationalization** is also abbreviated as I18N because there are total 18 characters between the first letter 'I' and the last letter 'N'.

Internationalization is a mechanism to create such an application that can be adapted to different languages and regions.

Internationalization is one of the powerful concept of java if you are developing an application and want to display messages, currencies, date, time etc. according to the specific region or language.

**Localization** is also abbreviated as I10N because there are total 10 characters between the first letter 'L' and last letter 'N'. Localization is the mechanism to create such an application that can be adapted to a specific language and region by adding locale-specific text and component.

## Understanding the culturally dependent data before starting internationalization

Before starting the internationalization, Let's first understand what are the informations that differ from one region to another. There is the list of culturally dependent data:

- Messages
- Dates
- Times
- Numbers
- Currencies
- Measurements
- Phone Numbers
- Postal Addresses
- Labels on GUI components etc.

## Importance of Locale class in Internationalization

An object of Locale class represents a geographical or cultural region. This object can be used to get the locale specific information such as country name, language, variant etc.

## Fields of Locale class

There are fields of Locale class:

1. public static final Locale ENGLISH
2. public static final Locale FRENCH
3. public static final Locale GERMAN
4. public static final Locale ITALIAN
5. public static final Locale JAPANESE
6. public static final Locale KOREAN
7. public static final Locale CHINESE
8. public static final Locale SIMPLIFIED\_CHINESE
9. public static final Locale TRADITIONAL\_CHINESE
10. public static final Locale FRANCE
11. public static final Locale GERMANY
12. public static final Locale ITALY
13. public static final Locale JAPAN
14. public static final Locale KOREA
15. public static final Locale CHINA
16. public static final Locale PRC
17. public static final Locale TAIWAN
18. public static final Locale UK
19. public static final Locale US
20. public static final Locale CANADA
21. public static final Locale CANADA\_FRENCH
22. public static final Locale ROOT

## Constructors of Locale class

There are three constructors of Locale class. They are as follows:

1. Locale(String language)
2. Locale(String language, String country)
3. Locale(String language, String country, String variant)

## Commonly used methods of Locale class

There are given commonly used methods of Locale class.

1. **public static Locale getDefault()** it returns the instance of current Locale
2. **public static Locale[] getAvailableLocales()** it returns an array of available locales.
3. **public String getDisplayCountry()** it returns the country name of this locale object.
4. **public String getDisplayLanguage()** it returns the language name of this locale object.
5. **public String getDisplayVariant()** it returns the variant code for this locale object.
6. **public String getISO3Country()** it returns the three letter abbreviation for the current locale's country.
7. **public String getISO3Language()** it returns the three letter abbreviation for the current locale's language.

## Example of Local class that prints the informations of the default locale

In this example, we are displaying the informations of the default locale. If you want to get the informations about any specific locale, comment the first line statement and uncomment the second line statement in the main method.

```
import java.util.*;
public class LocaleExample {
public static void main(String[] args) {
Locale locale=Locale.getDefault();
//Locale locale=new Locale("fr","fr");//for the specific locale
```

```
System.out.println(locale.getDisplayCountry());
System.out.println(locale.getDisplayLanguage());
System.out.println(locale.getDisplayName());
System.out.println(locale.getISO3Country());
System.out.println(locale.getISO3Language());
System.out.println(locale.getLanguage());
System.out.println(locale.getCountry());

}
}
```

[download this example](#)

**Output:**United States  
English  
English (United States)  
USA  
eng  
en  
US

---

## Example of Local class that prints english in different languages

In this example, we are displaying english language in different language. Let's see how english is written in french and spanish languages.

```
import java.util.*;
public class LocaleExample2 {
    public static void main(String[] args) {
        Locale enLocale = new Locale("en", "US");
        Locale frLocale = new Locale("fr", "FR");
        Locale esLocale = new Locale("es", "ES");
        System.out.println("English language name (default): " +
                           enLocale.getDisplayLanguage());

        System.out.println("English language name in French: " +
                           enLocale.getDisplayLanguage(frLocale));
        System.out.println("English language name in spanish: " +
                           enLocale.getDisplayLanguage(esLocale));
    }
}
```

**Output:**English language name (default): English  
English language name in French: anglais  
English language name in spanish: ingl?s

---

## Example of Local class that print display language of many locales

In this example, we are displaying the display lanuage of many locales.

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

```

Locale[] locales = { new Locale("en", "US"),
new Locale("es", "ES"), new Locale("it", "IT") };

for (int i=0; i< locales.length; i++) {
String displayLanguage = locales[i].getDisplayLanguage(locales[i]);
System.out.println(locales[i].toString() + ": " + displayLanguage);
}
}

}

Output:en_US: English
es_ES: espa?ol
it_IT: italiano

```

## ResourceBundle class in Java

The **ResourceBundle class** is used to internationalize the messages. In other words, we can say that it provides a mechanism to globalize the messages.

The hardcoded message is not considered good in terms of programming, because it differs from one country to another. So we use the ResourceBundle class to globalize the messages. The ResourceBundle class loads these informations from the properties file that contains the messages.

Conventionally, the name of the properties file should be **filename\_languagecode\_country** code for example **MyMessage\_en\_US.properties**.

---

### Commonly used methods of ResourceBundle class

There are many methods in the ResourceBundle class. Let's see the commonly used methods of the ResourceBundle class.

- **public static ResourceBundle getBundle(String basename)** returns the instance of the ResourceBundle class for the default locale.
  - **public static ResourceBundle getBundle(String basename, Locale locale)** returns the instance of the ResourceBundle class for the specified locale.
  - **public String getString(String key)** returns the value for the corresponding key from this resource bundle.
- 

## Example of ResourceBundle class

Let's see the simple example of ResourceBundle class. In this example, we are creating three files:

- **MessageBundle\_en\_US.properties** file contains the localize message for US country.
- **MessageBundle\_in\_ID.properties** file contains the localize message for Indonaisa country.
- **InternationalizationDemo.java** file that loads these properties file in a bundle and prints the messages.

### MessageBundle\_en\_US.properties

```
greeting=Hello, how are you?
```

### MessageBundle\_in\_ID.properties

```
greeting=Halo, apa kabar?
```

## InternationalizationDemo.java

```
import java.util.Locale;
import java.util.ResourceBundle;
public class InternationalizationDemo {
    public static void main(String[] args) {

        ResourceBundle bundle = ResourceBundle.getBundle("MessageBundle", Locale.US);
        System.out.println("Message in "+Locale.US +":"+bundle.getString("greeting"));

        //changing the default locale to indonasan
        Locale.setDefault(new Locale("in", "ID"));
        bundle = ResourceBundle.getBundle("MessageBundle");
        System.out.println("Message in "+Locale.getDefault()+":"+bundle.getString("greeting"));

    }
}
```

```
Output:Message in en_US : Hello, how r u?
Message in in_ID : halo, apa kabar?
```

[download this example of ResourceBundle class](#)

## Internationalizing Date (I18N with Date)

The format of the dates differ from one region to another that is why we internationalize the dates.

We can internationalize the date by using the **getDateInstance()** method of the **DateFormat** class. It receives the locale object as a parameter and returns the instance of the DateFormat class.

### Commonly used methods of DateFormat class for internationalizing date

There are many methods of the DateFormat class. Let's see the two methods of the DateFormat class for internationalizing the dates.

- **public static DateFormat getDateInstance(int style, Locale locale)** returns the instance of the DateFormat class for the specified style and locale. The style can be DEFAULT, SHORT, LONG etc.
- **public String format(Date date)** returns the formatted and localized date as a string.

### Example of Internationalizing Date

In this example, we are displaying the date according to the different locale such as UK, US, FRANCE etc. For this purpose we have created the printDate() method that receives Locale object as an instance. The format() method of the DateFormat class receives the Date object and returns the formatted and localized date as a string.

```
import java.text.DateFormat;
import java.util.*;
public class InternationalizationDate {
```

```

static void printDate(Locale locale){
    DateFormat formatter=DateFormat.getDateInstance(DateFormat.DEFAULT,locale);
    Date currentDate=new Date();
    String date=formatter.format(currentDate);
    System.out.println(date+" "+locale);
}

public static void main(String[] args) {
    printDate(Locale.UK);
    printDate(Locale.US);
    printDate(Locale.FRANCE);
}
}

```

[download this example](#)

Output:01-Mar-2012 en\_GB  
 Mar 1, 2012 en\_US  
 1 mars 2012 fr\_FR

## Internationalizing Time (I18N with Time)

The display format of the time differs from one region to another, so we need to internationalize the time.

For internationalizing the time, the **DateFormat** class provides some useful methods.

The **getTimeInstance()** method of the DateFormat class returns the instance of the DateFormat class for the specified style and locale.

Syntax of the getTimeInstance() method is given below:

```
public static DateFormat getTimeInstance(int style, Locale locale)
```

---

### Example of Internationalizing Time

In this example, we are displaying the current time for the specified locale. The format() method of the DateFormat class receives date object and returns the formatted and localized time as a string. Notice that the object of Date class prints date and time both.

```

import java.text.DateFormat;
import java.util.*;

public class InternationalizingTime {

    static void printTime(Locale locale){
        DateFormat formatter=DateFormat.getTimeInstance(DateFormat.DEFAULT,locale);
        Date currentDate=new Date();
        String time=formatter.format(currentDate);
        System.out.println(time+" in locale "+locale);
    }

    public static void main(String[] args) {
        printTime(Locale.UK);
    }
}

```

```
printTime(Locale.US);
printTime(Locale.FRANCE);
}
}
```

[download this example](#)

```
Output:16:22:49 in locale en_GB
4:22:49 PM in locale en_US
16:22:49 in locale fr_FR
```

## Internationalizing Number (I18N with Number)

The representation of the numbers differ from one locale to another. Internationalizing the numbers is good approach for the application that displays the informations according to the locales.

The **NumberFormat** class is used to format the number according to the specific locale. To get the instance of the NumberFormat class, we need to call either **getInstance()** or **getNumberInstance()** methods.

Syntax of these methods is given below:

```
public static NumberFormat getNumberInstance(Locale locale)
public static NumberFormat getInstance(Locale locale)//same as above
```

## Internationalizing Currency (I18N with Currency)

As we have internationalize the date, time and numbers, we can internationalize the currency also. The currency differs from one country to another so we need to internationalize the currency.

The **NumberFormat** class provides methods to format the currency according to the locale. The **getCurrencyInstance()** method of the NumberFormat class returns the instance of the NumberFormat class.

The syntax of the **getCurrencyInstance()** method is given below:

```
public static NumberFormat getCurrencyInstance(Locale locale)
```