

Unit III

1. Explain JDBC
2. Explain the different types of drivers in JDBC
3. Explain the steps to connect to a database using JDBC
4. Explain the different types of statements in JDBC
5. What is a Naming Service? Explain
6. Explain a directory service.

Unit IV

1. Explain the J2EE architecture
1. Explain EJB architecture
2. Explain the different types of EJBs
3. Explain Session Beans with their lifecycle
4. Explain the difference between stateful and stateless session beans
5. Explain Entity Beans with their lifecycle
6. Explain Message Driven Beans

UNIT-I

INTRODUCTION:

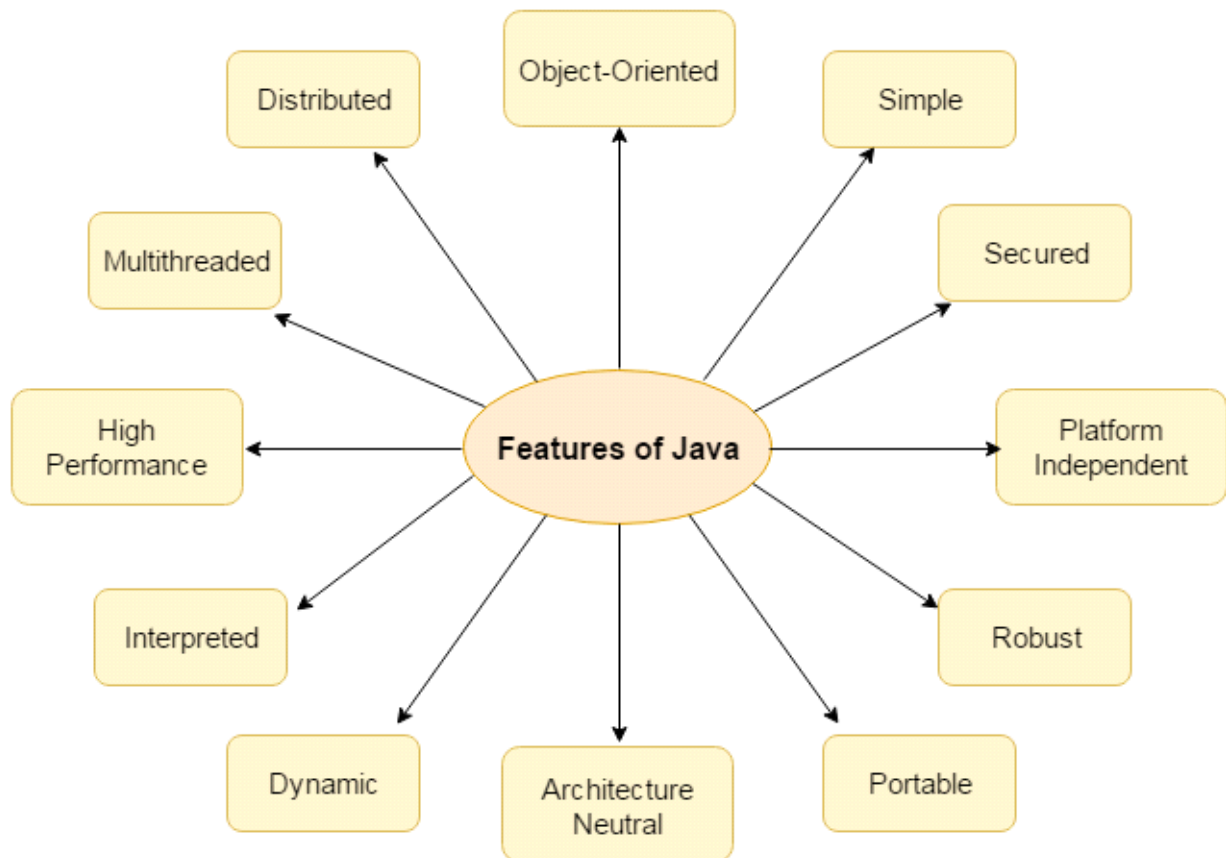
Java is a high-level, third generation programming language, like C, FORTRAN, Smalltalk, Perl, and many others. We can use Java to write computer applications that play games, store data or do any of the thousands of other things computer software can do. What's most special about Java in relation to other programming languages is that it lets you write special programs called applets that can be downloaded from the Internet and played safely within a web browser. Java language is called as an Object-Oriented Programming language.

HISTORY:

In 1990, Sun Microsystems began a project called Green to develop software for consumer electronics. Developer, Gosling began writing software in C++ for embedding into such items as toasters, VCR's, and Personal Digital Assistants (PDA's). The embedded software makes many appliances more intelligent. Gosling's solution to the problems of C++ was a new language called Oak. Finally in 1995, Oak was renamed Java. Since then Java has been rising in popularity.

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

- ✓ JDK Alpha and Beta (1995)
- ✓ JDK 1.0 (23rd Jan, 1996)
- ✓ JDK 1.1 (19th Feb, 1997)
- ✓ J2SE 1.2 (8th Dec, 1998)
- ✓ J2SE 1.3 (8th May, 2000)
- ✓ J2SE 1.4 (6th Feb, 2002)
- ✓ J2SE 5.0 (30th Sep, 2004)
- ✓ Java SE 6 (11th Dec, 2006)
- ✓ Java SE 7 (28th July, 2011)



FEATURES OF JAVA

There are many features in java , which are also known as java buzzwords. The Java Features are as follows:

- 1)Compiled and Interpreted
- 2)Platform Independent and portable
- 3)Object- oriented
- 4)Robust and secure
- 5)Distributed
- 6)Familiar, simple and small
- 7)Multithreaded and Interactive
- 8)High performance
- 9)Dynamic and Extensible

1)Compiled and Interpreted: Basically a computer language is either compiled or interpreted. Java comes together both these approach thus making Java a two-stage system.Java compiler translates Java code to Bytecode instructions and

Java Interpreter generate machine code that can be directly executed by machine that is running the Java program.

2) Platform Independent and portable: Java supports the feature portability. Java programs can be easily moved from one computer system to another and anywhere. Changes and upgrades in operating systems, processors and system resources will not force any alteration in Java programs. This is reason why Java has become a trendy language for programming on Internet which interconnects different kind of systems worldwide. Java certifies portability in two ways. First way is, Java compiler generates the bytecode and that can be executed on any machine. Second way is, size of primitive data types are machine independent.

3) Object- oriented: Java is truly object-oriented language. In Java, almost everything is an Object. All program code and data exist in objects and classes. Java comes with an extensive set of classes; organize in packages that can be used in program by Inheritance. The object model in Java is trouble-free and easy to enlarge.

- a. Object
- b. Class
- c. Data abstraction
- d. Data encapsulation
- e. Inheritance
- f. Polymorphism
- g. Dynamic binding

a) Object: Objects are important runtime entities in object oriented method. They may characterize a location, a bank account, and a table of data or any entry that the program

must handle. Each object holds data and code to operate the data. Object can interact without having to identify the details of each other's data or code. It is sufficient to identify the type of message received and the type of reply returned by the objects.

- b) **Classes:** A class is a set of objects with similar properties (attributes), common behaviour (operations), and common link to other objects. The complete set of data and code of an object can be made a user defined data type with the help of class. The objects are variable of type class. A class is a collection of objects of similar type. Classes are user defined data types and work like the build in type of the programming language. Once the class has been defined, we can make any number of objects belonging to that class. Each object is related with the data of type class with which they are formed. The classification of objects into various classes is based on its properties (States) and behaviour (methods). Classes are used to distinguish are type of object from another. The important thing about the class is to identify the properties and procedures and applicability to its instances.
- c) **Data Abstraction:** Data abstraction refers to the act of representing important description without including the background details or explanations. Classes use the concept of abstraction and are defined as a list of abstract attributes such as size, cost and functions operate on these attributes. They summarize all the important properties of the objects that are to be created. Classes use the concepts

of data abstraction and it is called as Abstract Data Type (ADT).

d) **Data Encapsulation:** Data Encapsulation means wrapping of data and functions into a single unit (i.e. class). It is most useful feature of class. The data is not easy to get to the outside world and only those functions which are enclosed in the class can access it. These functions provide the boundary between Object's data and program. This insulation of data from direct access by the program is called as **Data hiding**.

e) **Inheritance:** Inheritance is the process by which objects of one class can get the properties of objects of another class. It means one class of objects inherits the data and behaviours from another class. It maintains the hierarchical classification in which a class inherits from its parents. It provides the important feature of OOP that is reusability. That means we can include additional characteristics to an existing class without modification. In other words, it is property of object-oriented systems that allow objects to be built from other objects. It allows openly taking help of the commonality of objects when constructing new classes. It is a relationship between classes where one class is the parent class of another (derived) class. The derived class holds the properties and behaviour of base class in addition to the properties and behaviour of derived class.

f) **Polymorphism:** (Poly means “many” and morph means “form”). Polymorphism means the ability to take more than one form. It plays a main role in allocate objects

having different internal structures to share the same external interface. This means that a general class of operations may be accessed in the same manner even though specific activities associated with each operation may differ. It is broadly used in implementing inheritance. It means objects that can take on or assume many different forms. It means that the same operations may behave differently on different classes. It allows us to write generic, reusable code more easily, because we can specify general instructions and delegate the implementation detail to the objects involved.

g) **Dynamic Binding:** Binding refers to the linking of a procedure call to the code to be executed in response to the call. It means that the code related with a given procedure call is not known until the time of the call at run time. It is associated polymorphism and inheritance.

4) **Robust and secure:** Java is a most strong language which provides many securities to make certain reliable code. It is design as garbage – collected language, which helps the programmers virtually from all memory management problems. Java also includes the concept of exception handling, which detain serious errors and reduces all kind of threat of crashing the system. Security is an important feature of Java and this is the strong reason that programmer use this language for programming on Internet. The absence of pointers in Java ensures that programs cannot get right of entry to memory location without proper approval.

5) **Distributed:** Java is called as Distributed language for construct applications on networks which can contribute

both data and programs. Java applications can open and access remote objects on Internet easily. That means multiple programmers at multiple remote locations to work together on single task.

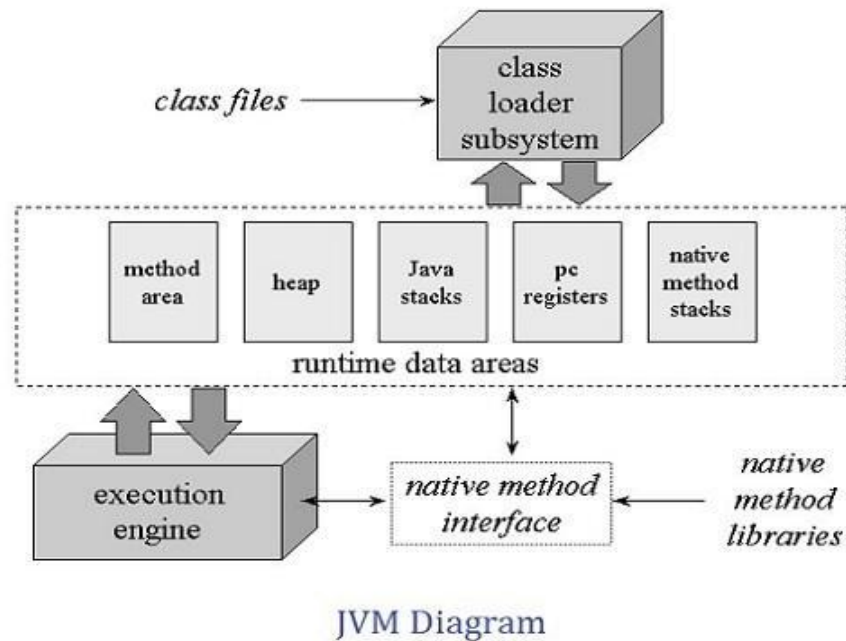
- 6) **Simple and small:** Java is very small and simple language. Java does not use pointer and header files, goto statements, etc. It eliminates operator overloading and multiple inheritance.
- 7) **Multithreaded and Interactive :** Multithreaded means managing multiple tasks simultaneously. Java maintains multithreaded programs. That means we need not wait for the application to complete one task before starting next task. This feature is helpful for graphic applications.
- 8) **High performance :** Java performance is very extraordinary for an interpreted language, majorly due to the use of intermediate bytecode. Java architecture is also designed to reduce overheads during runtime. The incorporation of multithreading improves the execution speed of program.
- 9) **Dynamic and Extensible:** Java is also dynamic language. Java is capable of dynamically linking in new class, libraries, methods and objects. Java can also establish the type of class through the query building it possible to either dynamically link or abort the program, depending on the reply. Java program support functions written in other language such as C and C++, known as native methods.

JVM (Java Virtual Machine)

What is JVM?

JVM is a virtual machine or a program that provides run-time environment in which java byte code can be executed. JVMs are available for many hardware and software platforms. The use of the same byte code for all JVMs on all platforms make java platform independent.

JVM diagram:



JVM details:

- 1) ***Class loader subsystem:*** It is a part of JVM that care of finding and loading of class files.
- 2) ***Class/method area:*** It is a part of JVM that contains the information of types/classes loaded by class loader. It also contains the static variable, method body etc.
- 3) ***Heap:*** It is a part of JVM that contains object. When a new object is created, memory is allocated to the object from the heap and object is no longer referenced memory is reclaimed by garbage collector.
- 4) ***Java Stack:*** It is a part of JVM that contains local variable, operands and frames. To perform an operation,

Byte code instructions takes operands from the stack, operate and then return the result in to the java stack.

5)**Program Counter:** For each thread JVM instance provide a separate program counter (PC) or pc register which contains the address of currently executed instruction.

6)**Native method stack:** As java program can call native methods (A method written in other language like c). Native method stack contains these native methods.

7)**Execution Engine:** It is a part JVM that uses Virtual processor (for execution), interpreter (for reading instructions) and JIT (Just in time) compiler (for performance improvement) to execute the instructions.

How JVM is created(Why JVM is virtual):

When JRE installed on your machine, you got all required code to create JVM. JVM is created when you run a java program, e.g. If you create a java program named FirstJavaProgram.java. To compile use – `javac FirstJavaProgram.java` and to execute use – `java FirstJavaProgram`. When you run second command – `java FirstJavaProgram`, JVM is created. That's why it is virtual.

Lifetime of JVM:

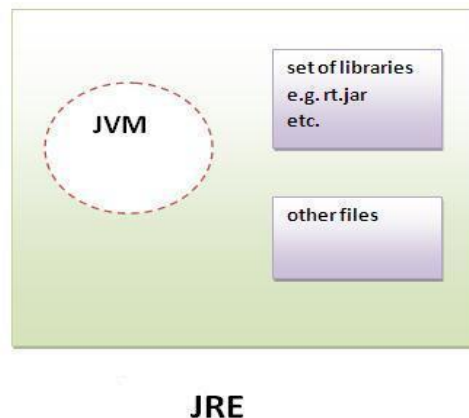
When an application starts, a runtime instance is created. When application ends, runtime environment destroyed. If 'n' no. of applications starts on one machine then n no. of runtime instances are created and every application run on its own JVM instance.

Main task of JVM:

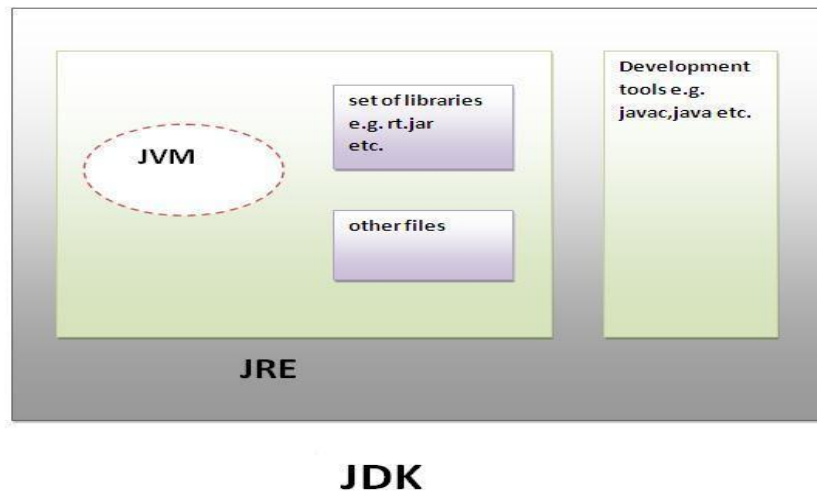
1. Search and locate the required files.
2. Convert byte code into executable code.
3. Allocate the memory into ram
4. Execute the code.
5. Delete the executable code.

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.



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



JAVA CODING GUIDELINES

1. **Basic Naming convention standards:** To describe variable/constant/method/class/interface etc use full descriptor. E.g.: rollNumber, firstName, lastName, getTotalMarks().
2. **Naming variable:** Use first word in small letters and all remaining words will be capitalized. E.g. – rollNumber, firstName.
3. **Naming Constants:** Use all letters in upper case. E.g. – MAX_MARKS.
4. **Naming methods:**
 - a). **Naming member methods :**

Use first word in small letters and all remaining words will be capitalized. E.g. – getTotalMarks().
 - b). **Naming accessor methods:**

for getters – use get as prefix to property (for non Boolean properties). e.g. – getRollNumber().

for setters – use set as prefix to property. e.g. – setRollNumber().

use is as prefix to property(for Boolean properties). e.g. – isNewStudent().

5. ***Naming class/interface:*** Use capitalized words for class/interface name. E.g.- HelloWorld.

6. ***Comment:*** For clarity of the code add comments

Java Versus C++

Java is a true object oriented language while C++ is basically C with an object oriented extension. Listed below are some major C++ features that were intentionally omitted from Java, or significantly modified:

- Java does not support operator overloading.
- Java does not have template classes.
- Java does not support multiple inheritance of classes.
- Java does not support global variables.
- Java does not use pointers.
- Java has replaced the destructor function with the finalize() function.
- There are no header files in Java.

Applications of Java

Java is being used in:

- Real-time Systems
- Simulation and Modelling
- object oriented Databases
- Artificial Intelligence and Expert Systems
- CIM/CAD/CAM Systems
- Neural Networks and Parallel Programming
- Decision Support Systems

OPERANDS , OPERATOR& EXPRESSION

Operand : Arguments on which operation is performed are known as operand. There are two type of operands used in the expression. They are as follows:

- **Constant:** It is an operand whose value can't be changed during the program execution.
- **Variable:** It is an operand whose value can be changed during the program execution.

Example: $A + 3$. In this expression **A** is a variable and **3** is a constant as an operands

Operator: Operator is a special symbol used for performing a specific task.

Example: $A + B$. Here $+$ symbol represents the operator.

Expression: A sequence of operands connected by operators is known as expression. e.g. $A + B * 5$.

Java provides a rich set of operators to manipulate variables. We can divide all the Java operators into the following groups:

- Arithmetic Operators
- Relational Operators
- Bitwise Operators
- Logical Operators
- Assignment Operators
- Misc Operators

The Arithmetic Operators: Arithmetic operators are used in mathematical expressions in the same way that they are used in algebra. The following table lists the arithmetic operators:

Assume integer variable A holds 10 and variable B holds 20, then:

SR.NO	Operator and Example
1	+ (Addition) Adds values on either side of the operator Example: A + B will give 30
2	- (Subtraction)Subtracts right hand operand from left hand operand Example: A - B will give -10
3	* (Multiplication) Multiplies values on either side of the operator Example: A * B will give 200
4	/ (Division) Divides left hand operand by right hand operand Example: B / A will give 2
5	% (Modulus) Divides left hand operand by right hand operand and returns remainder Example: B % A will give 0
6	++ (Increment) Increases the value of operand by 1 Example: B++ gives 21
7	-- (Decrement) Decreases the value of operand by 1 Example: B-- gives 19

The Relational Operators: There are following relational operators supported by Java language

Assume variable A holds 10 and variable B holds 20, then:

SR.NO	Operator and Description
1	== (equal to)

	<p>Checks if the values of two operands are equal or not, if yes then condition becomes true.</p> <p>Example: (A == B) is not true.</p>
2	<p>!= (not equal to)</p> <p>Checks if the values of two operands are equal or not, if values are not equal then condition becomes true.</p> <p>Example: (A != B) is true.</p>
3	<p>> (greater than)</p> <p>Checks if the value of left operand is greater than the value of right operand, if yes then condition becomes true.</p> <p>Example: (A > B) is not true.</p>
4	<p>< (less than)</p> <p>Checks if the value of left operand is less than the value of right operand, if yes then condition becomes true.</p> <p>Example: (A < B) is true.</p>
5	<p>>= (greater than or equal to)</p> <p>Checks if the value of left operand is greater than or equal to the value of right operand, if yes then condition becomes true.</p> <p>Example (A >= B) is not true.</p>
6	<p><= (less than or equal to)</p> <p>Checks if the value of left operand is less than or equal to the value of right operand, if yes then condition becomes true.</p> <p>example(A <= B) is true.</p>

The Bitwise Operators: Java defines several bitwise operators, which can be applied to the integer types, long, int, short, char, and byte. Bitwise operator works on bits and performs bit-by-bit operation.

Assume if $a = 60$; and $b = 13$; now in binary format they will be as follows:

```
a = 0011 1100
b = 0000 1101
-----
a&b = 0000 1100
a|b = 0011 1101
a^b = 0011 0001
~a  = 1100 0011
```

The following table lists the bitwise operators:

Assume integer variable A holds 60 and variable B holds 13 then:

SR.NO	Operator and Description
1	& (bitwise and) Binary AND Operator copies a bit to the result if it exists in both operands. Example: (A & B) will give 12 which is 0000 1100
2	 (bitwise or) Binary OR Operator copies a bit if it exists in either operand. Example: (A B) will give 61 which is 0011 1101
3	^ (bitwise XOR) Binary XOR Operator copies the bit if it is set in one operand but not both.

	Example: (A ^ B) will give 49 which is 0011 0001
4	~ (bitwise compliment) Binary Ones Complement Operator is unary and has the effect of 'flipping' bits. Example: (~A) will give -61 which is 1100 0011 in 2's complement form due to a signed binary number.
5	<< (left shift) Binary Left Shift Operator. The left operands value is moved left by the number of bits specified by the right operand Example: A << 2 will give 240 which is 1111 0000
6	>> (right shift) Binary Right Shift Operator. The left operands value is moved right by the number of bits specified by the right operand. Example: A >> 2 will give 15 which is 1111
7	>>> (zero fill right shift) Shift right zero fill operator. The left operands value is moved right by the number of bits specified by the right operand and shifted values are filled up with zeros. Example: A >>>2 will give 15 which is 0000 1111

The Logical Operators: **The following table lists the logical operators:**

Assume Boolean variables A holds true and variable B holds false, then:

Operator	Description
1	&& (logical and) Called Logical AND operator. If both the operands are non-zero, then the condition becomes true. Example (A && B) is false.
2	 (logical or) Called Logical OR Operator. If any of the two operands are non-zero, then the condition becomes true. Example (A B) is true.
3	! (logical not) Called Logical NOT Operator. Use to reverses the logical state of its operand. If a condition is true then Logical NOT operator will make false. Example !(A && B) is true.

The Assignment Operators : There are following assignment operators supported by Java language:

SR.NO	Operator and Description
1	= Simple assignment operator, Assigns values from right side operands to left side operand. Example: C = A + B will assign value of A + B into C
2	+=

	<p>Add AND assignment operator, It adds right operand to the left operand and assign the result to left operand.</p> <p>Example: $C += A$ is equivalent to $C = C + A$</p>
3	<p>-=</p> <p>Subtract AND assignment operator, It subtracts right operand from the left operand and assign the result to left operand.</p> <p>Example: $C -= A$ is equivalent to $C = C - A$</p>
4	<p>*=</p> <p>Multiply AND assignment operator, It multiplies right operand with the left operand and assign the result to left operand.</p> <p>Example: $C *= A$ is equivalent to $C = C * A$</p>
5	<p>/=</p> <p>Divide AND assignment operator, It divides left operand with the right operand and assign the result to left operand</p> <p>Example: $C /= A$ is equivalent to $C = C / A$</p>
6	<p>%=</p> <p>Modulus AND assignment operator, It takes modulus using two operands and assign the result to left operand.</p> <p>Example: $C \% = A$ is equivalent to $C = C \% A$</p>
7	<p><<=</p> <p>Left shift AND assignment operator.</p> <p>Example: $C <<= 2$ is same as $C = C << 2$</p>
8	<p>>>=</p>

	Right shift AND assignment operator Example $C \gg= 2$ is same as $C = C \gg 2$
9	&= Bitwise AND assignment operator. Example: $C \&= 2$ is same as $C = C \& 2$
10	^= bitwise exclusive OR and assignment operator. Example: $C \wedge= 2$ is same as $C = C \wedge 2$
11	 = bitwise inclusive OR and assignment operator. Example: $C = 2$ is same as $C = C 2$

Miscellaneous Operators: There are few other operators supported by Java Language.

Conditional Operator (? :) : Conditional operator is also known as the ternary operator. This operator consists of three operands and is used to evaluate Boolean expressions. The goal of the operator is to decide which value should be assigned to the variable. The operator is written as:

variable x = (expression) ? value if true : value if false

Following is the example:

```
public class Test {

    public static void main(String args[]){
        int a, b;
        a = 10;
        b = (a == 1) ? 20: 30;
        System.out.println( "Value of b is : " + b );
    }
}
```

```

    b = (a == 10) ? 20: 30;
    System.out.println( "Value of b is : " + b );
}
}

```

This would produce the following result –

```

Value of b is : 30
Value of b is : 20

```

instance of Operator:

This operator is used only for object reference variables. The operator checks whether the object is of a particular type (class type or interface type). instanceof operator is written as:

(Object reference variable) instanceof (class/interface type)

If the object referred by the variable on the left side of the operator passes the IS-A check for the class/interface type on the right side, then the result will be true. Following is the example:

```

public class Test {

    public static void main(String args[]){
        String name = "James";
        // following will return true since name is type of String
        boolean result = name instanceof String;
        System.out.println( result );
    }
}

```

This would produce the following result:

```
true
```

This operator will still return true if the object being compared is the assignment compatible with the type on the right. Following is one more example:

```
class Vehicle {}
```

```
public class Car extends Vehicle {  
    public static void main(String args[]){  
        Vehicle a = new Car();  
        boolean result = a instanceof Car;  
        System.out.println( result );  
    }  
}
```

This would produce the following result:

```
true
```

PRECEDENCE OF JAVA OPERATORS:

Operator precedence determines the grouping of terms in an expression. This affects how an expression is evaluated. Certain operators have higher precedence than others; for example, the multiplication operator has higher precedence than the addition operator:

For example, $x = 7 + 3 * 2$; here x is assigned 13, not 20 because operator $*$ has higher precedence than $+$, so it first gets multiplied with $3*2$ and then adds into 7.

Here, operators with the highest precedence appear at the top of the table, those with the lowest appear at the bottom. Within an expression, higher precedence operators will be evaluated first.

Category	Operator	Associativity
Postfix	() [] . (dot operator)	Left to right

Unary	++ - - ! ~	Right to left
Multiplicative	* / %	Left to right
Additive	+ -	Left to right
Shift	>> >>> <<	Left to right
Relational	> >= < <=	Left to right
Equality	== !=	Left to right
Bitwise AND	&	Left to right
Bitwise XOR	^	Left to right
Bitwise OR		Left to right
Logical AND	&&	Left to right
Logical OR		Left to right
Conditional	?:	Right to left
Assignment	= += -= *= /= %= >>= <<= &= ^= =	Right to left

VARIABLE AND DATATYPES IN JAVA

Variable: Variable is the name of reserved memory location. It means when we declare a variable some part of memory is reserved.

Variable Declaration: Identifiers are the names of variables. They must be composed of only letters, numbers, the underscore, and the dollar sign (\$). They cannot contain white spaces. Identifiers may only begin with a letter, the underscore, or the dollar sign. A variable cannot begin with a number. All variable names are case sensitive.

Syntax for variable declaration:

*datatype1 variable1, datatype2 variable2, ... datatypen
variablen;*

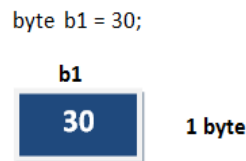
Example:

```
int a, char ch;
```

Rules to declare a Variable:

- Every variable name should start with either alphabets or underscore (_) or dollar (\$) symbol.
- No space are allowed in the variable declarations.
- Except underscore (_) no special symbol are allowed in the middle of variable declaration
- Variable name always should exist in the left hand side of assignment operators.
- Maximum length of variable is 64 characters.
- No keywords should access variable name.

Variable initialization: It is the process of storing user defined values at the time of allocation of memory space.



Variable assignment: Value is assigned to a variable if that is already declared or initialized.

Syntax: Variable_Name = value

Example: int a = 100;



Example:

```
int a= 100;
```

```
int b;
```

```
b = 25; // -----> direct assigned variable
```

```
b = a;  // -----> assigned value in term of variable
```

```
b = a+15; // -----> assigned value as term of expression
```

Variable Types: There are three types of variable used in java. They are as follows:

- ***Local variable:*** A variable that is declared within the method, constructor, or block is known as local variable. No Access modifier is used for local variables. Scope of local variable is limited to that method, constructor, or block only in which it is declared.
- ***Instance variable:*** A variable that is declared within the class but outside of method, constructor, or block is known as instance variable (Non static). They are associated with object. Access modifiers can be used with instance variables. Inside class we can access instance variable directly by variable-name without any object reference.
- ***Static variable:*** A variable that is declared within the class with static keyword but outside of method, constructor, or block is known as Static/class variable. They are associated with class. Static variable are accessed by *ClassName.VariableName*.

Data types: When a variable is declared some part of memory is reserved. But how much memory will be reserved. It depends upon the data type of the variable. i.e. how much memory will be reserved and which type of data can be stored in reserved memory is depends upon data type of the variable.

DATA TYPES IN JAVA

Based on the data type of a variable, the operating system allocates memory and decides what can be stored in the reserved memory. Therefore, by assigning different data types to

variables, you can store integers, decimals, or characters in these variables.

There are two data types available in Java:

- Primitive Data Types
- Reference/Object Data Types

Primitive Data Types: There are eight primitive data types supported by Java. Primitive data types are predefined by the language and named by a keyword. Let us now look into detail about the eight primitive data types.

byte:

- Byte data type is an 8-bit signed two's complement integer.
- Minimum value is -128 (-2^7)
- Maximum value is 127 (inclusive) ($2^7 - 1$)
- Default value is 0
- Byte data type is used to save space in large arrays, mainly in place of integers, since a byte is four times smaller than an int.
- Example: byte a = 100 , byte b = -50

short:

- Short data type is a 16-bit signed two's complement integer.
- Minimum value is -32,768 (-2^{15})
- Maximum value is 32,767 (inclusive) ($2^{15} - 1$)
- Short data type can also be used to save memory as byte data type. A short is 2 times smaller than an int
- Default value is 0.
- Example: short s = 10000, short r = -20000

int:

- Int data type is a 32-bit signed two's complement integer.
- Minimum value is - 2,147,483,648. (-2^{31})

- Maximum value is 2,147,483,647(inclusive).($2^{31} - 1$)
- Int is generally used as the default data type for integral values unless there is a concern about memory.
- The default value is 0.
- Example: int a = 100000, int b = -200000

long:

- Long data type is a 64-bit signed two's complement integer.
- Minimum value is -9,223,372,036,854,775,808.(-2^{63})
- Maximum value is 9,223,372,036,854,775,807 (inclusive). ($2^{63} - 1$)
- This type is used when a wider range than int is needed.
- Default value is 0L.
- Example: long a = 100000L, long b = -200000L

float:

- Float data type is a single-precision 32-bit IEEE 754 floating point.
- Float is mainly used to save memory in large arrays of floating point numbers.
- Default value is 0.0f.
- Float data type is never used for precise values such as currency.
- Example: float f1 = 234.5f

double:

- double data type is a double-precision 64-bit IEEE 754 floating point.
- This data type is generally used as the default data type for decimal values, generally the default choice.
- Double data type should never be used for precise values such as currency.
- Default value is 0.0d.
- Example: double d1 = 123.4

boolean:

- boolean data type represents one bit of information.
- There are only two possible values: true and false.
- This data type is used for simple flags that track true/false conditions.
- Default value is false.
- Example: `boolean one = true`

char:

- char data type is a single 16-bit Unicode character.
- Minimum value is `'\u0000'` (or 0).
- Maximum value is `'\uffff'` (or 65,535 inclusive).
- Char data type is used to store any character.
- Example: `char letter = 'A'`

Reference Data Types:

- Reference variables are created using defined constructors of the classes. They are used to access objects. These variables are declared to be of a specific type that cannot be changed. For example, Employee, Puppy etc.
- Class objects, and various type of array variables come under reference data type.
- Default value of any reference variable is null.
- A reference variable can be used to refer to any object of the declared type or any compatible type.
- Example: `Student std = new Student(" Smith");`

The below table shows the default size and value of the data types

Data Type	Default Value	Default Size
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
char	'\u0000'	2 byte
boolean	false	1 bit

2. Referenced or object data types: Any class object or array type. Default value of object data types is null.

Note:

1. Compiler never initialize local variable with default values. So you have to initialize local variable before using otherwise it will result in compile-time error.
2. Char in java takes 2 bytes because java uses Unicode system (UTF-16) which needed a minimum of 2 bytes memory to store a character.

JAVA TOKENS

A token is the smallest element in a program that is meaningful to the compiler. These tokens define the structure of the language. The Java token set can be divided into five categories: Identifiers, Keywords, Literals.

1. **Identifiers:** Identifiers are names provided by you. These can be assigned to variables, methods, functions, classes etc. to uniquely identify them to the compiler.
2. **Keywords:** Keywords are reserved words that have a specific meaning for the compiler. They cannot be used as identifiers. Java has a rich set of keywords. Some examples are: boolean, char, if, protected, new, this, try, catch, null, thread safe etc.

List of Java reserved words

abstract	do	instance	this	case	final	super	new
assert	double	of	throw	catch	finall	switch	null
boolean	else	int	throws	char	y	synchroniz	packag
break	enum	private	transie	class	float	ed	e
byte	extend	protecte	nt	const	for	volatile	native
if	s	d	true	defau	goto	while	
implemen	continup	public	try	lt	static	interface	
ts	e	return	void	false	strictf	long	
	import	short			p		

- a. **Literals:** Literals are variables whose values remain constant throughout the program. They are also called Constants. Literals can be of four types. They are:
String Literals: String Literals are always enclosed in double quotes and are implemented using the **java.lang.String** class. Enclosing a character string within double quotes will automatically create a new String object. For example, `String s = "this is a string";`. String objects are immutable, which means that once created, their values cannot be changed.
- b. **Character Literals:** These are enclosed in single quotes and contain only one character.
- c. **Boolean Literals:** They can only have the values `true` or `false`. These values do not correspond to 1 or 0 as in C or C++.
- d. **Numeric Literals:** Numeric Literals can contain integer or floating point values.

STRUCTURE OF JAVA PROGRAM

Structure of a java program is the standard format released by Language developer to the Industry programmer. Sun Micro System has prescribed the following structure for the java programmers for developing java application.

package details	→	import java.io.*
class className	→	class Sum
{		
Data members;	→	int a, b, c;
user_defined method;	→	void display();
public static void main(String args[])		
{		
Block of Statements;	→	System.out.println("Hello Java !");
}		
}		

Structure Of Java Program

- A **package** is a collection of classes, interfaces and sub-packages. A sub package contains collection of classes, interfaces and sub-sub packages etc. java.lang.*; package is imported by default and this package is known as default package.
- **Class** is keyword used for developing user defined data type and every java program must start with a concept of class.
- "**ClassName**" represent a java valid variable name treated as a name of the class each and every class name in java is treated as user-defined data type.
- **Data member** represents either instance or static they will be selected based on the name of the class.

- ***User-defined methods*** represents either instance or static they are meant for performing the operations either once or each and every time.
- ***main()*** - Each and every java program starts execution from the main() method. And hence main() method is known as program driver.
 - Since main() method of java is not returning any value and hence its return type must be void.
 - Since main() method of java executes only once throughout the java program execution and hence its nature must be static.
 - Since main() method must be accessed by every java programmer and hence whose access specifier must be public.
 - Each and every main() method of java must take array of objects of String.
- ***Block of statements*** represents set of executable statements which are in term calling user-defined methods are containing business-logic.
- The file naming convention in the java programming is that which-ever class is containing main() method, that class name must be given as a file name with an extension .java.

First Java Program

Requirements for java Program

For executing any java program we need given things.

- Install the JDK any version if you don't have installed it.
- Set path of the jdk/bin directory.
- Create the java program

- Compile and run the java program

Steps For compiling and executing the java program

Java is very simple programming language first we write a java program and save it with program class name.

In below program we create a java program with "First" name so we save this program with "First.java" file name. We can save our java program anywhere in our system or computer.

Create First program

Example

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

Compile and Execute Java Code

To compile: `javac First.java`

To execute: `java First`

Output

Hello Java

My First Java Program

Save Java Program

Syntax: *Filename.java*

Example: First.java

Compile Java Program

Syntax: *Javac Filename.java*

Example: Javac First.java

Note:

Here **Javac** and **Java** are called tools or application programs or exe files developed by sun micro system and supply as a part of jdk 1.5/1.6/1.7 in bin folder. **Javac** is used for compile the java program and **java** is used for run the java program.

During the program execution internally following steps will be occurs.

- Class loader subsystem loads or transfer the specified class into main memory(RAM) from secondary memory(hard disk)
- JVM takes the loaded class
- JVM looks for main method because each and every java program start executing from main() method.
- Since main() method of java is static in nature, JVM call the main() method with respect to loaded class (Example: First as First.main(--))

Note: A java program can contain any number of main method but JVM start execution from that main() method which is taking array of object of String class.

Main() Method in Java

main() method is starting execution block of a java program or any java program start their execution from main method. If any class contain main() method known as main class.

Syntax of main() method:

```
public static void main(String args[])  
{  
// Body of the main() method  
}
```

1. ***public*** : public is a keyword in a java language whenever if it is preceded by main() method the scope is available anywhere in the java environment that means main() method can be executed from anywhere. main() method must be accessed by every java programmer and hence whose access specifier must be public.
2. ***static***: static is a keyword in java if it is preceded by any class properties for that memory is allocated only once in the program. Static method are executed only once in the program. main() method of java executes only once throughout the java program execution and hence it declare must be static.
3. ***void***: void is a special datatype also known as no return type, whenever it is preceded by main() method that will be never return any value to the operating system. main() method of java is not returning any value and hence its return type must be void.
4. ***String args[]***: String args[] is a String array used to hold command line arguments in the form of String values.

In case of main() method following changes are acceptable

1. We can declare String[] in any valid form.

String[] args

String args[]

String []args

2. Instance of String[] we can take var-arg String parameter is String...

Syntax

main(String[] args) --> main(String... args)

3. We can change the order of modifiers i.e Instead of

Syntax

public static we can take static public

4. Instead of args we can take any valid java identifier.

Syntax: public static void main(String a[])

We can overload main() method. A Java class can have any number of main() methods. But run the java program, which class should have main() method with signature as "public static void main(String[] args)". If you do any modification to this signature, compilation will be successful. But, not run the java program. we will get the run time error as main method not found.

Example of override main() method

Example

```
public class mainclass
```

```
{  
    public static void main(String[] args)  
    {  
        System.out.println("Execution starts from Main()");  
    }  
    void main(int args)  
    {  
        System.out.println("Override main()");  
    }  
    double main(int i, double d)  
    {  
        System.out.println("Override main()");  
        return d;  
    }  
}
```

Output

Execution starts from Main()

COMMAND LINE ARGUMENTS

If any input value is passed through command prompt at the time of running of program is known as **command line argument** by default every command line argument will be treated as string value and those are stored in string array of main() method.

Syntax for Compile and Run CMD programs

Compile By -> Javac Mainclass.java

Run By -> Java Mainclass value1 value2 value3

```
javac Mainclass.java
```

```
java Mainclass value1 value2 value3.....
```



Command Line Arguments

Example of command-line argument in java

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

Compile and Run above programs

Compile By > Javac CommandLineExample.java

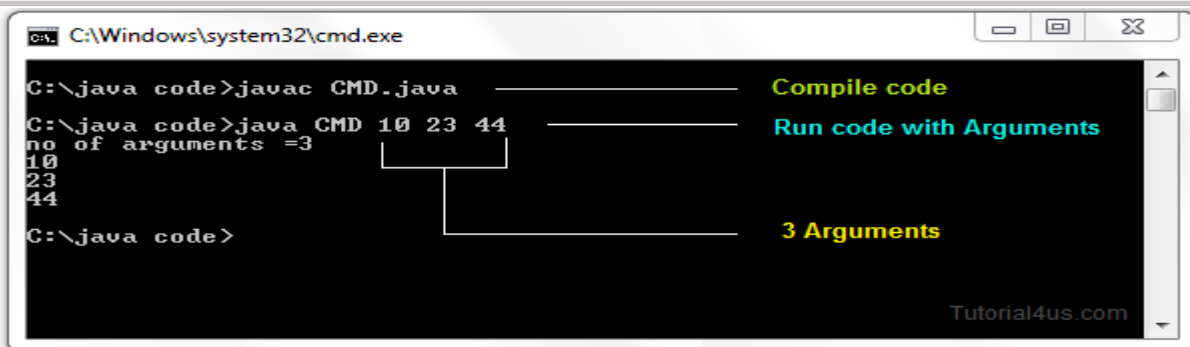
Run By > Java CommandLineExample Porter

Output

Argument is: Porter

Accept command line arguments and display their values

```
class CMD
{
    public static void main(String k[])
    {
        System.out.println("no. of arguments =" + k.length);
        for(int i=0; i < k.length; i++)
        {
            System.out.println(k[i]);
        }
    }
}
```



```
C:\Windows\system32\cmd.exe
C:\java code>javac CMD.java
C:\java code>java CMD 10 23 44
no of arguments =3
10
23
44
C:\java code>
```

Compile code
Run code with Arguments
3 Arguments

Tutorial4us.com

Note: Except + operator any numeric operation not allowed in command line arguments.

Square of Number by reading value from command prompt.

```
class squareDemo
{
    int no, result;
    void square(String s)
    {
```



```
int no=Integer.parseInt(s);
result=no*no;
System.out.println("Square: " +result);
}
}
class CMD
{
public static void main(String args[])
{
System.out.println("no of arguments: "+args.length);
squareDemo obj=new squareDemo();
obj.square(args[0]);
}
}
```

DECISION MAKING STATEMENT

Decision making statement statements is also called selection statement. That is depending on the condition block need to be executed or not which is decided by condition. If the condition is "true" statement block will be executed, if condition is "false" then statement block will not be executed. In java there are three types of decision making statement.

- if

- if-else
- switch

if-then Statement : if-then is most basic statement of Decision making statement. It tells to program to execute a certain part of code only if particular condition is true.

Syntax

```
if(condition)
{
    Statement(s)
}
```

Example if statement

```
class Hello
{
    int a=10;
    public static void main(String[] args)
    {
        if(a<15)
        {
            System.out.println("Hello good morning!");
        }
    }
}
```

Output

Hello good morning

if-else statement: In general it can be used to execute one block of statement among two blocks, in java language **if** and **else** are the keyword in java.

Syntax

```
if(condition)
{
    Statement(s)
}
else
{
    Statement(s)
}
.....
```

In the above syntax whenever condition is true all the if block statement are executed, remaining statement of the program by neglecting. If the condition is false else block statement executed and neglecting if block statements.

Example if else

```
import java.util.Scanner;
class Oddeven
{
    public static void main(String[] args)
    {
        int no;
        Scanner s=new Scanner(System.in);
        System.out.println("Enter any number :");
        no=s.nextInt();
        if(no%2==0)
        {
            System.out.println("Even number");
        }
        else
        {
            System.out.println("Odd number");
        }
    }
}
```

```
}  
}  
}
```

Output

```
Enter any number :  
10  
Even number
```

Switch Statement : The **switch** statement in java language is used to execute the code from multiple conditions or case. It is same like if else-if ladder statement. A switch statement work with byte, short, char and int primitive data type, it also works with enumerated types and string.

Syntax

```
switch(expression/variable)  
{  
  case value:  
    //statements  
    // any number of case statements  
    break; //optional  
    default: //optional  
    //statements  
}
```

Rules for apply switch statement

With switch statement use only byte, short, int, char data type (float data type is not allowed). You can use any number of case

statements within a switch. Value for a case must be same as the variable in switch.

Limitations of switch statement

Logical operators cannot be used with switch statement. For instance

Example

```
case k>=20: // not allowed
```

Example of switch case

```
import java.util.*;
class switchCase
{
    public static void main(String arg[])
    {
        int ch;
        System.out.println("Enter any number (1 to 7) :");
        Scanner s=new Scanner(System.in);
        ch=s.nextInt();
        switch(ch)
        {
            case 1:
                System.out.println("Today is Monday");
                break;
            case 2:
                System.out.println("Today is Tuesday");
                break;
            case 3:
                System.out.println("Today is Wednesday");
                break;
            case 4:
                System.out.println("Today is Thursday");
```

```
break;
case 5:
System.out.println("Today is Friday");
break;
case 6:
System.out.println("Today is Saturday");
break;
case 7:
System.out.println("Today is Sunday");
default:
System.out.println("Only enter value 1 to 7");
}
}
}
```

Output

```
Enter any number (1 to 7) :
5
Today is Friday
```

LOOPING STATEMENT

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

Why use loop ?

When you need to execute a block of code several number of times then you need to use looping concept in Java language.

Advantage with looping statement

- Reduce length of Code
- Take less memory space.
- Burden on the developer is reducing.
- Time consuming process to execute the program is reduced.

Difference between conditional and looping statement

Conditional statement executes only once in the program where as looping statements executes repeatedly several number of time.

While loop: In **while loop** first check the condition if condition is true then control goes inside the loop body otherwise goes outside of the body. while loop will be repeats in clock wise direction.

Syntax

```
while(condition)
{
  Statement(s)
  Increment / decrements (++ or --);}
```

Example while loop

```
class whileDemo
{
  public static void main(String args[])
  {
    int i=0;
    while(i<5)
    {
      System.out.println(+i);
      i++;
    }
  }
}
```

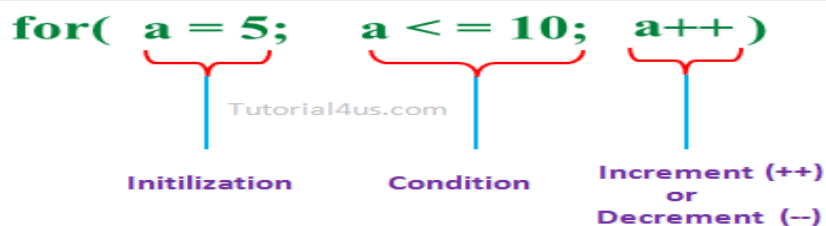
Output

1
2
3
4
5

for loop: **for loop** is a statement which allows code to be repeatedly executed. For loop contains 3 parts Initialization, Condition and Increment or Decrements

Syntax

```
for ( initialization; condition; increment )  
{  
    statement(s);  
}
```



The diagram illustrates the three components of a for loop: **for(a = 5; a <= 10; a++)**. Brackets below the code identify each part: **a = 5;** is labeled **Initilization** (note the typo), **a <= 10;** is labeled **Condition**, and **a++** is labeled **Increment (++) or Decrement (-)** (note the typo). The website **Tutorial4us.com** is visible in the background.

Initialization: This step is execute first and this is execute only once when we are entering into the loop first time. This step is allow to declare and initialize any loop control variables.

Condition: This is next step after initialization step, if it is true, the body of the loop is executed, if it is false then the body of the loop does not execute and flow of control goes outside of the for loop.

Increment or Decrements: After completion of Initialization and Condition steps loop body code is executed and then

Increment or Decrements steps is execute. This statement allows to update any loop control variables.

```

    ①      ②      ④
    for ( a = 1;    a <= 5;    a ++ )
    {
        System.out.println("Hello World") ③
    }
    Tutorial4us.com
```

- First initialize the variable
- In second step check condition
- In third step control goes inside loop body and execute.
- At last increase the value of variable
- Same process is repeat until condition not false.

Display any message exactly 5 times.

Example of for loop

```
class Hello
{
    public static void main(String args[])
    {
        int i;
        for (i=0; i<5; i++)
        {
            System.out.println("Hello Friends !");
        }
    }
}
```

Output

```
Hello Friends !
Hello Friends !
Hello Friends !
```

Hello Friends !

Hello Friends !

do-while: A **do-while** loop is similar to a while loop, except that a do-while loop is execute at least one time. A do while loop is a control flow statement that executes a block of code at least once, and then repeatedly executes the block, or not, depending on a given condition at the end of the block (in while).

When use do..while loop

when we need to repeat the statement block **at least one time** then ues do-while loop. In do-while loop post-checking process will be occur, that is after execution of the statement block condition part will be executed.

Syntax

```
do
{
Statement(s)
increment/decrement (++ or --)
}while();
```

In below example you can see in this program i=20 and we check condition i is less than 10, that means condition is false but do..while loop execute once and print Hello world ! at one time.

Example do..while loop

```
class dowhileDemo
{
public static void main(String args[])
{
int i=20;
do
```

```
{  
System.out.println("Hello world !");  
i++;  
}  
while(i<10);  
}  
}
```

Output

Hello world !

Example do..while loop

```
class dowhileDemo  
{  
public static void main(String args[])  
{  
int i=0;  
do  
{  
System.out.println(+i);  
i++;  
}  
while(i<5);  
}  
}
```

Output

1
2
3
4

Special Control Statements

Return Statements: The return statement is used in the definition of a method to set its returned value and to terminate execution of the method. It has two forms. Methods with returned type void use the following form.

return;

Methods with non-void returned type use the following form.

return expression;

Here, *expression* is an expression that yields the desired return value. This value must be convertible to the return type declared for the method.

Continue Statements : The continue statement is used in while loop, for loop, or do-while loop to terminate an iteration of the loop. A continue statement has the following form.

continue;

After a continue statement is executed in a for loop, its increment and boolean expression are evaluated. If the boolean expression is true then the nested statements are executed again.

After a continue statement is executed in a while or do-while loop, its boolean expression is evaluated. If the boolean expression is true then the nested statements are executed again.

Break Statements: The break statement is used in loop (for, while, and do-while) statements and switch statements to terminate execution of the statement. A break statement has the following form.

break;

After a break statement is executed, execution proceeds to the statement that follows the enclosing loop or switch statement.

WRAPPER CLASSES IN JAVA

For each and every fundamental data type there exist a pre-defined class, Such predefined class is known as wrapper class. The purpose of wrapper class is to convert numeric string data into numerical or fundamental data.

Why use wrapper classes ?

We know that in java whenever we get input from user, it is in the form of string value so here we need to convert these string values in different different datatype (numerical or fundamental data), for this conversion we use wrapper classes.

Example of wrapper class

```
class WrapperDemo
{
    public static void main(String[] args)
    {
        String s[] = {"10", "20"};
        System.out.println("Sum before:" + s[0] + s[1]); // 1020
        int x=Integer.parseInt(s[0]); // convert String to Integer
        int y=Integer.parseInt(s[1]); // convert String to Integer
        int z=x+y;
        System.out.println("sum after: " + z); // 30
    }
}
```

```
}  
}
```

Output

Sum before: 1020

Sum after: 30

Explanation: In the above example 10 and 20 are store in String array and next we convert these values in Integer using "int x=Integer.parseInt(s[0]);" and "int y=Integer.parseInt(s[1]);" statement
In "System.out.println("Sum before:"+ s[0] + s[1]);"
Statement normally add two string and output is 1020 because these are String numeric type not number.

```
String s = " 10.6f ";  
  
float x = Float.parseFloat(s);  
      ↓      ↓      ↓  
data  wrapper  method  
type  class    name  
  
System.out.println(x); // 10.6
```

Converting String data into fundamental or numerical

We know that every command line argument of java program is available in the main() method in the form of array of object of string class on String data, one can not perform numerical operation. To perform numerical operation it is highly desirable to convert numeric String into fundamental numeric value.

Example

"10" --> numeric String --> only numeric string convert into numeric type or value.

"10x" --> alpha-numeric type --> this is not conversion.

"ABC" --> non-numeric String no conversion.

"A" --> char String no conversion.

Only 'A' is convert into ASCII value that is 65 but 'A' is not convert into numeric value because it is a String value.

Fundamental data type and corresponding wrapper classes

The following table gives fundamental data type corresponding wrapper class name and conversion method from numerical String into numerical values or fundamental value.

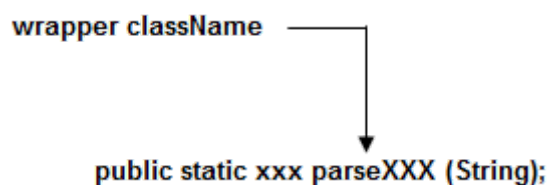
Fundamental Data Type	Wrapper CalssName	Conversion method from numeric string into fundamental or numeric value
byte	Byte	public static byte parseByte(String)
short	Short	public static short parseShort(String)
int	Integer	public static integer parseInt(String)
long	Long	public static long parseLong(String)
float	Float	public static float parseFloat(String)
double	Double	public static double parseDouble(String)
char	Character	
boolean	Boolean	public static boolean parseBoolean(String)

How to use wrapper class methods

All the wrapper class methods are static in nature so we need to call these method using class.methodName().

- for Integer: `int x=Integer.parseInt(String);`
- for float: `float x=Float.parseFloat(String);`
- for double: `double x=Double.parseDouble(String);`

Each and every wrapper class contains the following generalized method for converting numeric String into fundamental values.



Here xxx represents any fundamental data type.

ACCESS MODIFIERS IN JAVA

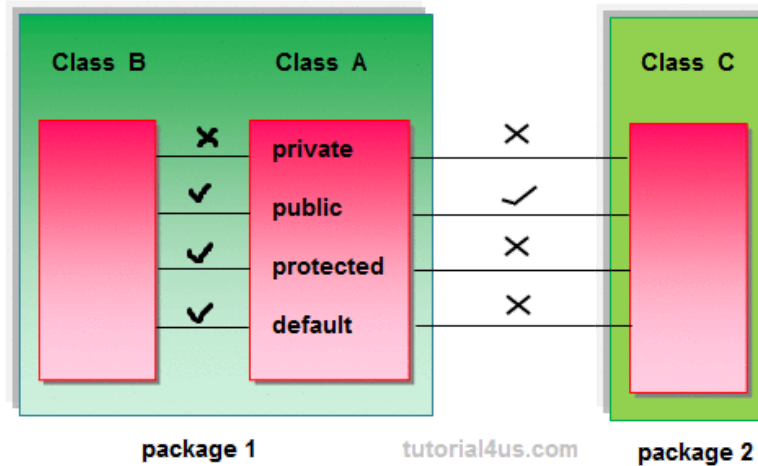
Access modifiers are those which are applied before data members or methods of a class. These are used to where to access and where not to access the data members or methods. In java programming these are classified into four types:

- Private
- Default (not a keyword)
- Protected
- Public

Note: Default is not a keyword (like public, private, protected are keyword)

If we are not using private protected and public keywords then JVM is by default taking as default access modifiers.

Access modifiers are always used for how to reuse the features within the package and access the package between class to class, interface to interface and interface to class. Access modifiers provides features accessing and controlling mechanism among the classes and interfaces.



Note: protected members of class is accessible within the same class and other class of same package and also accessible in inherited class of other package.

Rules for access modifiers:

The following diagram gives rules for Access modifiers.

Modifiers	Within Same Class	Within other class of Same package	Within derived class of other package	Within external Class of other package
Private (Class level A.S)	Yes	No	No	No
Default (Package level A.S)	Yes	Yes	No	No
Protected (Derived level A.S)	Yes	Yes	Yes	No
Public (Universal A.S)	Yes	Yes	Yes	Yes
A.S --> Access Specifier				Tutorial4us.com

private: private members of class in not accessible any where in program these are only accessible within the class. private are also called class level access modifiers.

Example

```
class Hello
```

```

{
private int a=20;
private void show()
{
System.out.println("Hello java");
}
}

public class Demo
{
public static void main(String args[])
{
Hello obj=new Hello();
System.out.println(obj.a); //Compile Time Error, you can't
access private data
obj.show(); //Compile Time Error, you can't access private
methods
}
}

```

public: public members of any class is accessible any where in program inside the same class and outside of class, within the same package and outside of the package. public are also called universal access modifiers.

Example

```

class Hello
{
public int a=20;
public void show()
{
System.out.println("Hello java");
}
}

```

```
}  
}  
public class Demo  
{  
    public static void main(String args[])  
    {  
        Hello obj=new Hello();  
        System.out.println(obj.a);  
        obj.show();  
    }  
}
```

Output

```
20  
Hello Java
```

protected: protected members of class is accessible within the same class and other class of same package and also accessible in inherited class of other package. protected are also called derived level access modifiers.

In below example we have create two packages pack1 and pack2. In pack1, class A is public so we can access this class outside of pack1 but method show is declared as a protected so it is only access outside of package pack1 only through inheritance.

Example

```
// save A.java
package pack1;
public class A
{
protected void show()
{
System.out.println("Hello Java");
}
}

//save B.java
package pack2;
import pack1.*;

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

Output

Hello Java

default: default members of class is accessible only within same class and other class of same package. default are also called package level access modifiers.

Example

```
//save by A.java
package pack;
class A
```

```

{
    void show()
    {
        System.out.println("Hello Java");
    }
}
//save by B.java
package pack2;
import pack1.*;
class B
{
    public static void main(String args[])
    {
        A obj = new A(); //Compile Time Error, can't access outside
        the package
        obj.show(); //Compile Time Error, can't access outside the
        package
    }
}

```

Output

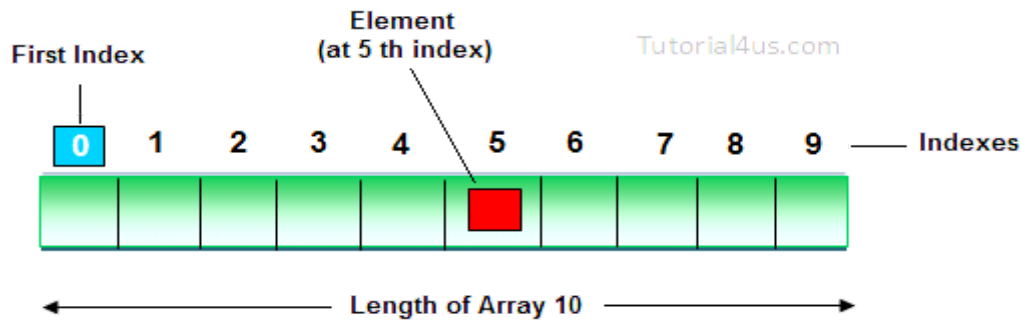
Hello Java

Note: private access modifiers is also known as native access modifiers, default access modifiers is also known as package access modifiers, protected access modifiers is also known as inherited access modifiers, public access modifiers is also known as universal access modifiers.

ARRAY IN JAVA

Array is a collection of similar type of data. It is fixed in size that means you can't increase the size of array at run time. It is

collection of homogeneous data elements. It store the value on the basis of index value.



Advantage of Array:

- **One variable can store multiple value:** The main advantage of array is we can represent multiple value under the same name.
- **Code Optimization:** No, need to declare a lot of variable of same type data, We can retrieve and short data easily.
- **Random access:** We can retrieve any data from array with the help of index value.

Disadvantage of Array

The main limitation of array is **Size Limit** when once we declare array there is no chance to increase and decrease the size of array according to our requirement, Hence memory point of view array concept is not recommended to use. To over come this limitation in java introduce collection concept.

TYPES OF ARRAY

There are two types of array in java.

- Single Dimensional Array
- Multidimensional Array

Array Declaration

Single dimension array declaration.

Syntax

1. `int[] a;`
2. `int a[];`
3. `int []a;`

Note: At the time of array declaration we can not specify the size of array. For Example `int[5] a;` this is wrong.

2D Array declaration.

Syntax

1. `int[][] a;`
2. `int a[][];`
3. `int [][]a;`
4. `int[] a[];`
5. `int[] []a;`
6. `int []a[];`

Array creation: Every array in a java is an object, Hence we can create array by using **new** keyword.

Syntax

```
int[] arr = new int[10]; // The size of array is 10.  
or  
int[] arr = {10,20,30,40,50};
```

Accessing array elements: Access the elements of array by using index value of an elements.

Syntax

```
arrayname[n-1];
```

Access Array Elements

```
int[] arr={10,20,30,40};
```

```
System.out.println("Element at 4th place"+arr[2]);
```

Example of Array

```
public class ArrayEx
{
    public static void main(String []args)
    {
        int arr[] = {10,20,30};
        for (int i=0; i < arr.length; i++)
        {
            System.out.println(arr[i]);
        }
    }
}
```

Output

```
10
20
30
```

Note:

- 1) At the time of array creation we must be specify the size of array otherwise get an compile time error. For Example
int[] a=new int[]; Invalid.
int[] a=new int[5]; Valid
- 2) If we specify array size as negative int value, then we will get run-time error, NegativeArraySizeException.
- 3) To specify array size the allowed data types are byte, short, int, char If we use other data type then we will get an Compile time error.

4) The maximum allowed size of array in java is 2147483647 (It is maximum value of int data type)

length vs length()

length: It is a final variable and only applicable for array. It represent size of array.

Example

```
int[] a=new int[10];  
System.out.println(a.length); // 10  
System.out.println(a.length()); // Compile time error
```

length(): It is final method applicable only for String objects. It represent number of character present in String.

Example

```
String s="Java";  
System.out.println(s.length()); // 4  
System.out.println(s.length); // Compile time error
```

STATIC, FINAL, THIS AND SUPER KEYWORDS USED IN JAVA

Final keyword: It is used to make a variable as a constant, Restrict method overriding, Restrict inheritance. It is used at variable level, method level and class level. In java language final keyword can be used in following way.

- Final at variable level
- Final at method level
- Final at class level
- **Final at variable level :** Final keyword is used to make a variable as a constant. This is similar to const in other

language. A variable declared with the final keyword cannot be modified by the program after initialization. This is useful to universal constants, such as "PI".

Final Keyword in java Example

```
public class Circle
{
    public static final double PI=3.14159;

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

- **Final at method level:** It makes a method final, meaning that sub classes cannot override this method. The compiler checks and gives an error if you try to override the method. When we want to restrict overriding, then make a method as a final.

Example

```
public class A
{
    public void fun1()
    {
        .....
    }
    public final void fun2()
    {
        .....
    }
}
```

```

}
class B extends A
{
public void fun1()
{
.....
}
public void fun2()
{
// it gives an error because we can not override final method
}
}

```

Example of final keyword at method level

Example

```

class Employee
{
final void disp()
{
System.out.println("Hello Good Morning");
}
}
class Developer extends Employee
{
void disp()
{
System.out.println("How are you ?");
}
}
class FinalDemo
{
public static void main(String args[])
{

```

```
Developer obj=new Developer();  
obj.disp();  
}  
}
```

Output

It gives an error

- **Final at class level:** It makes a class final, meaning that the class can not be inheriting by other classes. When we want to restrict inheritance then make class as a final.

Example

```
public final class A  
{  
.....  
.....  
}  
public class B extends A  
{  
// it gives an error, because we can not inherit final class  
}
```

Example of final keyword at class level

Example

```
final class Employee  
{  
int salary=10000;  
}  
class Developer extends Employee
```

```
{  
void show()  
{  
System.out.println("Hello Good Morning");  
}  
}  
class FinalDemo  
{  
public static void main(String args[])  
{  
Developer obj=new Developer();  
Developer obj=new Developer();  
obj.show();  
}  
}
```

Output

Output:

It gives an error

Static keyword: The **static keyword** is used in java mainly for memory management. It is used with variables, methods, blocks and nested class. It is a keyword that are used for share the same variable or method of a given class. This is used for a constant variable or a method that is the same for every instance of a class. The main method of a class is generally labeled static.

No object needs to be created to use static variable or call static methods, just put the class name before the static variable or method to use them. Static method cannot call non-static method.

In java language static keyword can be used for following

- variable (also known as class variable)
 - method (also known as class method)
 - block
 - nested class
-
- **Static variable :** If any variable we declared as static is known as static variable. Static variable is used for fulfill the common requirement. For Example company name of employees, college name of students etc. Name of the college is common for all students. The static variable allocate memory only once in class area at the time of class loading.

Advantage of static variable

Using static variable we make our program memory efficient (i.e it saves memory).

When and why we use static variable

Suppose we want to store record of all employee of any company, in this case employee id is unique for every employee but company name is common for all. When we create a static variable as a company name then only once memory is allocated otherwise it allocate a memory space each time for every employee.

Syntax for declare static variable:

```
public static variableName;
```

Syntax for declare static method:

```
public static void methodName()  
{
```

```
.....  
.....  
}
```

Syntax for access static methods and static variable

Syntax

```
className.variableName=10;  
className.methodName();
```

Example

```
public static final double PI=3.1415;  
public static void main(String args[])  
{  
.....  
.....  
}
```

Difference between static and final keyword

static keyword always fixed the memory that means that will be located only once in the program where as **final** keyword always fixed the value that means it makes variable values constant.

Note: As for as real time statement there concern every final variable should be declared the static but there is no compulsion that every static variable declared as final.

Example of static variable.

In the below example College_Name is always same, and it is declared as static.

Example

```

class Student
{
int roll_no;
String name;
static String College_Name="ITM";
}
class StaticDemo
{
public static void main(String args[])
{
Student s1=new Student();
s1.roll_no=100;
s1.name="abcd";
System.out.println(s1.roll_no);
System.out.println(s1.name);
System.out.println(Student.College_Name);
Student s2=new Student();
s2.roll_no=200;
s2.name="zyx";
System.out.println(s2.roll_no);
System.out.println(s2.name);
System.out.println(Student.College_Name);
}
}

```

Example

Output:

100

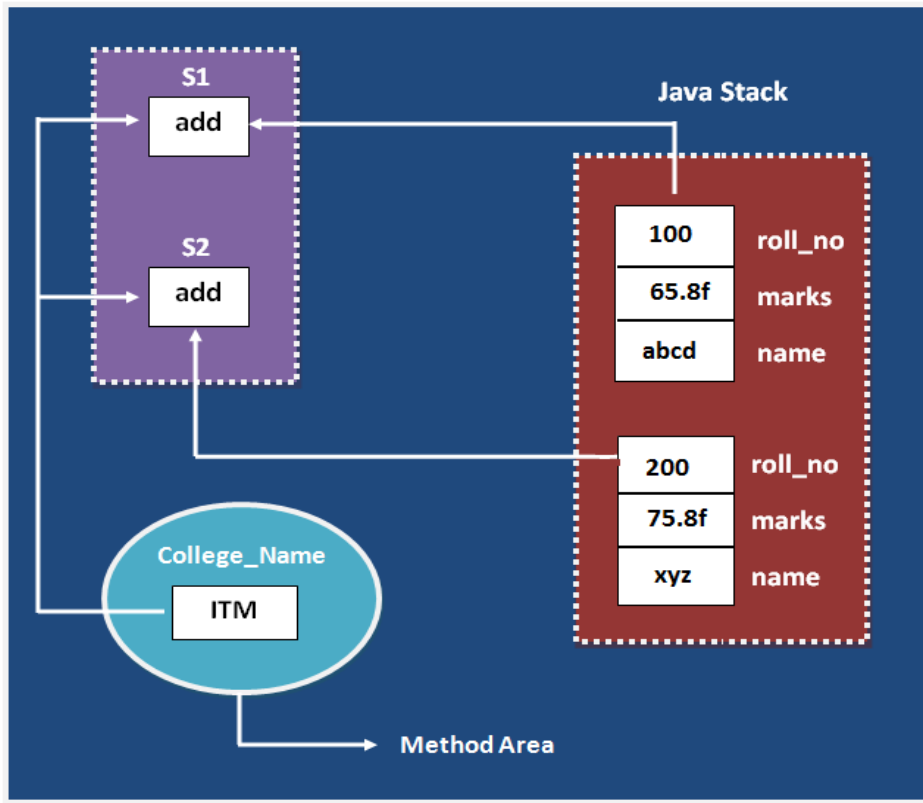
abcd

ITM

200

zyx

In the above example `College_Name` variable is commonly sharable by both S1 and S2 objects.



In the above image static data variable are store in method and non static variable is store in java stack. Read more about this in JVM Architecture chapter

Why main method is static ?

Because object is not required to call static method if `main()` is non-static method, then JVM create object first then call `main()` method due to that face the problem of extra memory allocation.

This keyword: **this** is a reference variable that refers to the current object. It is a keyword in java language represents current class object

Usage of this keyword

- It can be used to refer current class instance variable.
- `this()` can be used to invoke current class constructor.
- It can be used to invoke current class method (implicitly)
- It can be passed as an argument in the method call.
- It can be passed as argument in the constructor call.
- It can also be used to return the current class instance.

Why use **this** keyword in java ?

The main purpose of **using this keyword** is to differentiate the formal parameter and data members of class, whenever the formal parameter and data members of the class are similar then jvm get ambiguity (no clarity between formal parameter and member of the class)

To differentiate between formal parameter and data member of the class, the data member of the class must be preceded by "this".

"this" keyword can be use in two ways.

- `this .` (this dot)
- `this()` (this off)

this . (this dot): It can be used to differentiate variable of class and formal parameters of method or constructor. **"this"** keyword are used for two purpose, they are

- It always points to current class object.
- Whenever the formal parameter and data member of the class are similar and JVM gets an ambiguity (no clarity between formal parameter and data members of the class).

To differentiate between formal parameter and data member of the class, the data members of the class must be preceded by **"this"**.

Syntax

this.data member of current **class**.

Note: If any variable is preceded by **"this"** JVM treated that variable as class variable.

Example without using this keyword

```
class Employee
{
    int id;
    String name;

    Employee(int id,String name)
    {
        id = id;
        name = name;
    }
    void show()
    {
        System.out.println(id+" "+name);
    }
    public static void main(String args[])
    {
        Employee e1 = new Employee(111,"Harry");
        Employee e2 = new Employee(112,"Jacy");
        e1.show();
        e2.show();
    }
}
```

Output

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.

Example of this keyword in java

```
class Employee
{
    int id;
    String name;

    Employee(int id,String name)
    {
        this.id = id;
        this.name = name;
    }

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

    public static void main(String args[])
    {
        Employee e1 = new Employee(111,"Harry");
        Employee e2 = new Employee(112,"Jacy");
        e1.show();
        e2.show();
    }
}
```

```
}
```

Output

```
111 Harry  
112 Jacy
```

Note 1: The scope of "**this**" keyword is within the class.

Note 2: The main purpose of using "**this**" keyword in real life application is to differentiate variable of class or formal parameters of methods or constructor (it is highly recommended to use the same variable name either in a class or method and constructor while working with similar objects).

Difference between this and super keyword

Super keyword is always pointing to base class (scope outside the class) features and "**this**" keyword is always pointing to current class (scope is within the class) features.

Example when no need of this keyword

```
class Employee  
{  
    int id;  
    String name;  
  
    Employee(int i,String n)  
    {  
        id = i;  
        name = n;  
    }  
    void show()  
    {  
        System.out.println(id+" "+name);  
    }  
}
```

```
public static void main(String args[])
{
    Employee e1 = new Employee(111,"Harry");
    Employee e2 = new Employee(112,"Jacy");
    e1.show();
    e2.show();
}
```

Output

```
111 Harry
112 Jacy
```

In the above example, no need of use this keyword because parameter (formal arguments) and instance variables are different. This keyword is only use when parameter (formal arguments) and instance variables are same.

this ():which can be used to call one constructor within the another constructor without creation of objects multiple time for the same class.

Syntax

```
this(); // call no parametrized or default constructor
this(value1,value2,.....) //call parametrize constructor
```

this keyword used to invoke current class method (implicitly)

By using this keyword you can invoke the method of the current class. If you do not use the this keyword, compiler automatically adds this keyword at time of invoking of the method.

Example of this keyword

```

class Student
{
    void show()
    {
        System.out.println("You got A+");
    }
    void marks()
    {
        this.show(); //no need to use this here because compiler does
it.
    }
    void display()
    {
        show(); //compiler act marks() as this.marks()
    }
    public static void main(String args[])
    {
        Student s = new Student();
        s.display();
    }
}

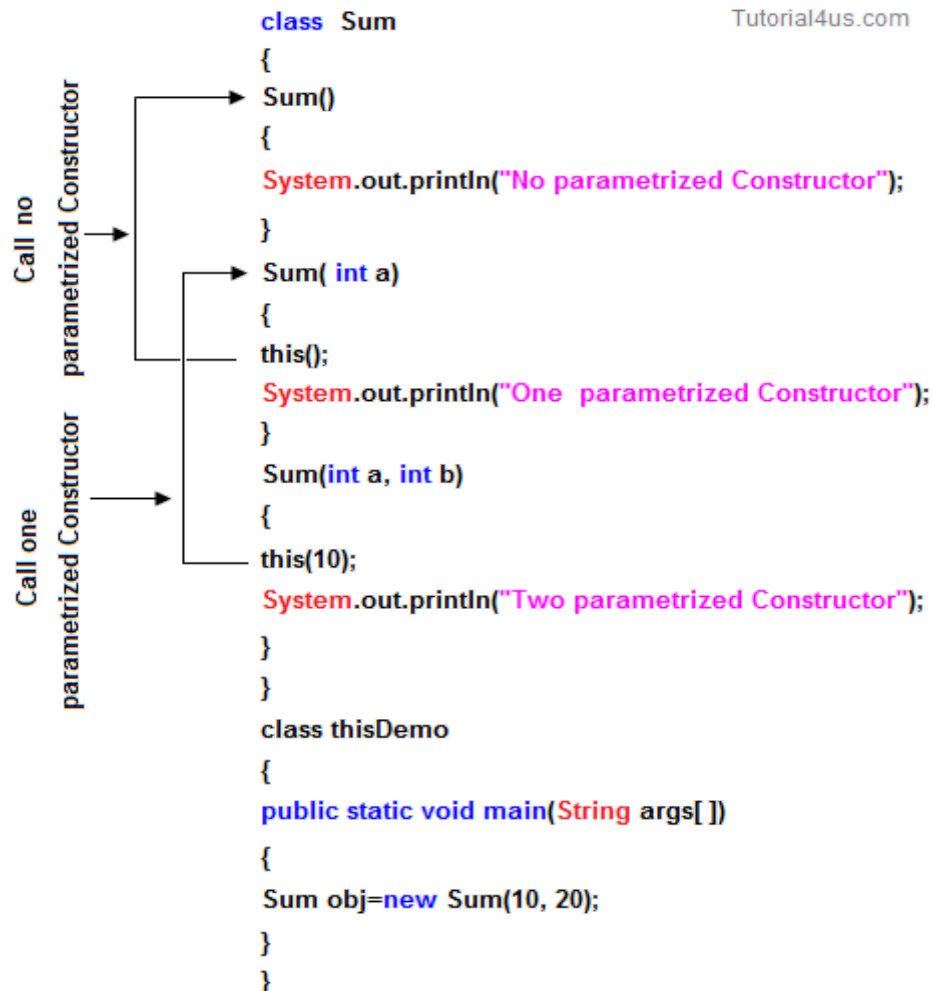
```

Syntax

You got A+

Rules to use this()

this() always should be the first statement of the constructor. One constructor can call only other single constructor at a time by using **this()**.



Super keyword : Super keyword in java is a reference variable that is used to refer parent class object. Super is an implicit keyword create by JVM and supply each and every java program for performing important role in three places.

- At variable level
- At method level
- At constructor level

Need of super keyword:

Whenever the derived class is inherits the base class features, there is a possibility that base class features are similar to derived class features and JVM gets an ambiguity. In order to differentiate between base class features and derived class features must be preceded by super keyword.

Syntax

super.baseclass features.

Super at variable level:

Whenever the derived class inherit base class data members there is a possibility that base class data member are similar to derived class data member and JVM gets an ambiguity.

In order to differentiate between the data member of base class and derived class, in the context of derived class the base class data members must be preceded by super keyword.

Syntax

super.baseclass datamember name

if we are not writing super keyword before the base class data member name than it will be referred as current class data member name and base class data member are hidden in the context of derived class.

Program without using super keyword

Example

```
class Employee
{
float salary=10000;
}
class HR extends Employee
{
float salary=20000;
void display()
{
```

```

System.out.println("Salary: "+salary); //print current class
salary
}
}
class Supervariable
{
public static void main(String[] args)
{
HR obj=new HR();
obj.display();
}
}

```

Output

Salary: 20000.0

In the above program in Employee and HR class salary is common properties of both class the instance of current or derived class is referred by instance by default but here we want to refer base class instance variable that is why we use super keyword to distinguish between parent or base class instance variable and current or derived class instance variable.

Program using super keyword at variable level

Example

```

class Employee
{
float salary=10000;
}
class HR extends Employee
{
float salary=20000;
void display()

```

```

{
System.out.println("Salary: "+super.salary);//print base class
salary
}
}
class Supervariable
{
public static void main(String[] args)
{
HR obj=new HR();
obj.display();
}
}

```

Output

Salary: 10000.0

Super at method level

The **super keyword** can also be used to invoke or call parent class method. It should be use in case of method overriding. In other word **super keyword** use when base class method name and derived class method name have same name.

Example of super keyword at method level

Example

```

class Student
{
void message()
{
System.out.println("Good Morning Sir");
}
}

```

```

}
}

class Faculty extends Student
{
void message()
{
System.out.println("Good Morning Students");
}

void display()
{
message();//will invoke or call current class message()
method
super.message();//will invoke or call parent class message()
method
}

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

```

Output

```

Good Morning Students
Good Morning Sir

```

In the above example Student and Faculty 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 of local is high.

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 where super is not required

Example

```
class Student
{
void message()
{
System.out.println("Good Morning Sir");
}
}

class Faculty extends Student
{

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

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

Output

Good Morning Sir

Super at constructor level

The super keyword can also be used to invoke or call the parent class constructor. Constructors are called from bottom to top and executed from top to bottom.

To establish the connection between base class constructor and derived class constructors JVM provides two implicit methods they are:

- Super()
- Super(...)

Super() It is used for calling super class default constructor from the context of derived class constructors.

Super keyword used to call base class constructor

Syntax

```
class Employee
{
    Employee()
    {
        System.out.println("Employee class Constructor");
    }
}
class HR extends Employee
{
    HR()
    {
```

```

super(); //will invoke or call parent class constructor
System.out.println("HR class Constructor");
}
}
class Supercons
{
public static void main(String[] args)
{
HR obj=new HR();
}
}

```

Output

```

Employee class Constructor
HR class Constructor

```

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

In constructor, default constructor is provided by compiler automatically but it also adds **`super()`** before the first statement of constructor. If you are creating your own constructor and you do not have either `this()` or `super()` as the first statement, compiler will provide `super()` as the first statement of the constructor.

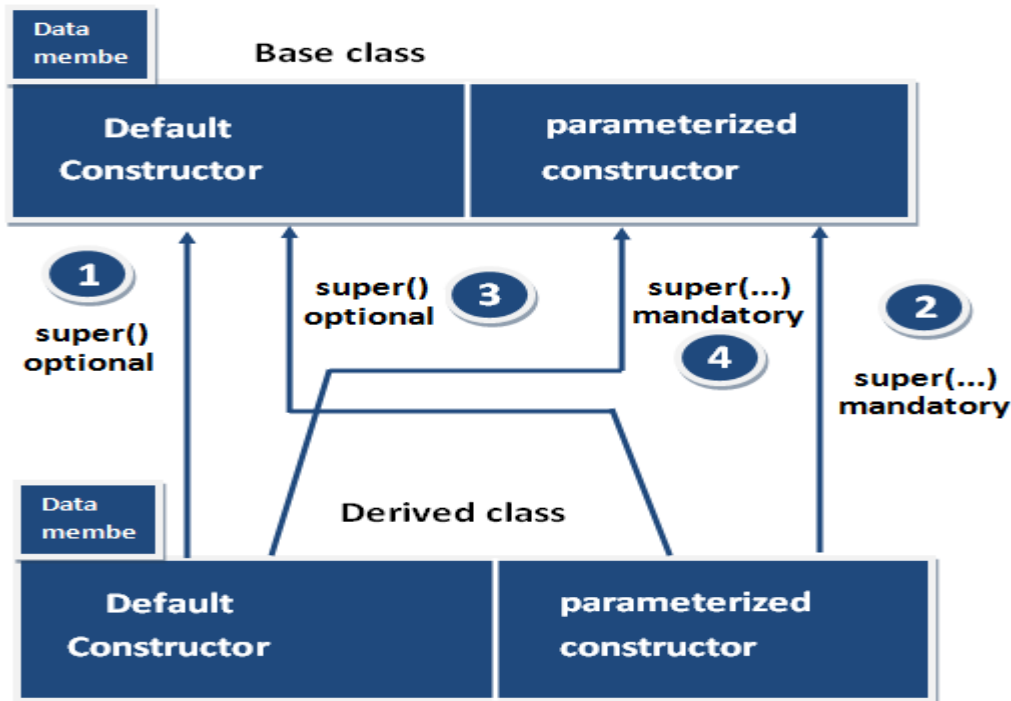
Super(...): It is used for calling super class parameterize constructor from the context of derived class constructor.

Important rules

Whenever we are using either `super()` or `super(...)` in the derived class constructors the **`super`** always must be as a first

executable statement in the body of derived class constructor otherwise we get a compile time error.

The following diagram use possibilities of using super() and super(.....)



Rule 1 and Rule 3

Whenever the derived class constructor want to call default constructor of base class, in the context of derived class constructors we write `super()`. Which is optional to write because every base class constructor contains single form of default constructor?

Rule 2 and Rule 4

Whenever the derived class constructor wants to call parameterized constructor of base class in the context of derived class constructor we must write `super(...)`. which is mandatory to write because a base class may contain multiple forms of parameterized constructors.

OBJECT AND CLASS IN JAVA

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);  
    }  
}
```

Creating “main” in a separate class

We can create the main method in a separate class, but during compilation you need to make sure that you compile the class with the “main” method.

```
class Demo
{
private int x,y,z;
public void input() {
x=10;
y=15;
}
public void sum()
{
z=x+y;
}
public void print_data()
{
System.out.println("Answer is =" +z);
}
}
class SumDemo
{
public static void main(String args[])
{
Demo object=new Demo();
object.input();
object.sum();
object.print_data();
}
}
```

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();
```

```
}  
} OUTPUT:  
    111 Karan  
    222 Aryan
```

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.

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 annonymous
    object
}
}

```

Output: Factorial is 120

METHODS WITH PARAMETERS

Following program shows the method with passing parameter.

```

class prg
{
int n,n2,sum;
public void take(int x,int y)
{
n=x;
n2=y;
}
public void sum()
{
sum=n+n2;
}
public void print()
{
System.out.println("The Sum is"+sum);
}
}
class prg1
{
public static void main(String args[])
{
prg obj=new prg();

```

```
obj.take(10,15);  
obj.sum();  
obj.print();  
}  
}
```

Methods with a Return Type

When method return some value that is the type of that method.

For Example: some methods are with parameter but that method did not return any value that means type of method is void. And if method return integer value then the type of method is an integer.

Following program shows the method with their return type.

Class Demo1

```
{  
int n,n2;  
public void take( int x,int y)  
{  
n=x;  
n=y;  
}  
public int process()  
{  
return (n+n2);  
}  
}  
class prg  
{  
public static void main(String args[])  
{
```

```
int sum;  
Demo1 obj=new Demo1();  
obj.take(15,25);  
sum=obj.process();  
System.out.println("The sum is"+sum);  
}  
}
```

Output:

The sum is25

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**.

There are two ways to overload the method in java

- By changing number of arguments
- By changing the data type

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

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);  
  
    }  
}
```

Output:

30

40

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);

    }
}

```

Output:

21.0

40

Passing Objects as Parameters: Objects can even be passed as parameters.

```

class para123
{
    int n,n2,sum,mul;
    public void take(int x,int y)
    {
        n=x;
        n2=y;
    }

    public void sum()
    {
        sum=n+n2;
        System.out.println("The Sum is"+sum);
    }
}

```

```

}
public void take2(para123 obj)
{
n=obj.n;
n2=obj.n2;
}
public void multi()
{
mul=n*n2;
System.out.println("Product is"+mul);
}
}
class DemoPara
{
public static void main(String args[])
{
para123 ob=new para123();
ob.take(3,7);
ob.sum();
ob.take2(ob);
ob.multi();
}
}

```

Output:

C:\cc>javac DemoPara.java

C:\cc>java DemoPara

The Sum is10

Product is21

We have defined a method “*take2*” that declares an object named obj as parameter. We have passed ob to our method. The method “*take2*” automatically gets 3,7 as values for n and n2.

Passing Values to methods and Constructor:

These are two different ways of supplying values to methods. Classified under these two titles -

1.Pass by Value

2.Pass by Address or Reference

1. Pass by Value-When we pass a data type like int, float or any other datatype to a method or some constant values like(15,10). They are all passed by value. A copy of variable's value is passed to the receiving method and hence any changes made to the values do not affect the actual variables.

```
class Demopbv
{
    int n,n2;
    public void get(int x,int y)
    {
        x=x*x; //Changing the values of passed arguments
        y=y*y; //Changing the values of passed arguments
    }
}

class Demo345
{
    public static void main(String args[])
    {
        int a,b;
        x=1;
        y=2;
        System.out.println("Initial Values of a & b "+a+" "+b);
        Demopbv obj=new Demopbv();
        obj.get(&x,&y);
    }
}
```

```
System.out.println("Final Values "+a+" "+b);  
}  
}
```

Output:

```
C:\cc>javac Demo345.java
```

```
C:\cc>java Demo345
```

Initial Values of a & b 1 2

Final Values 1 4

2. Pass by Reference: Objects are always passed by reference.

When we pass a value by reference, the reference or the memory address of the variables is passed. Thus any changes made to the argument causes a change in the values which we pass. Demonstrating Pass by Reference---

```
class pass_by_ref  
{  
int n,n2;  
public void get(int a,int b)  
{  
n=a;  
n2=b;  
}  
public void doubleit(pass_by_ref temp)  
{  
temp.n=temp.n*2;  
temp.n2=temp.n2*2;  
}  
}  
class apply7  
{  
public static void main(String args[])  
{  
int x=5,y=10;
```

```
pass_by_ref obj=new pass_by_ref();  
obj.get(x,y); //Pass by Value  
System.out.println("Initial Values are-- ");  
System.out.println(+obj.n);  
System.out.println(+obj.n2);  
obj.doubleit(obj); //Pass by Reference  
System.out.println("Final Values are");  
System.out.println(+obj.n);  
System.out.println(+obj.n2);  
}  
}
```

CONSTRUCTOR IN JAVA

constructor in Java is a special member method which will be called implicitly (automatically) by the JVM whenever an object is created for placing user or programmer defined values in place of default values. In a single word constructor is a special member method which will be called automatically whenever object is created.

The purpose of constructor is to initialize an object called object initialization. Constructors are mainly create for initializing the object. Initialization is a process of assigning user defined values at the time of allocation of memory space.

Syntax

```

className()
{
.....
.....
}

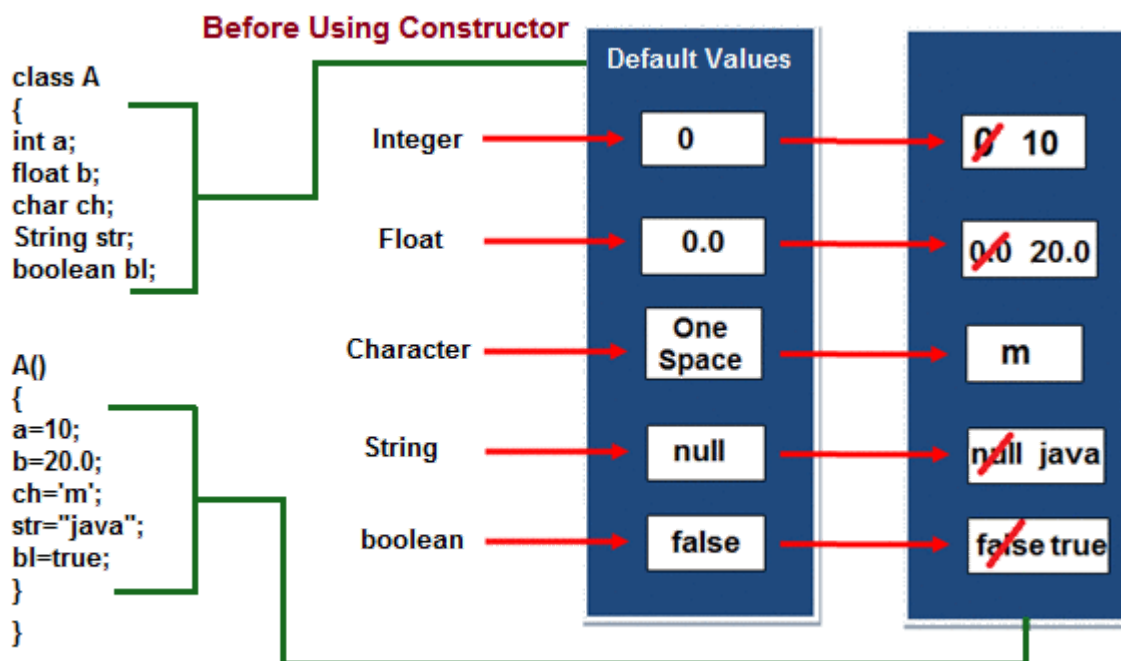
```

Advantages of constructors in Java

- A constructor eliminates placing the default values.
- A constructor eliminates calling the normal or ordinary method implicitly.

How Constructor eliminate default values ?

Constructor are mainly used for eliminate default values by user defined values, whenever we create an object of any class then its allocate memory for all the data members and initialize there default values. To eliminate these default values by user defined values we use constructor.



Tutorial4us.com

Example

```

class Sum
{
int a,b;

```

```
Sum()
{
a=10;
b=20;
}
public static void main(String s[])
{
Sum s=new Sum();
c=a+b;
System.out.println("Sum: "+c);
}
}
```

Output

Sum: 30

In above example when we create an object of "Sum" class then constructor of this class call and initialize user defined value in a=10 and b=20. if we can not create constructor of Sum class then it print " Sum: 0 " because default values of integer is zero.

Rules or properties of a constructor

- Constructor will be called automatically when the object is created.
- Constructor name must be similar to name of the class.
- Constructor should not return any value even void also. Because basic aim is to place the value in the object. (if we write the return type for the constructor then that constructor will be treated as ordinary method).
- Constructor definitions should not be static. Because constructors will be called each and every time, whenever an object is creating.

- Constructor should not be private provided an object of one class is created in another class (Constructor can be private provided an object of one class created in the same class).
- Constructors will not be inherited from one class to another class (Because every class constructor is create for initializing its own data members).
- The access specifier of the constructor may or may not be private.
 1. If the access specifier of the constructor is private then an object of corresponding class can be created in the context of the same class but not in the context of some other classes.
 2. If the access specifier of the constructor is not private then an object of corresponding class can be created both in the same class context and in other class context.

Difference between Method and Constructor

	Method	Constructor
1	Method can be any user defined name	Constructor must be class name
2	Method should have return type	It should not have any return type (even void)
3	Method should be called explicitly either with object reference or class reference	It will be called automatically whenever object is created
4	Method is not provided by compiler in any case.	The java compiler provides a default constructor if we do not have any constructor.

Types of constructors

Based on creating objects in Java constructor are classified in two types. They are

- Default or no argument Constructor
- Parameterized constructor.

Default Constructor

A constructor is said to be default constructor if and only if it never take any parameters.

If any class does not contain at least one user defined constructor than the system will create a default constructor at the time of compilation it is known as system defined default constructor.

Syntax

```
class className
{
..... // Call default constructor
clsname ()
{
Block of statements; // Initialization
}
.....
}
```

Note: System defined default constructor is created by java compiler and does not have any statement in the body part. This constructor will be executed every time whenever an object is created if that class does not contain any user defined constructor.

Example of default constructor.

In below example, we are creating the no argument constructor in the Test class. It will be invoked at the time of object creation.

Example

```
//TestDemo.java
class Test
```

```

{
int a, b;
Test ()
{
System.out.println("I am from default Constructor...");
a=10;
b=20;
System.out.println("Value of a: "+a);
System.out.println("Value of b: "+b);
}
};
class TestDemo
{
public static void main(String [] args)
{
Test t1=new Test ();
}
};

```

Output

Output:
I am from default Constructor...
Value of a: 10
Value of b: 20

Rule-1:

Whenever we create an object only with default constructor, defining the default constructor is optional. If we are not defining default constructor of a class, then JVM will call automatically system defined default constructor. If we define, JVM will call user defined default constructor.

Purpose of default constructor?

Default constructor provides the default values to the object like 0, 0.0, null etc. depending on their type (for integer 0, for string null).

Example of default constructor that displays the default values

```
class Student
{
    int roll;
    float marks;
    String name;
    void show()
    {
        System.out.println("Roll: "+roll);
        System.out.println("Marks: "+marks);
        System.out.println("Name: "+name);
    }
}

class TestDemo
{
    public static void main(String [] args)
    {
        Student s1=new Student();
        s1.show();
    }
}
```

Output

```
Roll: 0
Marks: 0.0
Name: null
```

Explanation: In the above class, we are not creating any constructor so compiler provides a default constructor. Here 0, 0.0 and null values are provided by default constructor.

parameterized constructor

If any constructor contain list of variable in its signature is known as parameterized constructor. A parameterized constructor is one which takes some parameters.

Syntax

```
class ClassName
{
.....
ClassName(list of parameters) //parameterized constructor
{
.....
}
.....
}
```

Syntax to call parametrized constructor

```
ClassName objref=new ClassName(value1, value2,.....);
OR
new ClassName(value1, value2,.....);
```

Example of Parametrized Constructor

```
class Test
{
int a, b;
Test(int n1, int n2)
{
System.out.println("I am from Parameterized Constructor...");
a=n1;
b=n2;
```

```

System.out.println("Value of a = "+a);
System.out.println("Value of b = "+b);
}
};
class TestDemo1
{
public static void main(String k [])
{
Test t1=new Test(10, 20);
}
};

```

Important points Related to Parameterized Constructor

- Whenever we create an object using parameterized constructor, it must be define parameterized constructor otherwise we will get compile time error. Whenever we define the objects with respect to both parameterized constructor and default constructor, It must be define both the constructors.
- In any class maximum one default constructor but 'n' number of parameterized constructors.

Example of default constructor, parameterized constructor and overloaded constructor

Example

```

class Test
{
int a, b;
Test ()
{
System.out.println("I am from default Constructor...");
a=1;
b=2;
System.out.println("Value of a =" +a);
}
}

```

```

System.out.println("Value of b =" + b);
}
Test (int x, int y)
{
System.out.println("I am from double Parameterized
Constructor");
a=x;
b=y;
System.out.println("Value of a =" + a);
System.out.println("Value of b =" + b);
}
Test (int x)
{
System.out.println("I am from single Parameterized
Constructor");
a=x;
b=x;
System.out.println("Value of a =" + a);
System.out.println("Value of b =" + b);
}
Test (Test T)
{
System.out.println("I am from Object Parameterized
Constructor...");
a=T.a;
b=T.b;
System.out.println("Value of a =" + a);
System.out.println("Value of b =" + b);
}
};
class TestDemo2
{
public static void main (String k [])

```

```
{  
Test t1=new Test ();  
Test t2=new Test (10, 20);  
Test t3=new Test (1000);  
Test t4=new Test (t1);  
}  
};
```

Note By default the parameter passing mechanism is call by reference.

Constructor Overloading

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 the number of parameters, and their type.

In other words whenever same constructor is existing multiple times in the same class with different number of parameters or order of parameters or type of parameters is known as **Constructor overloading**.

In general constructor overloading can be used to initialized same or different objects with different values.

Syntax

```
class ClassName  
{  
    ClassName()  
    {  
        .....  
        .....  
    }  
    ClassName(datatype1 value1)  
    {.....}  
    ClassName(datatype1 value1, datatype2 value2)  
    {.....}
```



```
ClassName(datatype2 variable2)
{.....}
ClassName(datatype2 value2, datatype1 value1)
{.....}
.....
}
```

Why overriding is not possible at constructor level.

The scope of constructor is within the class so that it is not possible to achieved overriding at constructor level.

Inheritance in Java

The process of obtaining the data members and methods from one class to another class is known as **inheritance**. It is one of the fundamental features of object-oriented programming.

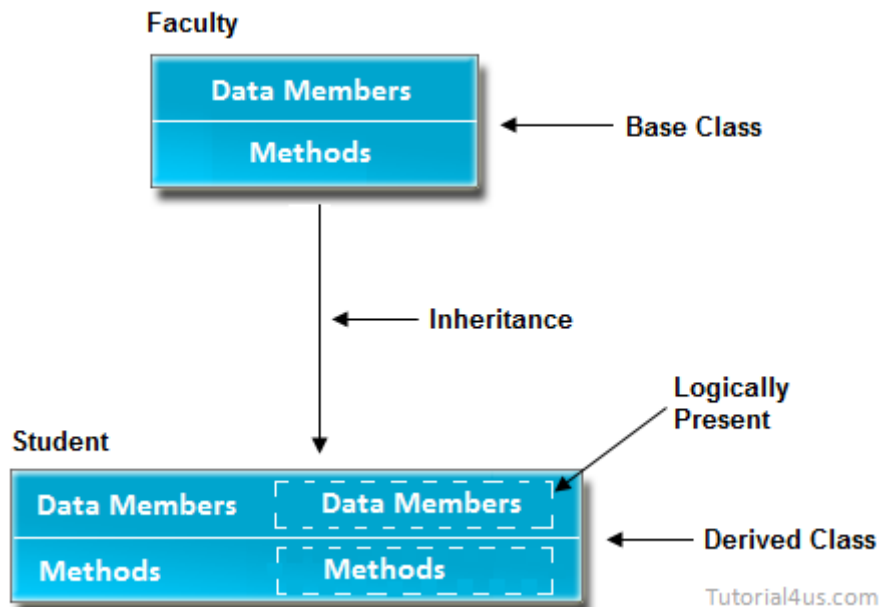
Why use Inheritance ?

- For Method Overriding (used for Runtime Polymorphism).
- It's main uses are to enable polymorphism and to be able to reuse code for different classes by putting it in a common super class
- For code Re-usability

Syntax of Inheritance

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

The following diagram use view about inheritance.



In the above diagram data members and methods are represented in broken line are inherited from faculty class and they are visible in student class logically.

Advantage of inheritance

If we develop any application using concept of Inheritance than that application have following advantages,

- Application development time is less.
- Application take less memory.
- Application execution time is less.
- Application performance is enhance (improved).
- Redundancy (repetition) of the code is reduced or minimized so that we get consistence results and less storage cost.

Note: In Inheritance the scope of access modifier increasing is allow but decreasing is not allow. Suppose in parent class method access modifier is default then it's present in child class with default or public or protected access modifier but not private(it decreased scope).

Types of Inheritance

Based on number of ways inheriting the feature of base class into derived class we have five types of inheritance; they are:

- Single inheritance
- Multiple inheritance
- Hierarchical inheritance
- Multilevel inheritance
- Hybrid inheritance

Single inheritance

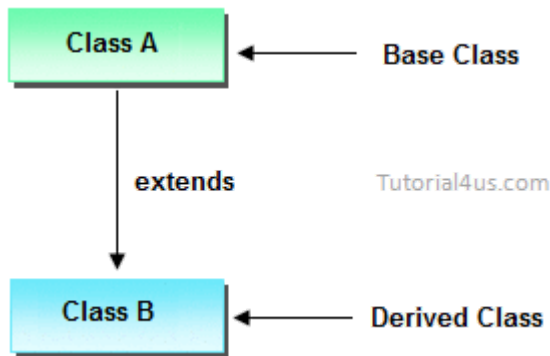
In single inheritance there exists single base class and single derived class.

Example of Single Inheritance

```
class Faculty
{
float salary=30000;
}
class Science extends Faculty
{
float bonous=2000;
public static void main(String args[])
{
Science obj=new Science();
System.out.println("Salary
is:"+obj.salary);
System.out.println("Bonous
is:"+obj.bonous);
}
}
```

Output

Salary is: 30000.0
Bonous is: 2000.0

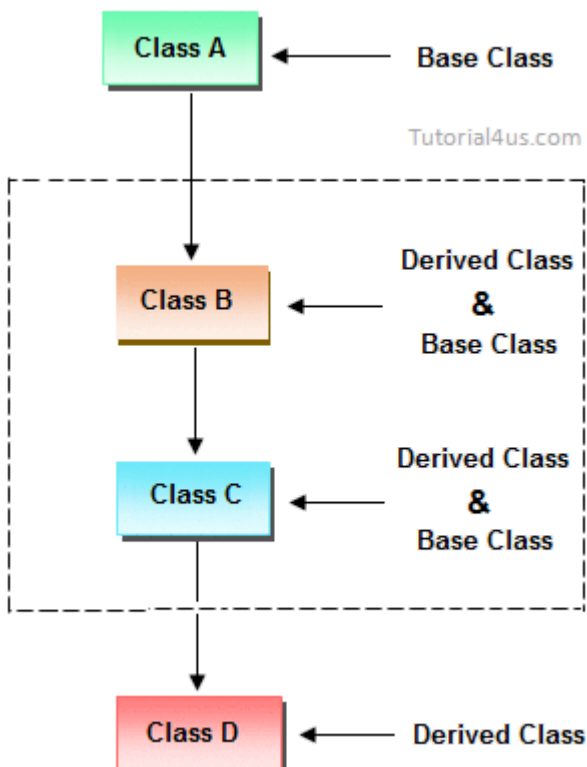


Multilevel inheritances in Java

In Multilevel inheritances there exists single base class, single derived class and multiple intermediate base classes.

Single base class + single derived class + multiple intermediate base classes.

Intermediate base classes:An intermediate base class is one in one context with access derived class and in another context same class access base class.



Hence all the above three inheritance types are supported by both classes and interfaces.

Example of Multilevel Inheritance

```
class Faculty
{
float total_sal=0, salary=30000;
}

class HRA extends Faculty
{
float hra=3000;
}

class DA extends HRA
{
float da=2000;
}

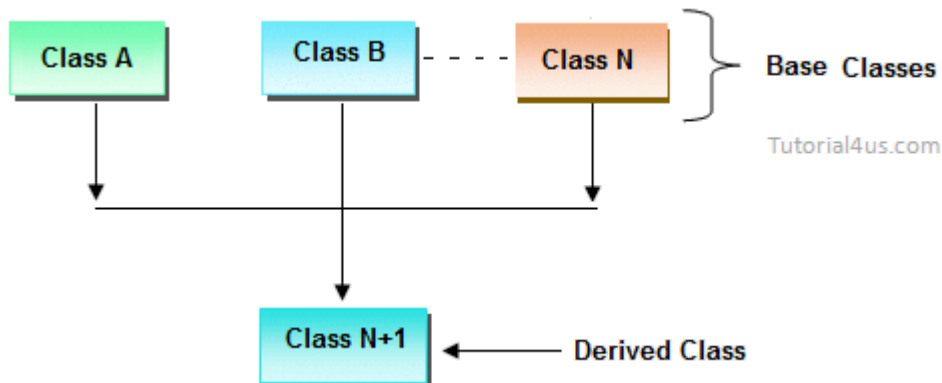
class Science extends DA
{
float bonous=2000;
public static void main(String args[])
{
Science obj=new Science();
obj.total_sal=obj.salary+obj.hra+obj.da+obj
.bonous;
System.out.println("Total Salary
is:"+obj.total_sal);
}
}
```

Output

Total Salary is: 37000.0

Multiple inheritance

In multiple inheritance there exist multiple classes and single derived class.

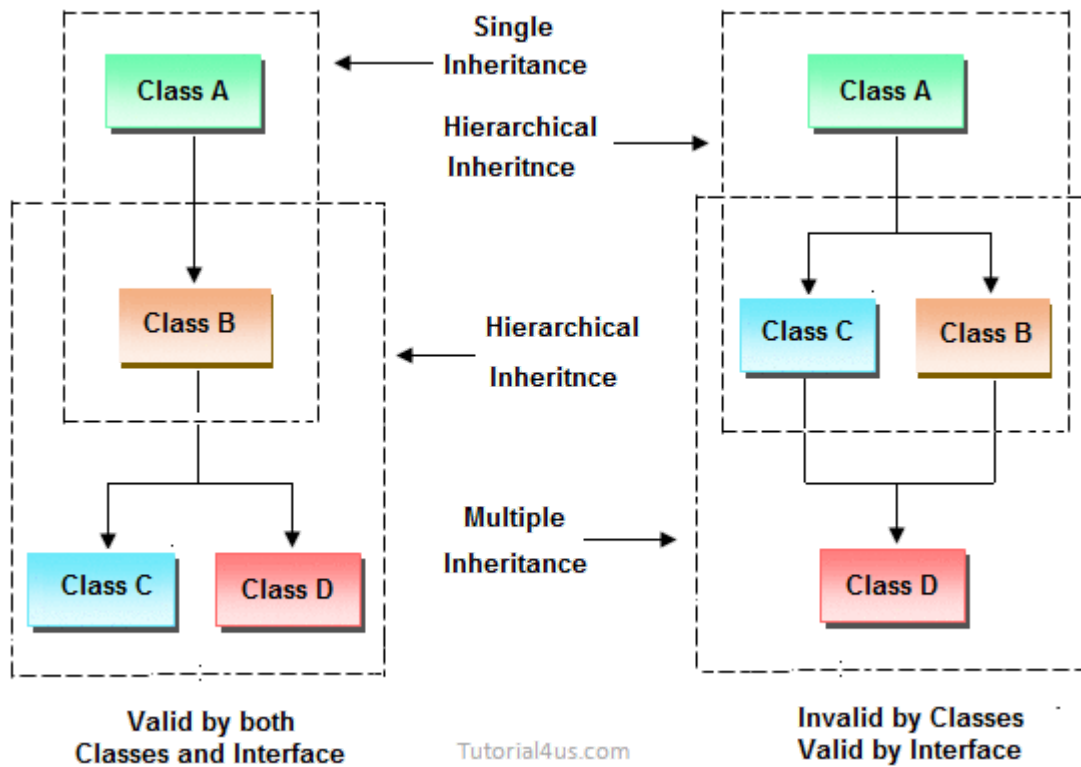


The concept of multiple inheritance is not supported in java through concept of classes but it can be supported through the concept of interface.

Hybrid inheritance

Combination of any inheritance type

In the combination if one of the combination is multiple inheritance then the inherited combination is not supported by java through the classes concept but it can be supported through the concept of interface.

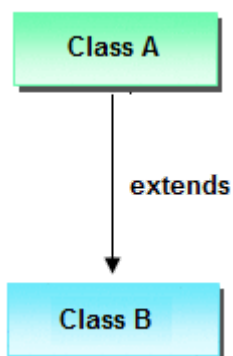


Inheriting the feature from base class to derived class

In order to inherit the feature of base class into derived class we use the following syntax

Syntax

```
class ClassName-2 extends ClasssName-1
{
variable declaration;
Method declaration;
}
```



Explanation

1. ClassName-1 and ClassName-2 represents name of the base and derived classes respectively.
2. extends is one of the keyword used for inheriting the features of base class into derived class it improves the functionality of derived class.

Important Points for Inheritance:

- In java programming one derived class can extends only one base class because java programming does not support multiple inheritance through the concept of classes, but it can be supported through the concept of Interface.
- Whenever we develop any inheritance application first create an object of bottom most derived class but not for top most base class.
- When we create an object of bottom most derived class, first we get the memory space for the data members of top most base class, and then we get the memory space for data member of other bottom most derived class.
- Bottom most derived class contains logical appearance for the data members of all top most base classes.
- If we do not want to give the features of base class to the derived class then the definition of the base class must be preceded by final hence final base classes are not reusable or not inheritable.
- If we are do not want to give some of the features of base class to derived class than such features of base class must be as private hence private features of base class are not inheritable or accessible in derived class.
- Data members and methods of a base class can be inherited into the derived class but constructors of base class can not be inherited because every constructor of a

class is made for initializing its own data members but not made for initializing the data members of other classes.

- An object of base class can contain details about features of same class but an object of base class never contains the details about special features of its derived class (this concept is known as scope of base class object).
- For each and every class in java there exists an implicit predefined super class called java.lang.Object. because it provides garbage collection facilities to its sub classes for collecting un-used memory space and improved the performance of java application.

Example of Inheritance

```
class Faculty
{
float salary=30000;
}
class Science extends Faculty
{
float bonous=2000;
public static void main(String args[])
{
Science obj=new Science();
System.out.println("Salary
is:"+obj.salary);
System.out.println("Bonous
is:"+obj.bonous);
}
}
```

Output

```
Salary is: 30000.0
Bonous is: 2000.0
```

Why multiple inheritance is not supported in java?

Due to ambiguity problem java does not support multiple inheritance at class level.

Example

```
class A
{
void disp()
{
System.out.println("Hello");
}
}
class B
{
void disp()
System.out.println("How are you ?");
}
}
class C extends A,B //suppose if it were
{
Public Static void main(String args[])
{
C obj=new C();
obj.disp();//Now which disp() method would
be invoked?
}
}
```

In above code we call both class A and class B disp() method then it confusion which class method is call. So due to this ambiguity problem in java do not use multiple inheritance at class level, but it support at interface level.

Method Overriding in Java

Whenever same method name is existing in both base class and derived class with same types of parameters or same order of parameters is known as **method Overriding**. Here we will discuss about **Overriding in Java**.

Note: Without Inheritance method overriding is not possible.

Advantage 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 Method Overriding

- method must have same name as in the parent class.
- method must have same parameter as in the parent class.
- must be IS-A relationship (inheritance).

Understanding the problem without method overriding

Lets understand the problem that we may face in the program if we do not use method overriding.

Example Method Overriding in Java

```
class Walking
{
void walk()
{
System.out.println("Man walking fastly");
}
}
```

```
class OverridingDemo
{
public static void main(String args[])
{
Man obj = new Man();
obj.walk();
}
}
```

Output

Man walking

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

Example of method overriding in Java

In this example, we have defined the walk 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.

Example

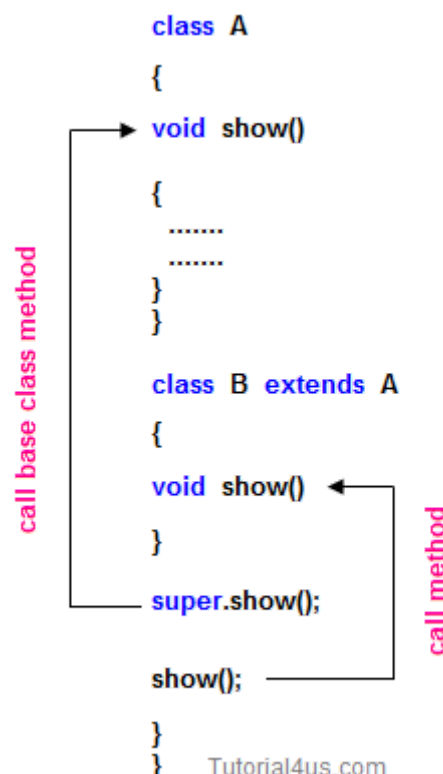
```
class Walking
{
void walk()
{
System.out.println("Man walking fastly");
}
}
class Man extends walking
{
void walk()
{
```

```
System.out.println("Man walking slowly");
}
}
```

```
class OverridingDemo
{
public static void main(String args[])
{
Man obj = new Man();
obj.walk();
}
}Output
Man walking slowly
```

Note: Whenever we are calling overridden method using derived class object reference the highest priority is given to current class (derived class). We can see in the above example high priority is derived class.

Note: super. (super dot) can be used to call base class overridden



method in the derived class.

Accessing properties of base class with respect to derived class object

```
class A
{
int x;
void f1()
{
x=10;
System.out.println(x);
}
void f4()
{
System.out.println("this is f4()");
System.out.println("-----");
}
};
class B extends A
{
int y;
void f1()
{
int y=20;
System.out.println(y);
System.out.println("this is f1()");
System.out.println("-----");
}
};
class C extends A
{
int z;
void f1()
{
z=10;
```

```

System.out.println(z);
System.out.println("this is f1()");
}
};
class Override
{
public static void main(String[] args)
{
A a1=new B();
a1.f1();
a1.f4();
A c1=new C();
c1.f1();
c1.f4();
}
}

```

Example of Implement overriding concept

```

class Person
{
String name;
void sleep(String name)
{
this.name=name;
System.out.println(this.name + "is
sleeping+8hr/day");
}
void walk()
{
System.out.println("this is walk()");
System.out.println("-----");
}
};
class Student extends Person

```

```

{
void writExams()
{
System.out.println("only student write the
exam");
}
void sleep(String name)
{
super.name=name;
System.out.println(super.name +"is sleeping
6hr/day");
System.out.println("-----");
}
};
class Developer extends Person
{
public void designProj()
{
System.out.println("Design the project");
}
void sleep(String name)
{
super.name=name;
System.out.println(super.name +"is sleeping
4hr/day");
System.out.println("-----");
}
};
class OverrideDemo
{
public static void main(String[] args)
{
Student s1=new Student();
s1.writExams();
}
}

```



```

s1.sleep("student");
s1.walk();
Developer d1=new Developer();
d1.designProj();
d1.sleep("developer");
}
}

```

Difference between Overloading and Overriding

	Overloading	Overriding
1	Whenever same method or Constructor is existing multiple times within a class either with different number of parameter or with different type of parameter or with different order of parameter is known as Overloading.	Whenever same method name is existing multiple time in both base and derived class with same number of parameter or same type of parameter or same order of parameters is known as Overriding.
2	Arguments of method must be different at least arguments.	Argument of method must be same including order.
3	Method signature must be different.	Method signature must be same.
4	Private, static and final methods can be overloaded.	Private, static and final methods can not be override.
5	Access modifiers point of view no restriction.	Access modifiers point of view not reduced scope of Access modifiers but increased.

6	Also known as compile time polymorphism or static polymorphism or early binding.	Also known as run time polymorphism or dynamic polymorphism or late binding.
7	Overloading can be exhibited both are method and constructor level.	Overriding can be exhibited only at method level.
8	The scope of overloading is within the class.	The scope of Overriding is base class and derived class.
9	Overloading can be done at both static and non-static methods.	Overriding can be done only at non-static method.
10	For overloading methods return type may or may not be same.	For overriding method return type should be same.

Note: In overloading we have to check only methods names (must be same) and arguments types (must be different) except these the remaining like return type access modifiers etc. are not required to check

But in overriding every things check like method names arguments types return types access modifiers etc.

Interface in Java

Interface is similar to class which is collection of public static final variables (constants) and abstract methods.

The interface is a mechanism to achieve fully abstraction in java. There can be only abstract methods in the interface. It is used to achieve fully abstraction and multiple inheritance in Java.

properties of Interface

- It is implicitly abstract. So we no need to use the abstract keyword when declaring an interface.
- Each method in an interface is also implicitly abstract, so the abstract keyword is not needed.
- Methods in an interface are implicitly public.
- All the data members of interface are implicitly public static final.

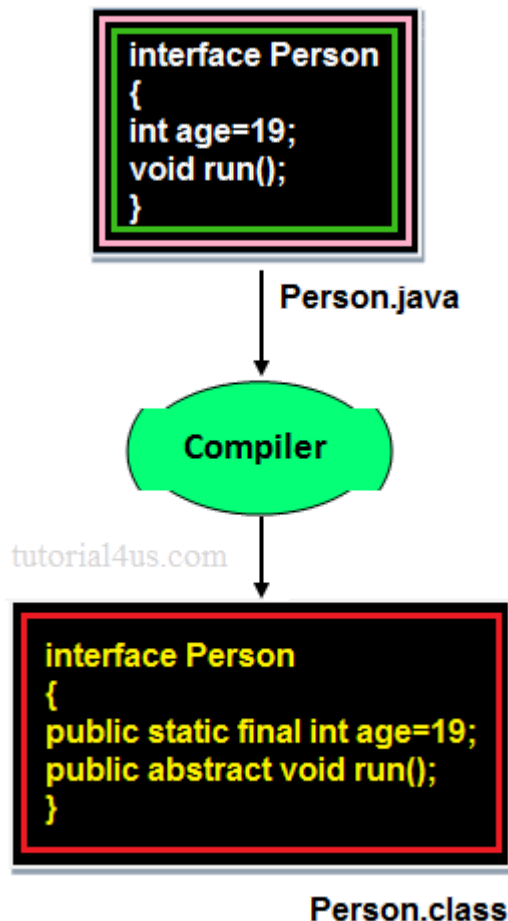
How interface is similar to class ?

Whenever we compile any Interface program it generate .class file. That means the bytecode of an interface appears in a .class file.

How interface is different from class ?

- You can not instantiate an interface.
- It does not contain any constructors.
- All methods in an interface are abstract.
- Interface can not contain instance fields. Interface only contains public static final variables.
- Interface is can not extended by a class; it is implemented by a class.
- Interface can extend multiple interfaces. It means interface support multiple inheritance

Behavior of compiler with Interface program



In the above image when we compile any interface program, by default compiler added public static final before any variable and public abstract before any method. Because **Interface** is design for fulfill universal requirements and to achieve fully abstraction.

Declaring Interfaces:

The **interface** keyword is used to declare an interface.

Example

```
interface Person
{
    datatype variablename=value;
    //Any number of final, static fields
    returntype methodname(list of parameters
    or no parameters)
```

```
//Any number of abstract method  
declarations  
}
```

Explanations

In the above syntax **Interface** is a keyword interface name can be user defined name the default signature of variable is public static final and for method is public abstract. JVM will be added implicitly public static final before data members and public abstract before method.

Example

```
public static final datatype variable  
name=value; ----> for data member  
public abstract returntype  
methodname(parameters) ---> for method
```

Implementing Interfaces:

A class uses the implements keyword to implement an interface. The implements keyword appears in the class declaration following the extends portion of the declaration.

Example

```
interface Person  
{  
void run();  
}  
class Employee implements Person  
{  
public void run()  
{  
System.out.println("Run fast");  
}}
```

}

When we use abstract and when Interface

If we do not know about any things about implementation just we have requirement specification then we should be go for **Interface**

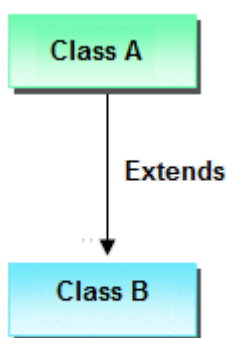
If we are talking about implementation but not completely (partially implemented) then we should be go for **abstract**

Rules for implementation interface

- A class can implement more than one interface at a time.
- A class can extend only one class, but implement many interfaces.
- An interface can extend another interface, similarly to the way that a class can extend another class.

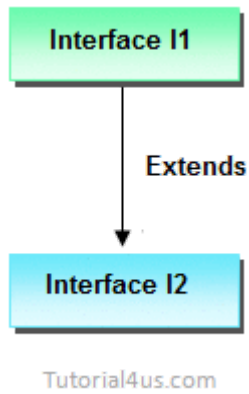
Relationship between class and Interface

- Any class can extends another class

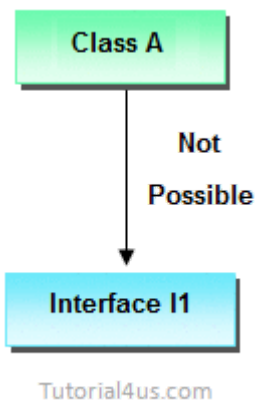


Tutorial4us.com

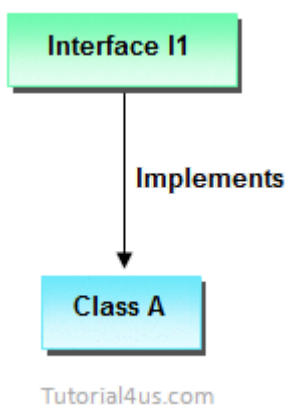
- Any Interface can extends another Interface.



- Any Interface can not extend or Implements any class.



- Any class can Implements another Interface



Difference between Abstract class and Interface	
Abstract class	Interface

1	It is collection of abstract method and concrete methods.	It is collection of abstract method.
2	There properties can be reused commonly in a specific application.	There properties commonly usable in any application of java environment.
3	It does not support multiple inheritance.	It support multiple inheritance.
4	Abstract class is preceded by abstract keyword.	It is preceded by Interface keyword.
5	Which may contain either variable or constants.	Which should contains only constants.
6	The default access specifier of abstract class methods are default.	There default access specifier of interface method are public.
7	These class properties can be reused in other class using extend keyword.	These properties can be reused in any other class using implements keyword.
8	Inside abstract class we can take constructor.	Inside interface we can not take any constructor.
9	For the abstract class there is no restriction like initialization of variable at	For the interface it should be compulsory to initialization of

	the time of variable declaration.	variable at the time of variable declaration.
10	There are no any restriction for abstract class variable.	For the interface variable can not declare variable as private, protected, transient, volatile.
11	There are no any restriction for abstract class method modifier that means we can use any modifiers.	For the interface method can not declare method as strictfp, protected, static, native, private, final, synchronized.

Example of Interface

```
interface Person
{
void run();    // abstract method
}
class A implements Person
{
public void run()
{
System.out.println("Run fast");
}
public static void main(String args[])
{
A obj = new A();
obj.run();
}
}
```

Output

Run fast

Multiple Inheritance using interface

Example

```
interface Developer
{
void disp();
}
interface Manager
{
void show();
}

class Employee implements Developer,
Manager
{
public void disp()
{
System.out.println("Hello Good Morning");
}
public void show()
{
System.out.println("How are you ?");
}
public static void main(String args[])
{
Employee obj=new Employee();
obj.disp();
obj.show();
}
}
```

Output

```
Hello Good Morning
How are you ?
```

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.

Example

```
//Way of writing Serializable interface
public interface Serializable
{
}
```

Why interface have no constructor ?

Because, constructor are used for eliminate the default values by user defined values, but in case of interface all the data members are public static final that means all are constant so no need to eliminate these values.

Other reason because constructor is like a method and it is concrete method and interface does not have concrete method it have only abstract methods that's why interface have no constructor.

Abstraction in Java

Abstraction is the concept of exposing only the required essential characteristics and behavior with respect to a context.

Hiding of data is known as **data abstraction**. In object oriented programming language this is implemented automatically while writing the code in the form of class and object.

Real Life Example of Abstraction in Java

Abstraction shows only important things to the user and hides the internal details, for example, when we ride a bike, we only know about how to ride bikes but can not know about how it work? And also we do not know the internal functionality of a bike.

Data abstraction can be used to provide security for the data from the unauthorized methods.

Note: In Java language data abstraction can achieve using class.

Example of Abstraction

```
class Customer
{
int account_no;
float balance_Amt;
String name;
int age;
String address;
void balance_inquiry()
{
/* to perform balance inquiry only account
number
is required that means remaining properties
are hidden for balance inquiry method */
}
void fund_Transfer()
{
/* To transfer the fund account number and
balance is required and remaining
properties
are hidden for fund transfer method */
}
```

How to achieve Abstraction ?

There are two ways to achieve abstraction in java

- Abstract class (0 to 100%)
- Interface (Achieve 100% abstraction)

Difference between Encapsulation and Abstraction

Encapsulation is not providing full security because we can access private member of the class using reflection API, but in case of Abstraction we can't access static, abstract data member of a class.

Encapsulation in Java

Encapsulation is a process of wrapping of data and methods in a single unit is called encapsulation. Encapsulation is achieved in java language by class concept.

Combining of state and behavior in a single container is known as encapsulation. In java language encapsulation can be achieve using **class** keyword, state represents declaration of variables on attributes and behavior represents operations in terms of method.

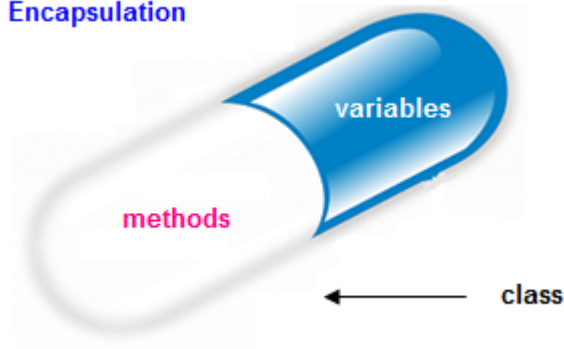
Advantage of Encapsulation

The main advantage of using of encapsulation is to secure the data from other methods, when we make a data private then these data only use within the class, but these data not accessible outside the class.

Real life example of Encapsulation in Java

The common example of encapsulation is capsule. In capsule all medicine are encapsulated in side capsule.

Encapsulation



Benefits of encapsulation

- Provides abstraction between an object and its clients.
- Protects an object from unwanted access by clients.
- Example: A bank application forbids (restrict) a client to change an Account's balance.

Let's see the Example of Encapsulation in java

Example

```
class Employee
{
    private String name;

    public String getName()
    {
        return name;
    }

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

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

```
Employee e=new Employee();  
e.setName("Harry");  
System.out.println(e.getName());  
}  
}
```

Output

Harry

Polymorphism in Java

The process of representing one form in multiple forms is known as **Polymorphism**.

Polymorphism is derived from 2 greek words: **poly** and **morphs**. The word "poly" means many and "morphs" means forms. So polymorphism means many forms.

Polymorphism is not a programming concept but it is one of the principal of OOPs. For many objects oriented programming language polymorphism principle is common but whose implementations are varying from one objects oriented programming language to another object oriented programming language.

How to achieve Polymorphism in Java ?

In java programming the Polymorphism principal is implemented with method overriding concept of java.

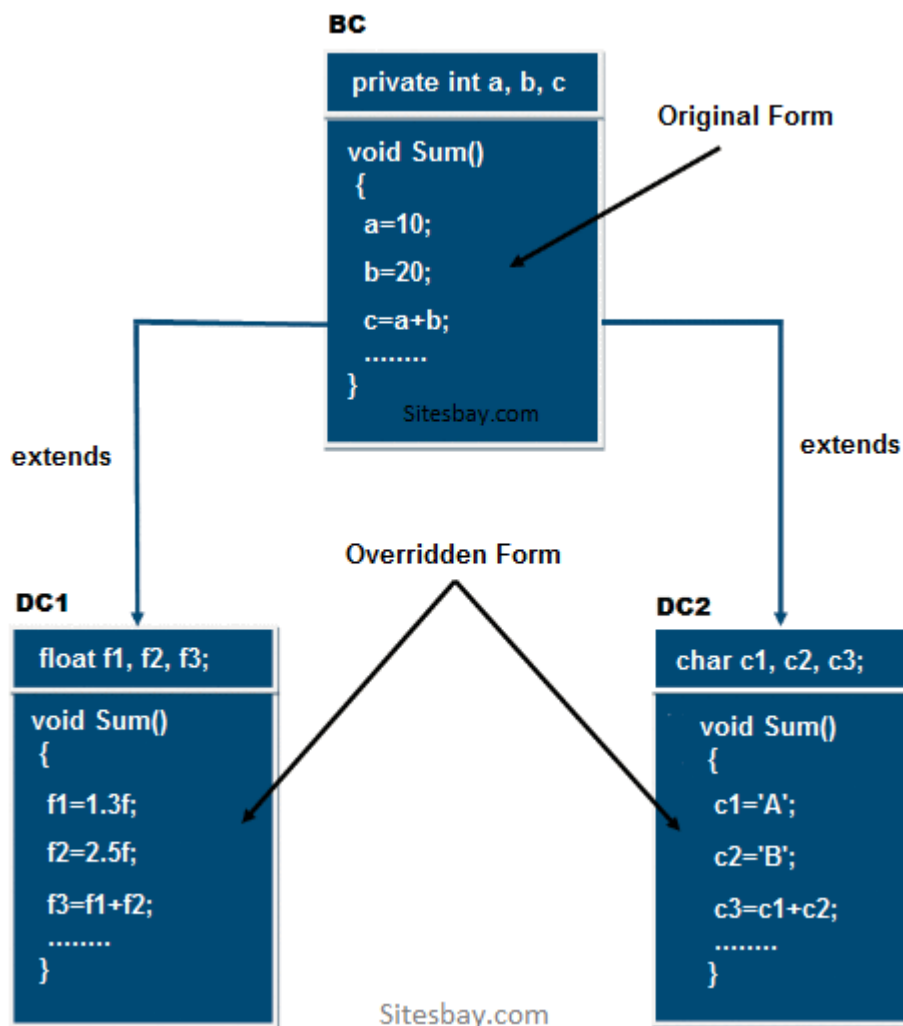
Polymorphism principal is divided into two sub principal they are:

- Static or Compile time polymorphism
- Dynamic or Runtime polymorphism

Note: Java programming does not support static polymorphism because of its limitations and java always supports dynamic polymorphism.

Let us consider the following diagram

Here original form or original method always resides in base class and multiple forms represents overridden method which resides in derived classes.



In the above diagram the sum method which is present in BC class is called original form and the sum() method which are present in DC1 and DC2 are called overridden form hence Sum() method is originally available in only one form and it is further implemented in multiple forms. Hence Sum() method is one of the polymorphism method.

Example of Runtime Polymorphism in Java

In below example we create two class Person an Employee, Employee class extends Person class feature and override walk() method. We are calling the walk() 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. Here method invocation is determined by the JVM not compiler, So it is known as runtime polymorphism.

Example of Polymorphism in Java

```
class Person
{
void walk()
{
System.out.println("Can Run....");
}
}
class Employee extends Person
{
void walk()
{
System.out.println("Running Fast...");
}
public static void main(String arg[])
{
Person p=new Employee(); //upcasting
p.walk();
}
}
```

Output

Running fast...

Dynamic Binding

Dynamic binding always says create an object of base class but do not create the object of derived classes. Dynamic binding principal is always used for executing polymorphic applications.

The process of binding appropriate versions (overridden method) of derived classes which are inherited from base class with base class object is known as dynamic binding.

Advantages of dynamic binding along with polymorphism with method overriding are.

- Less memory space
- Less execution time
- More performance

Static polymorphism

The process of binding the overloaded method within object at compile time is known as **Static polymorphism** due to static polymorphism utilization of resources (main memory space) is poor because for each and every overloaded method a memory space is created at compile time when it binds with an object. In C++ environment the above problem can be solve by using dynamic polymorphism by implementing with virtual and pure virtual function so most of the C++ developer in real worlds follows only dynamic polymorphism.

Dynamic polymorphism

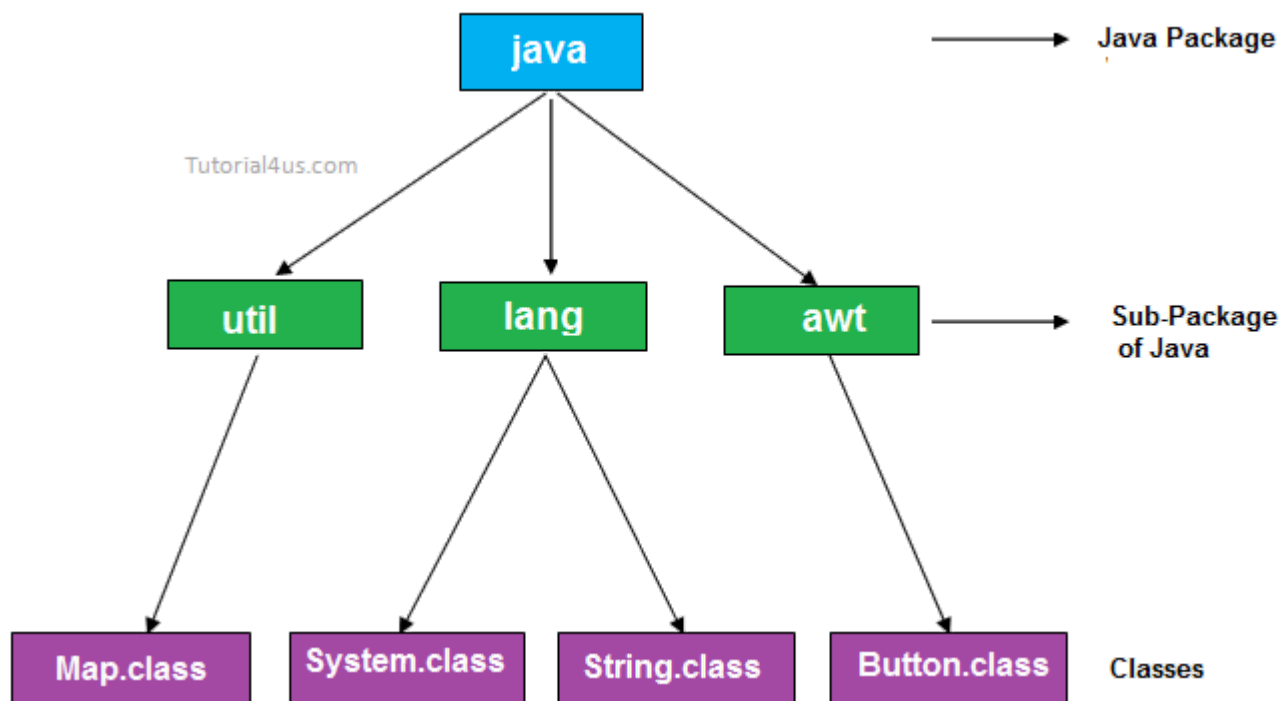
In dynamic polymorphism method of the program binds with an object at runtime the advantage of dynamic polymorphism is allocating the memory space for the method (either for overloaded method or for override method) at run time.

Package in Java

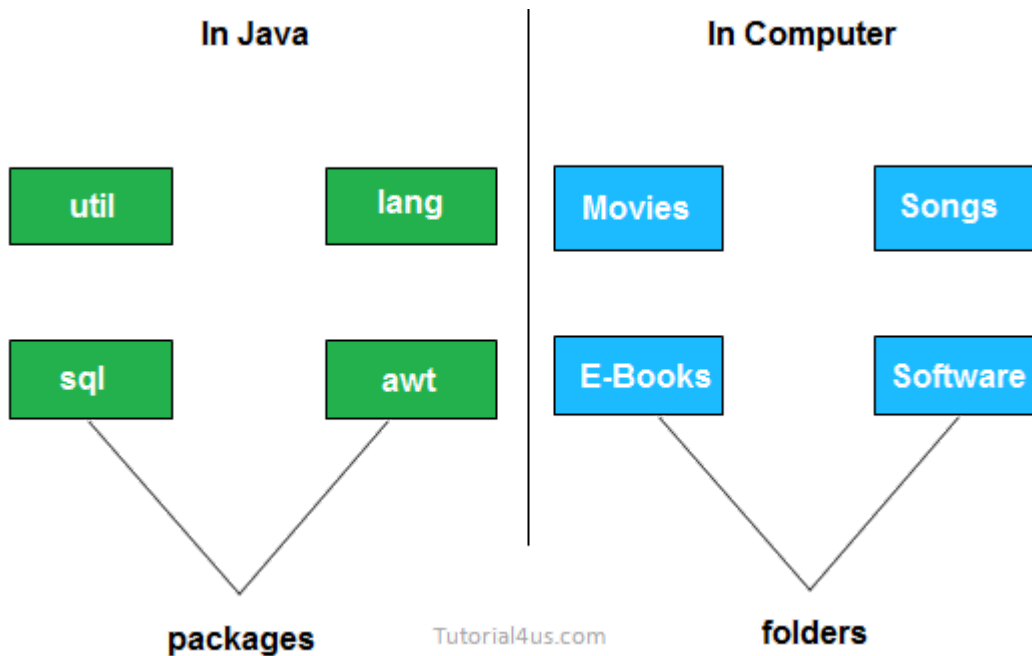
A package is a collection of similar types of classes, interfaces and sub-packages.

Purpose of package

The purpose of package concept is to provide common classes and interfaces for any program separately. In other words if we want to develop any class or interface which is common for most of the java programs than such common classes and interfaces must be place in a package.



Packages in Java are the way to organize files when a project has many modules. Same like we organized our files in Computer. For example we store all movies in one folder and songs in other folder, here also we store same type of files in a particular package for example in awt package have all classes and interfaces for design GUI components.



Advantage of package

- Package is used to categorize the classes and interfaces so that they can be easily maintained
- Application development time is less, because reuse the code
- Application memory space is less (main memory)
- Application execution time is less
- Application performance is enhance (improve)
- Redundancy (repetition) of code is minimized
- Package provides access protection.
- Package removes naming collision.

Type of package

Package are classified into two type which are given below.

1. Predefined or built-in package
2. User defined package

Predefined or built-in package

These are the package which are already designed by the Sun Microsystem and supply as a part of java API, every predefined package is collection of predefined classes, interfaces and sub-package.

User defined package

If any package is design by the user is known as user defined package. User defined package are those which are developed by java programmer and supply as a part of their project to deal with common requirement.

Rules to create user defined package

- package statement should be the first statement of any package program.
- Choose an appropriate class name or interface name and whose modifier must be public.
- Any package program can contain only one public class or only one public interface but it can contain any number of normal classes.
- Package program should not contain any main class (that means it should not contain any main())
- modifier of constructor of the class which is present in the package must be public. (This is not applicable in case of interface because interface have no constructor.)
- The modifier of method of class or interface which is present in the package must be public (This rule is optional in case of interface because interface methods by default public)

- Every package program should be save either with public class name or public Interface name

<code>//Sum.java</code>	→	Save package program with 'public' class name
<code>package MyPackage</code>	→	First statement of java program is package
<code>public class Sum</code>	→	class modifier must be 'public'
<code>{</code>		
<code>public Sum()</code>	→	constructor modifier must be 'public'
<code>{</code>		
<code>System.out.println("Sum class constructor");</code>		
<code>}</code>		
<code>public void show()</code>	→	method modifier must be 'public'
<code>{</code>		
<code>System.out.println("Sum class method");</code>		
<code>}</code>		
<code>}</code>		

Tutorial4us.com

Compile package programs

For compilation of package program first we save program with public className.java and it compile using below syntax:

Syntax

```
javac -d . className.java
```

Syntax

```
javac -d path className.java
```

Explanations: In above syntax "-d" is a specific tool which is tell to java compiler create a separate folder for the given package in given path. When we give specific path then it create a new folder at that location and when we use . (dot) then it crate a folder at current working directory.

Note: Any package program can be compile but can not be execute or run. These program can be executed through user defined program which are importing package program.

Example of package program

Package program which is save with A.java and compile by
javac -d . A.java

Example

```
package mypack;  
public class A  
{  
    public void show()  
    {  
        System.out.println("Sum method");  
    }  
}
```

Import above class in below program using import
packageName.className

Example

```
import mypack.A;  
public class Hello  
{  
    public static void main(String arg[])  
    {  
        A a=new A();  
        a.show();  
        System.out.println("show() class A");  
    }  
}
```

Explanations: In the above program first we create Package program which is save with A.java and compiled by "**javac -d . A.java**". Again we import class "A" in class Hello using "**import mypack.A;**" statement.

How to access package from another package?

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

- `import package.*;`
- `import package.classname;`
- 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.

Difference between Inheritance and package

Inheritance concept always used to reuse the feature within the program between class to class, interface to interface and interface to class but not accessing the feature across the program.

Package concept is to reuse the feature both within the program and across the programs between class to class, interface to interface and interface to class.

Difference between package keyword and import keyword

Package keyword is always used for creating the undefined package and placing common classes and interfaces.

import is a keyword which is used for referring or using the classes and interfaces of a specific package.

Exception Handling in Java

The process of converting system error messages into user friendly error message is known as **Exception handling**. This is one of the powerful feature of Java to handle run time error and maintain normal flow of java application.

Exception

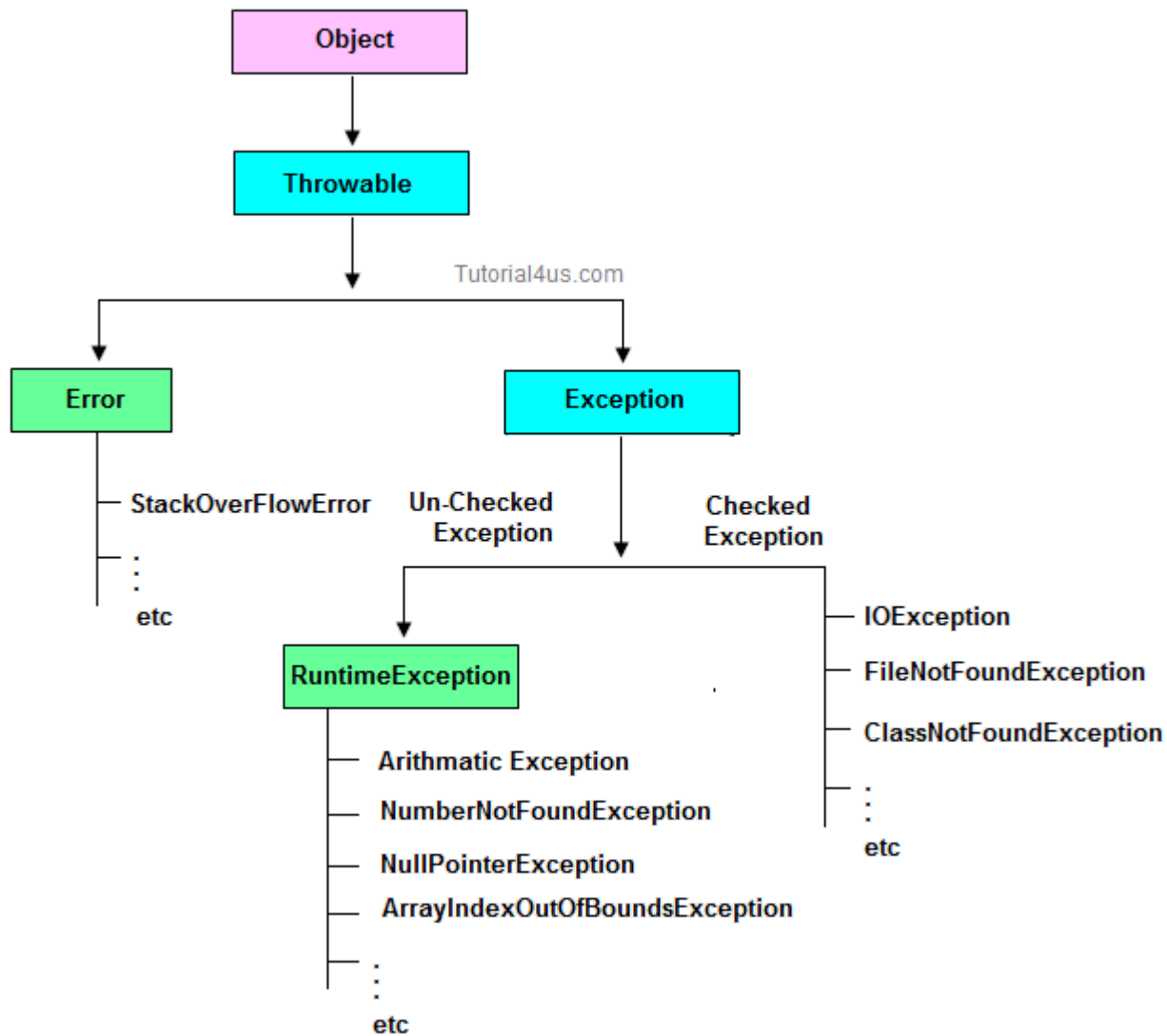
An **Exception** is an event, which occurs during the execution of a program, that disrupts the normal flow of the program's Instructions.

Why use Exception Handling

Handling the exception is nothing but converting system error generated message into user friendly error message. Whenever an exception occurs in the java application, JVM will create an object of appropriate exception of sub class and generates system error message, these system generated messages are not understandable by user so need to convert it into user friendly error message. You can convert system error message into user friendly error message by using exception handling feature of java.

For Example: when you divide any number by zero then system generate / **by zero** so this is not understandable by user so you can convert this message into user friendly error message like **Don't enter zero for denominator.**

Hierarchy of Exception classes



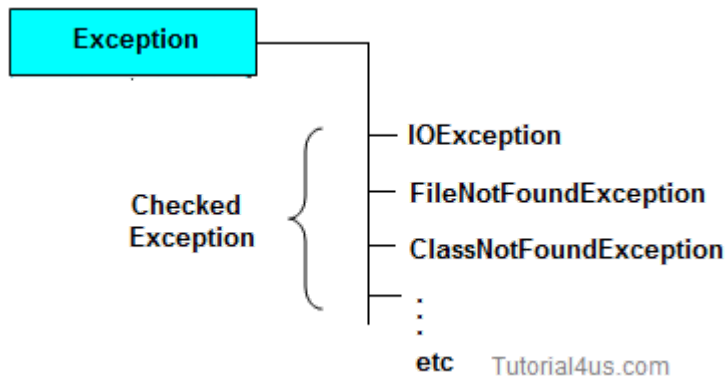
Type of Exception

- Checked Exception
- Un-Checked Exception

Checked Exception

Checked Exception are the exception which checked at compile-time. These exception are directly sub-class of `java.lang.Exception` class.

Only for remember: Checked means checked by compiler so checked exception are checked at compile-time.

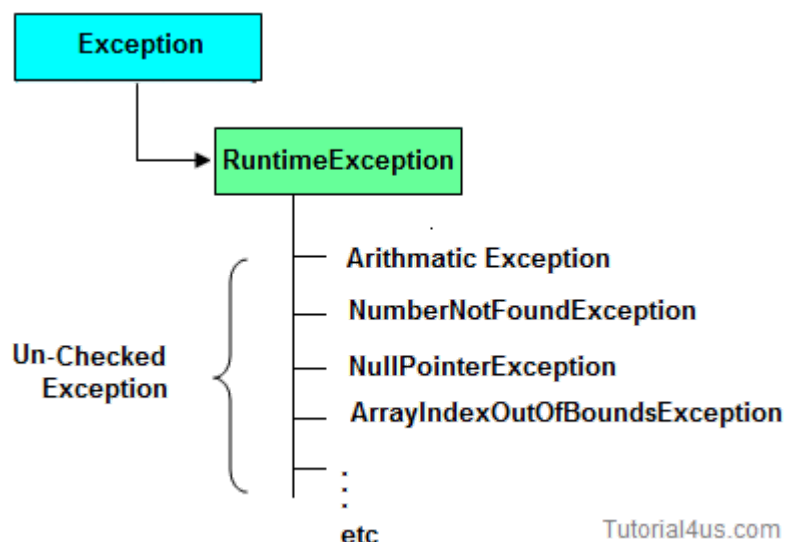


Un-Checked Exception

Un-Checked Exception are the exception both identifies or raised at run time. These exception are directly sub-class of `java.lang.RuntimeException` class.

Note: In real time application mostly we can handle un-checked exception.

Only for remember: Un-checked means not checked by compiler so un-checked exception are checked at run-time not compile time.



Difference between checked Exception and un-checked Exception

Checked Exception

Un-Checked Exception

1	checked Exception are checked at compile time	un-checked Exception are checked at run time
	e.g. FileNotFoundException, 3 NumberNotFoundException etc.	e.g. ArithmeticException, NullPointerException, ArrayIndexOutOfBoundsException etc.

Difference between Error and Exception

Error	Exception
1 Can't be handle.	Can be handle.
Example:	Example:
2 NoSuchMethodError	ClassNotFoundException
OutOfMemoryError	NumberFormatException

Handling the Exception

Handling the exception is nothing but converting system error generated message into user friendly error message in others word whenever an exception occurs in the java application, JVM will create an object of appropriate exception of sub class and generates system error message, these system generated messages are not understandable by user so need to convert it into user-friendly error message. You can convert system error message into user-friendly error message by using exception handling feature of java.

Use Five keywords for Handling the Exception

- try

- catch
- finally
- throws
- throw

Syntax for handling the exception

Syntax

```
try
{
    // statements causes problem at run time
}
catch(type of exception-1 object-1)
{
    // statements provides user friendly
    error message
}
catch(type of exception-2 object-2)
{
    // statements provides user friendly
    error message
}
finally
{
    // statements which will execute
    compulsory
}
```

Example without Exception Handling

Syntax

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

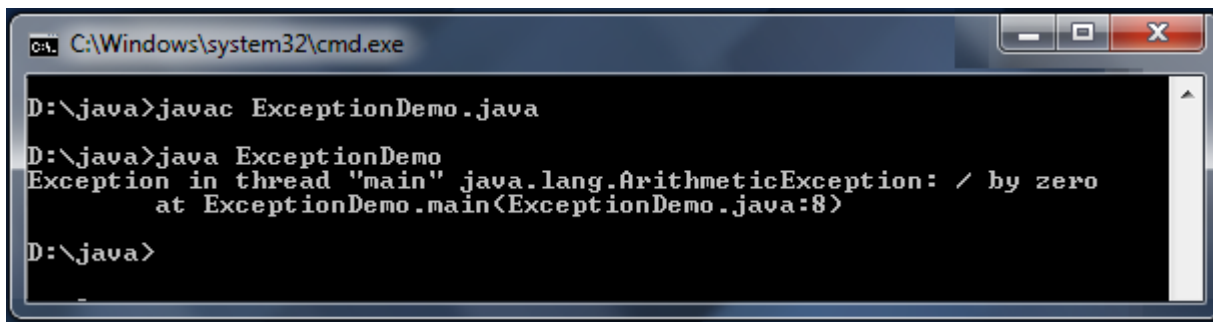


```

{
int a=10, ans=0;
ans=a/0;
System.out.println("Denominator not be
zero");
}
}

```

Abnormally terminate program and give a message like below, this error message is not understandable by user so we convert this error message into user friendly error message, like "denominator not be zero".



```

C:\Windows\system32\cmd.exe

D:\java>javac ExceptionDemo.java

D:\java>java ExceptionDemo
Exception in thread "main" java.lang.ArithmeticException: / by zero
    at ExceptionDemo.main(ExceptionDemo.java:8)

D:\java>

```

Example of Exception Handling

Example

```

class ExceptionDemo
{
public static void main(String[] args)
{
int a=10, ans=0;
try
{
ans=a/0;
}
catch (Exception e)
{

```

```
System.out.println("Denominator not be  
zero");  
}  
}  
}
```

Output

Denominator not be zero

try and catch block

try block

Inside **try block** we write the block of statements which causes executions at run time in other words try block always contains problematic statements.

Important points about try block

- If any exception occurs in try block then CPU controls comes out to the try block and executes appropriate catch block.
- After executing appropriate catch block, even through we use run time statement, CPU control never goes to try block to execute the rest of the statements.
- Each and every try block must be immediately followed by catch block that is no intermediate statements are allowed between try and catch block.

Syntax

```
try  
{  
    . . . . .  
}  
/* Here no other statements are allowed  
between try and catch block */
```

```
catch ()
{
    . . . . .
}
```

- Each and every try block must contains at least one catch block. But it is highly recommended to write multiple catch blocks for generating multiple user friendly error messages.
- One try block can contains another try block that is nested or inner try block can be possible.

Syntax

```
try
{
    . . . . .
    try
    {
        . . . . .
    }
}
```

catch block

Inside **catch** block we write the block of statements which will generates user friendly error messages.

catch block important points

- Catch block will execute exception occurs in try block.
- You can write multiple catch blocks for generating multiple user friendly error messages to make your application strong. You can see below example.
- At a time only one catch block will execute out of multiple catch blocks.

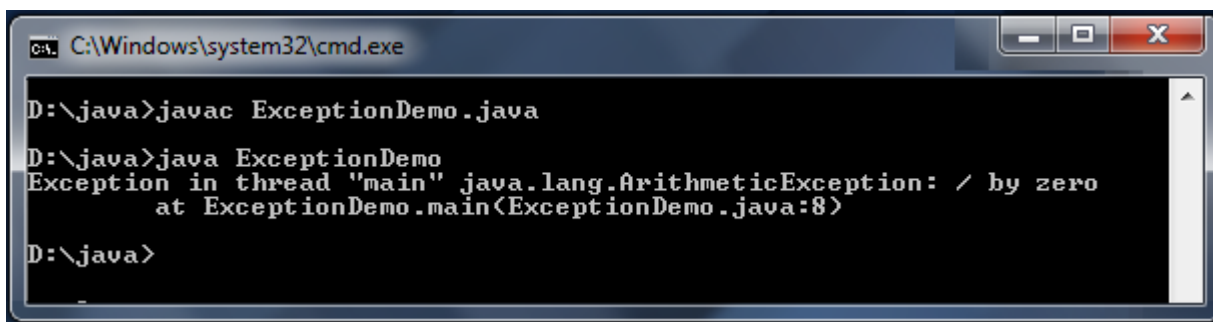
- in catch block you declare an object of sub class and it will be internally referenced by JVM.

Example without Exception Handling

Example

```
class ExceptionDemo
{
public static void main(String[] args)
{
int a=10, ans=0;
ans=a/0;
System.out.println("Denominator not be
zero");
}
}
```

Abnormally terminate program and give a message like below, this error message is not understandable by user so we convert this error message into user friendly error message, like "denominator not be zero".



Example of Exception Handling

Example

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

```
int a=10, ans=0;
try
{
ans=a/0;
}
catch (Exception e)
{
System.out.println("Denominator not be
zero");
}
}
}
```

Output

Denominator not be zero

Multiple catch block

You can write multiple catch blocks for generating multiple user friendly error messages to make your application strong. You can see below example.

Example

```
import java.util.*;
class ExceptionDemo
{
public static void main(String[] args)
{
int a, b, ans=0;
Scanner s=new Scanner(System.in);
System.out.println("Enter any two numbers:
");
try
{
a=s.nextInt();
```

```

        b=s.nextInt();
        ans=a/b;
        System.out.println("Result: "+ans);
    }
    catch (ArithmeticException ae)
    {
        System.out.println("Denominator not be
        zero");
    }
    catch (Exception e)
    {
        System.out.println("Enter valid number");
    }
}

```

Output

```

Enter any two number: 5 0
Denominator not be zero

```

finally Block in Exception Handling

Inside **finally** block we write the block of statements which will relinquish (released or close or terminate) the resource (file or database) where data store permanently.

finally block important points

- Finally block will execute compulsory
- Writing finally block is optional.
- You can write finally block for the entire java program
- In some of the circumstances one can also write try and catch block in finally block.

Example

```
class ExceptionDemo
{
public static void main(String[] args)
{
int a=10, ans=0;
try
{
ans=a/0;
}
catch (Exception e)
{
System.out.println("Denominator not be
zero");
}
finally
{
System.out.println("I am from finally
block");
}
}
}
```

Output

```
Denominator not be zero
I am from finally block
```

Exception Classes in Java

Exception are mainly classified into two type checked exception and un-checked exception.

Checked Exception Classes

- FileNotFoundException

- `ClassNotFoundException`
- `IOException`
- `InterruptedException`

Un-Checked Exception Classes

- `ArithmeticException`
- `ArrayIndexOutOfBoundsException`
- `StringIndexOutOfBoundsException`
- `NumberFormatException`
- `NullPointerException`
- `NoSuchMethodException`
- `NoSuchFieldException`

FileNotFoundException

If the given filename is not available in a specific location (in file handling concept) then `FileNotFoundException` will be raised. This exception will be thrown by the `FileInputStream`, `FileOutputStream`, and `RandomAccessFile` constructors.

ClassNotFoundException

If the given class name is not existing at the time of compilation or running of program then `ClassNotFoundException` will be raised. In other words this exception is occurred when an application tries to load a class but no definition for the specified class name could be found.

IOException

This is exception is raised whenever problem occurred while writing and reading the data in the file. This exception is occurred due to following reason;

- When try to transfer more data but less data are present.

- When try to read data which is corrupted.
- When try to write on file but file is read only.

InterruptedException

This exception is raised whenever one thread is disturb the other thread. In other words this exception is thrown when a thread is waiting, sleeping, or otherwise occupied, and the thread is interrupted, either before or during the activity.

ArithmeticException

This exception is raised because of problem in arithmetic operation like divide by zero. In other words this exception is thrown when an exceptional arithmetic condition has occurred. For example, an integer "divide by zero".

Example

```
class ExceptionDemo
{
public static void main(String[] args)
{
int a=10, ans=0;
try
{
ans=a/0;
}
catch (Exception e)
{
System.out.println("Denominator not be
zero");
}
}
}
```

ArrayIndexOutOfBoundsException

This exception will be raised whenever given index value of an array is out of range. The index is either negative or greater than or equal to the size of the array.

Example

```
int a[]=new int[5];  
a[10]=100; //ArrayIndexOutOfBoundsException
```

StringIndexOutOfBoundsException

This exception will be raised whenever given index value of string is out of range. The index is either negative or greater than or equal to the size of the array.

Example

```
String s="Hello";  
s.charAt(3);  
s.charAt(10); // Exception raised
```

charAt() is a predefined method of string class used to get the individual characters based on index value.

NumberFormatException

This exception will be raised whenever you trying to store any input value in the un-authorized datatype.

Example: Storing string value into int datatype.

Example

```
int a;  
a="Hello";
```

Example

```
String s="hello";
```

```
int  
i=Integer.parseInt(s); //NumberFormatException  
on
```

NoSuchMethodException

This exception will be raised whenever calling method is not existing in the program.

NullPointerException

A NullPointerException is thrown when an application is trying to use or access an object whose reference equals to null.

Example

```
String s=null;  
System.out.println(s.length()); //NullPointerException
```

StackOverflowException

This exception throw when full the stack because the recursion method are stored in stack area.

Difference Between Throw and Throws Keyword

throw

throw is a keyword in java language which is used to throw any user defined exception to the same signature of method in which the exception is raised.

Note: throw keyword always should exist within method body.

whenever method body contain throw keyword than the call method should be followed by throws keyword.

Syntax

```
class className
{
    returnType method(...) throws
    Exception_class
    {
        throw (Exception obj)
    }
}
```

throws

throws is a keyword in java language which is used to throw the exception which is raised in the called method to it's calling method throws keyword always followed by method signature.

Example

```
returnType methodName (parameter) throws
Exception_class....
{
    ....
}
```

Difference between throw and throws

throw

throw is a keyword used for hitting and generating the exception which are occurring as a part of method body

throws

throws is a keyword which gives an indication to the specific method to place the common exception methods as a part of try and catch block for generating user friendly error messages

The place of using throw
2 keyword is always as a part
of method body.

The place of using throws is a
keyword is always as a part of
method heading

When we use throw keyword
as a part of method body, it
is mandatory to the java
3 programmer to write throws
keyword as a part of method
heading

When we write throws keyword
as a part of method heading, it is
optional to the java programmer
to write throw keyword as a part
of method body.

Example of throw and throws

Example

```
// save by DivZero.java
```

```
package pack;
```

```
public class DivZero
{
    public void division(int a, int b) throws
    ArithmeticException
    {
        if (b==0)
        {
            ArithmeticException ae=new
            ArithmeticException("Does not enter zero
            for Denominator");
            throw ae;
        }
        else
        {
```

```
int c=a/b;
System.out.println("Result: "+c);
}
}
}
```

Compile: javac -d . DivZero.java

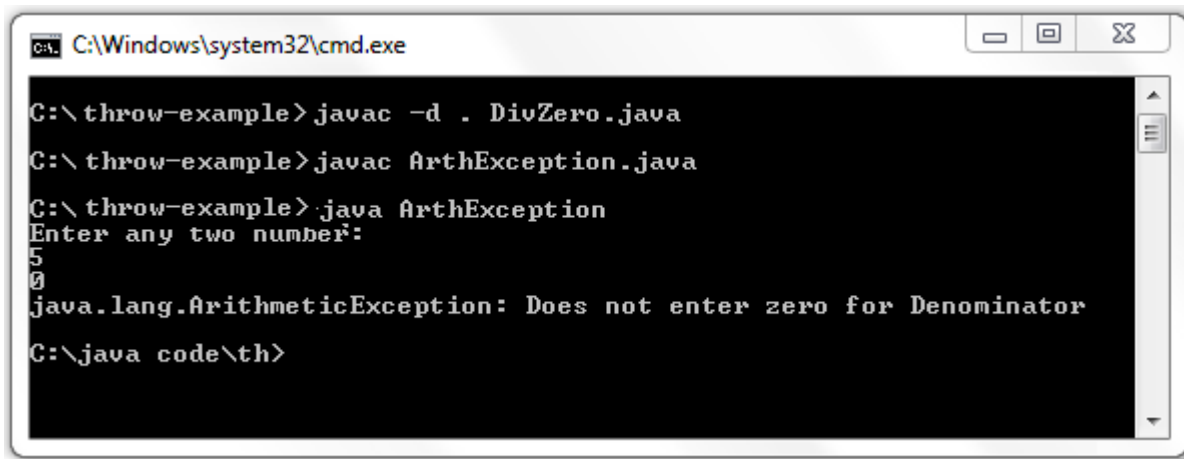
Example

// save by ArthException.java

```
import pack.DivZero;
import java.util.*;

class ArthException
{
public static void main(String args[])
{
System.out.println("Enter any two number:
");
Scanner s=new Scanner(System.in);
try
{
int a=s.nextInt();
int b=s.nextInt();
DivZero dz=new DivZero();
dz.division(a, b);
}
catch(Exception e)
{
System.err.println(e);
}
}
}
```

Compile: javac ArthException.java



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

C:\throw-example>javac -d . DivZero.java
C:\throw-example>javac ArthException.java
C:\throw-example>.java ArthException
Enter any two number:
5
0
java.lang.ArithmeticException: Does not enter zero for Denominator
C:\java code\th>
```

Download Code [Click](#)

Steps to Compile and Run code

First you save throw-example files into you PC in any where, here i will save this file in C:\>

- C:\throw-example\>javac -d . DivZero.java
- C:\throw-example\>javac ArthException.java

Note: First compile DivZero.java code then compile ArthException.java code.

Custom Exception in Java

If any exception is design by the user known as user defined or Custom Exception. Custom Exception is created by user.

Rules to design user defined Exception

1. Create a package with valid user defined name.
2. Create any user defined class.
3. Make that user defined class as derived class of Exception or RuntimeException class.
4. Declare parametrized constructor with string variable.

5. call super class constructor by passing string variable within the derived class constructor.
6. Save the program with public class name.java

```
// AgeException.java → ⑥
package pack; → ①

public class AgeException extends Exception
{
    ↓ ②           ↓ ③
    public AgeException(String s) → ④
    {
        super(s); → ⑤
    }
}
```

Tutorial4us.com

Example

```
// save by AgeException.java
package nage;
```

```
public class AgeException extends Exception
{
    public AgeException(String s)
    {
        super(s);
    }
}
```

Compile: javac -d . AgeException.java

Example

```
// save by CheckAge.java
package nage;
```

```
public class CheckAge
{
```



```

public void verify(int age)throws
AgeException
{
if (age>0)
{
System.err.print("valid age");
}
else
{
AgeException ae=new AgeException("Invalid
age");
throw(ae);
}
}
}

```

Compile: javac -d . CheckAge.java

Example

```

// save by VerifyAgeException

import nage.AgeException;
import nage.CheckAge;
import java.util.*;

public class VerifyAgeException
{
public static void main(String args[])
{
int a;
System.out.println("Enter your age");
Scanner s=new Scanner(System.in);
a=s.nextInt();
try

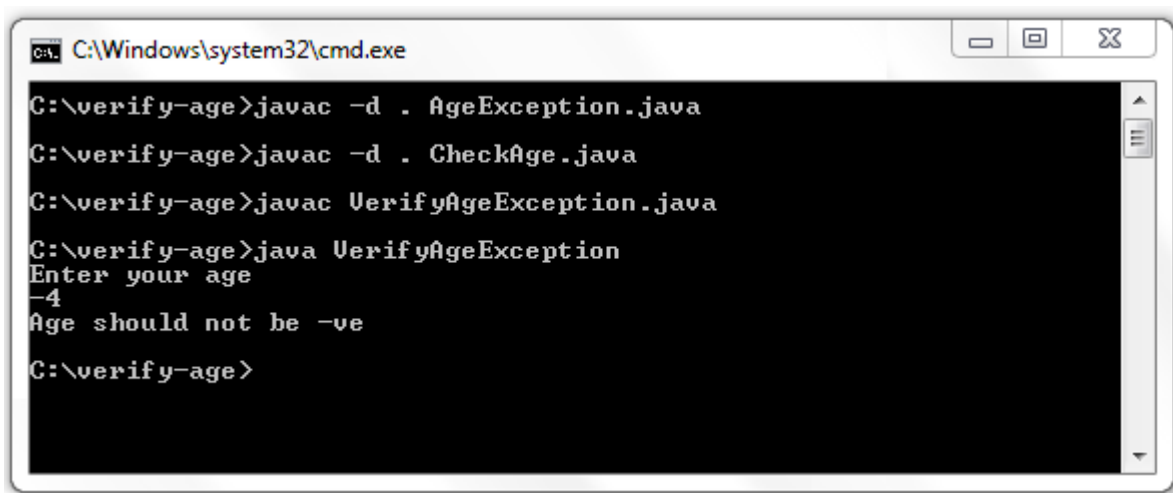
```

```

{
CheckAge ca=new CheckAge();
ca.verify(a);
}
catch(AgeException ae)
{
System.err.println("Age should not be
-ve");
}
catch(Exception e)
{
System.err.println(e);
}
}
}

```

Compile: javac VerifyAgeException.java



```

C:\Windows\system32\cmd.exe

C:\verify-age>javac -d . AgeException.java
C:\verify-age>javac -d . CheckAge.java
C:\verify-age>javac VerifyAgeException.java
C:\verify-age>java VerifyAgeException
Enter your age
-4
Age should not be -ve
C:\verify-age>

```

Steps to compile and run above program

First you save verify-age files into you PC in any where, here i will save this file in C:\>

- C:\verify-age\>javac -d . AgeException.java
- C:\verify-age\>javac -d . CheckAge.java
- C:\verify-age\>javac VerifyAgeException.java

Note: First compile AgeException.java code then CheckAge.java and at last compile VerifyAgeException.java code.

Multithreading in Java

Multithreading in java is a process of executing multiple threads simultaneously. The aim of multithreading is to achieve the concurrent execution.

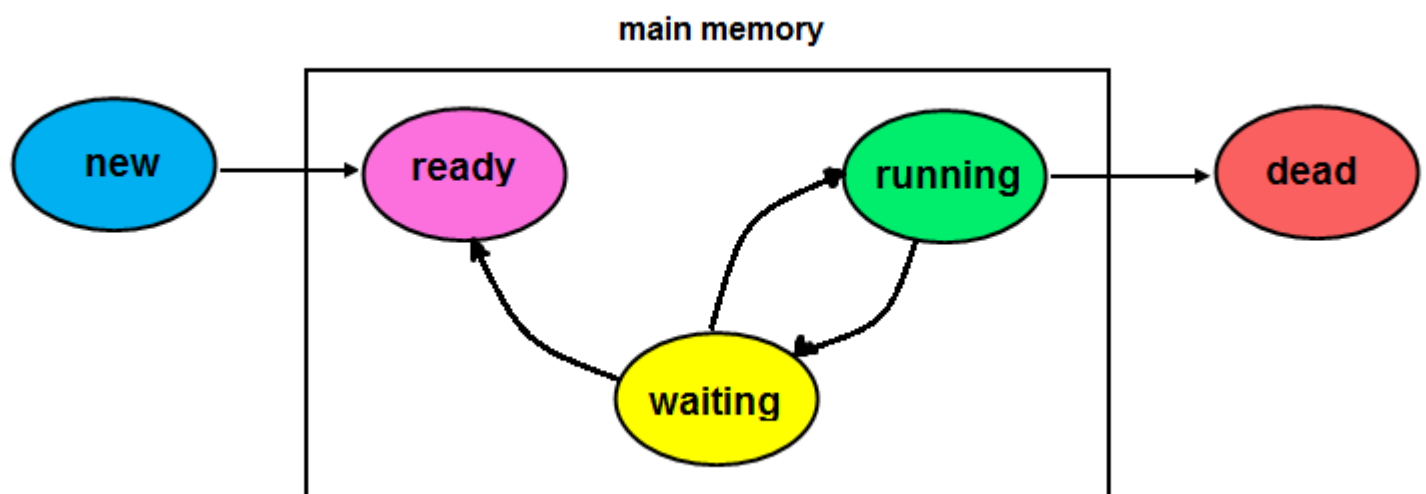
Thread

Thread is a lightweight components and it is a flow of control. In other words a flow of control is known as thread.

State or Life cycle of thread

State of a thread are classified into five types they are

1. New State
2. Ready State
3. Running State
4. Waiting State
5. Halted or dead State



New State

If any new thread class is created that represent new state of a thread, In new state thread is created and about to enter into main memory. No memory is available if the thread is in new state.

Ready State

In ready state thread will be entered into main memory, memory space is allocated for the thread and 1st time waiting for the CPU.

Running State

Whenever the thread is under execution known as running state.

Halted or dead State

If the thread execution is stoped permanently than it comes under dead state, no memory is available for the thread if its comes to dead state.

Note: If the thread is in new or dead state no memory is available but sufficient memory is available if that is in ready or running or waiting state.

Achieve multithreading in java

In java language multithreading can be achieve in two different ways.

1. Using thread class
2. Using Runnable interface

Using thread class

In java language multithreading program can be created by following below rules.

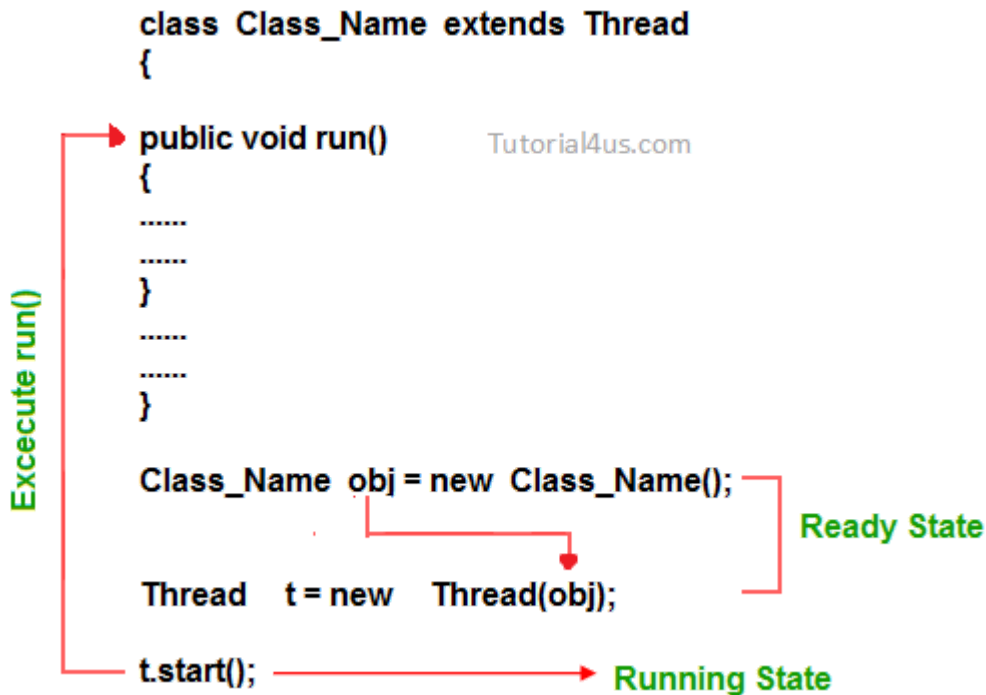
1. Create any user defined class and make that one as a derived class of thread class.

```
class Class_Name extends Thread
{
    . . . . .
}
```

2. Override run() method of Thread class (It contains the logic of perform any operation)
3. Create an object for user-defined thread class and attached that object to predefined thread class object.

```
Class_Name obj=new Class_Name
Thread t=new Thread(obj);
```

4. Call start() method of thread class to execute run() method.
5. Save the program with filename.java



Example of multithreading using Thread class

Thread based program for displaying 1 to 10 numbers after each and every second.

```
// Threaddemo2.java
```

```

class Th1 extends Thread
{
    public void run()
    {
        try
        {
            for(int i=1;i<=10;i++)
            {
                System.out.println("value of i="+i);
                Thread.sleep(1000);
            }
        }
        catch (InterruptedException ie)
        {

```

```

System.err.println("Problem in thread
execution");
}
}
}
class Threaddemo2
{
public static void main(String args[])
{
Th1    t1=new    Th1();
System.out.println("Execution status of t1
before start="+t1.isAlive());
t1.start();
System.out.println("Execution status of t1
after start="+t1.isAlive());
try
{
Thread.sleep(5000);
}
catch (InterruptedException ie)
{
System.out.println("Problem in thread
execution");
}
System.out.println("Execution status of t1
during execution="+t1.isAlive());
try
{
Thread.sleep(5001);
}
catch (InterruptedException ie)
{
System.out.println("problem in thread
execution");
}
}
}

```

```
}  
System.out.println("Execution status of t1  
after completion="+t1.isAlive());  
}  
}
```

Output

Execution status of t1 before start=false
//new state

Execution status of t1 after start=true
//ready state

1

2

3

4

5

6

Execution status of t1 during
execution=true //running state

7

8

9

10

Execution status of t1 after
completion=false //halted state

Thread class properties

Thread class contains constant data members, constructors, predefined methods.

Constant data members

- MAX-PRIORITY
- MIN-PRIORITY
- NORM-PRIORITY

MAX-PRIORITY

Which represent the minimum priority that a thread can have whose values is 10.

Syntax:

```
public static final int MAX-PRIORITY=10
```

MIN-PRIORITY

Which represents the minimum priority that a thread can have.

Syntax:

```
public static final int MIN-PRIORITY=0
```

NORM-PRIORITY

Which represent the default priority that is assigned to a thread.

Syntax:

```
public static final int NORM-PRIORITY=5
```

Constructors of Thread class

- Thread()
- Thread(String name)
- Thread(object)
- Thread(object, String name)

Thread()

Which will be execute to set the predefined name for newly created thread, these names are generally in the form of thread -0, thread -1,

Syntax to call constructor:

Syntax

```
Thread t=new Thread();
```

Thread(String name)

Which can be used to provide user defined name for newly created thread.

Syntax

```
Thread t=new Thread("newthread");
```

Thread(object)

Which can be used to provide default name for newly created user defined thread.

Syntax

```
UserdefinedThreadclass obj=new  
UserdefinedThreadclass();  
Thread t=new Thread("obj");
```

object, String name

Which will be used to provide user defined name for the newly created user defined thread.

Syntax

```
UserdefinedThreadclass obj=new  
UserdefinedThreadclass();  
Thread t=new Thread(object,  
"secondthread");
```

Methods of Thread class

- **getPriority()**
- **setPriority()**

- getName()
- setName()
- isDeamon()
- run()
- start()
- sleep()
- suspend()
- resume()
- stop()
- isAlive()
- currentThread()
- join()
- getState()
- yield()

getPriority()

This method is used to get the current priority of thread.

```
Thread t=new Thread();
int x=t.getPriority();
System.out.println(x);
```

setPriority()

This method is used to set the current priority of thread.

```
Thread t=new Thread();
t.setPriority(any priority number between 0
to 10)
or
t.setPriority(Thread.MAX-PRIORITY)
```

getName()

This method is used to get the current executing thread name.

```
Thread t=new Thread();  
String s=t.getName();  
System.out.println(s);
```

setName()

This method is used to set the userdefined name for the thread.

```
Thread t=new Thread();  
t.setName("mythread");
```

isDeamon()

Which returns true if the current thread is background thread otherwise return false.

```
Thread t=new Thread();  
boolean b=t.isDeamon();
```

run()

Which contains the main business logic that can be executed by multiple threads simultaneously in every user defined thread class run method should be overridden.

```
public Class_Name extends Thread  
{  
    public void run()  
    {  
        .....  
        .....  
    }  
}
```

start()

Used to convert ready state thread to running state.

```
Thread t=new Thread();  
t.start();
```

sleep()

Used to change running state thread to waiting state based on time period it is a static method should be called with class reference.

```
public static final sleep(long  
millisecond)throws InterruptedException  
{  
try  
{  
Thread.sleep(3000);  
}  
catch (InterruptedException ie)  
{  
.....  
.....  
}  
}
```

Once the given time period is completed thread state automatically change from waiting to running state.

suspend()

Used to convert running state thread to waiting state, which will never come back to running state automatically.

```
Thread t=new Thread();
```

```
t.suspend();
```

resume()

Used to change the suspended thread state(waiting state) to ready state.

```
Thread t=new Thread();  
t.resume();
```

Note: Without using suspend() method resume() method can not be use.

What is the difference between sleep() and suspend()

Sleep() can be used to convert running state to waiting state and automatically thread convert from waiting state to running state once the given time period is completed. Where as suspend() can be used to convert running state thread to waiting state but it will never return back to running state automatically.

stop()

This method is used to convert running state thread to dead state.

```
Thread t=new Thread();  
t.stop();
```

isAlive()

Which is return true if the thread is in ready or running or waiting state and return false if the thread is in new or dead state.

```
Thread t=new Thread();  
t.isAlive();
```

currentThread()

Used to get the current thread detail like thread name thread group name and priority

```
Thread t=new Thread();  
t.currentThread();
```

Note:

- The default thread name is thread-0, (if it is a main thread default name is main)
- The default thread group name is main
- Default thread priority is "5" is normal priority.

join()

Which can be used to combined more than one thread into a single group signature is public final void join()throws InterruptedException

```
try  
{  
t.join();  
t2.join();  
.....  
.....  
}
```

getState()

This method is used to get the current state of thread.

```
Thread t=new Thread();  
t.getState();
```

yield()

Which will keep the currently executing thread into temporarily pass and allows other threads to execute

Using Runnable Interface

Runnable is one of the predefined interface in java.lang package, which is containing only one method and whose prototype is "
Public abstract void run "

The run() method of thread class defined with null body and run() method of Runnable interface belongs to abstract. Industry is highly recommended to override abstract run() method of Runnable interface but not recommended to override null body run() method of thread class.

In some of the circumstance if one derived class is extending some type of predefined class along with thread class which is not possible because java programming never supports multiple inheritance. To avoid this multiple inheritance problem, rather than extending thread class we implement Runnable interface.

Rules to create the thread using Runnable interface

- Create any user defined class and implements runnable interface within that
- Override run() method within the user defined class.
- call start() method to execute run() method of thread class
- Save the program with classname.java

```
class Class_Name implement Runnable
{
public void run()
{
```



```

.....
}
}
Class_Name obj=new Class_name();
Thread t=new Thread();
t.start();

```

Note: While implementing runnable interface it is very mandatory to attach user defined thread class object reference to predefined thread class object reference. It is optional while creating thread by extending Thread class.

Thread Synchronization

Whenever multiple threads are trying to use same resource than they may be chance to of getting wrong output, to overcome this problem thread synchronization can be used.

Definition: Allowing only one thread at a time to utilized the same resource out of multiple threads is known as thread synchronization or thread safe.

In java language thread synchronization can be achieve in two different ways.

1. Synchronized block
2. Synchronized method

Note: synchronization is a keyword(access modifier in java)

Synchronized block

Whenever we want to execute one or more than one statement by a single thread at a time(not allowing other thread until thread one execution is completed) than those statement should be placed in side synchronized block.

```

class Class_Name implement Runnable or
extends Thread
{
public void run()
{
synchronized(this)
{
.....
.....
}
}
}

```

Synchronized method

Whenever we want to allow only one thread at a time among multiple thread for execution of a method than that should be declared as synchronized method.

```

class Class_Name implement Runnable or
extends Thread
{
public void run()
{
synchronized void fun()
{
.....
.....
}
public void run()
{
fun();
....
}
}
}

```

Interthread Communication

The process of execution of exchanging of the data / information between multiple threads is known as Interthread communication or if an output of first thread giving as an input to second thread the output of second thread giving as an input to third thread then the communication between first second and third thread known as Interthread communication.

In order to develop Interthread communication application we use some of the methods of java.lang.Object class and these methods are known as Interthread communication methods.

Interthread communication methods

1. public final void wait(long msec)
2. public final void wait()
3. public final void notify()
4. public final void notifyAll()

public final void wait(long msec)

public final void wait (long msec) is used for making the thread to wait by specifying the waiting time in terms of milliseconds. Once the waiting time is completed, automatically the thread will be interred into ready state from waiting state. This methods is not recommended to used to make next thread to wait on the basis of time because java programmer may not be able to decide or determine the CPU burst time of current thread and CPU burst time is decided by OS but not by the programmer.

public final void wait()

public final void wait() is used for making the thread to wait without specifying any waiting time this method is

recommended to use to make the next thread to wait until current thread complete its execution.

public final void notify()

public final void notify() is used for transferring one thread at a time from waiting state to ready state.

public final void notifyAll()

public final void notifyAll() is used for transferring all the threads at a time from waiting state to ready state.

Note: public final void wait (long msec) and public final void wait() throws a predefined Exception called java.lang.InterruptedException.

String Handling in Java

The basic aim of **String Handling** concept is storing the string data in the main memory (RAM), manipulating the data of the String, retrieving the part of the String etc. **String Handling** provides a lot of concepts that can be performed on a string such as concatenation of string, comparison of string, find sub string etc.

Character

It is an identifier enclosed within single quotes (' ').

Example: 'A', '\$', 'p'

String:

String is a sequence of characters enclosed within double quotes (" ") is known as **String**.

Example: "Java Programming".

In java programming to store the character data we have a fundamental datatype called **char**. Similarly to store the string data and to perform various operation on String data, we have three predefined classes they are:

- String
- StringBuffer
- StringBuilder

String class

It is a predefined class in java.lang package can be used to handle the String. String class is **immutable** that means whose content can not be changed at the time of execution of program.

String class object is immutable that means when we create an object of String class it never changes in the existing object.

Example:

Example

```
class StringHandling
{
public static void main(String arg[])
{
String s=new String("java");
s.concat("software");
System.out.println(s);
}
}
```

Output

java

Explanation: Here we can not change the object of String class so output is only java not java software.

Methods of String class

length()

length(): This method is used to get the number of character of any string.

Example

```
class StringHandling
{
public static void main(String arg[])
{
int l;
String s=new String("Java");
l=s.length();
System.out.println("Length: "+l);
}
}
```

Output

Length: 4

charAt(index)

charAt(): This method is used to get the character at a given index value.

Example

```
class StringHandling
{
public static void main(String arg[])
{
char c;
String s=new String("Java");
c=s.charAt(2);
System.out.println("Character: "+c);
}
```

```
}  
}
```

Output

Character: v

toUpperCase()

toUpperCase(): This method is use to convert lower case string into upper case.

Example

```
class StringHandling  
{  
public static void main(String arg[])  
{  
String s="Java";  
System.out.println("String:  
"+s.toUpperCase() );  
}  
}
```

Output

String: JAVA

toLowerCase()

toLowerCase(): This method is used to convert lower case string into upper case.

Example

```
class StringHandling  
{  
public static void main(String arg[])  
{  
String s="JAVA";
```

```
System.out.println("String:
"+s.toLowerCase());
}
```

Output

String: java

concat()

concat(): This method is used to combined two string.

Example

```
class StringHandling
{
public static void main(String arg[])
{
String s1="Hitesh";
String s2="Raddy";
System.out.println("Combined String:
"+s1.concat(s2));
}
}
```

Output

Combined String: HiteshRaddy

equals()

equals(): This method is used to compare two strings, It return true if strings are same otherwise return false. It is case sensitive method.

Example

```
class StringHandling
{
```



```
public static void main(String arg[])
{
String s1="Hitesh";
String s2="Raddy";
String s3="Hitesh";
System.out.println("Compare String:
"+s1.equals(s2));
System.out.println("Compare String:
"+s1.equals(s3));
}
}
```

Output

```
Compare String: false
Compare String: true
```

equalsIgnoreCase()

equalsIgnoreCase(): This method is case insensitive method, It return true if the contents of both strings are same otherwise false.

Example

```
class StringHandling
{
public static void main(String arg[])
{
String s1="Hitesh";
String s2="HITESH";
String s3="Raddy";
System.out.println("Compare String:
"+s1.equalsIgnoreCase(s2));
System.out.println("Compare String:
"+s1.equalsIgnoreCase(s3));
}
}
```

```
}
```

Output

```
Compare String: true  
Compare String: false
```

compareTo()

compareTo(): This method is used to compare two strings by taking unicode values, It return 0 if the string are same otherwise return +ve or -ve integer values.

Example

```
class StringHandling  
{  
public static void main(String arg[])  
{  
String s1="Hitesh";  
String s2="Raddy";  
int i;  
i=s1.compareTo(s2);  
if(i==0)  
{  
System.out.println("Strings are same");  
}  
else  
{  
System.out.println("Strings are not same");  
}  
}  
}
```

Output

```
Strings are not same
```

compareToIgnoreCase()

compareToIgnoreCase(): This method is case insensitive method, which is used to compare two strings similar to compareTo().

Example

```
class StringHandling
{
public static void main(String arg[])
{
String s1="Hitesh";
String s2="HITESH";
int i;
i=s1.compareToIgnoreCase(s2);
if(i==0)
{
System.out.println("Strings are same");
}
else
{
System.out.println("Strings are not same");
}
}
}
```

Output

Strings are same

startsWith()

startsWith(): This method return true if string is start with given another string, otherwise it returns false.

Example

```
class StringHandling
{
```

```
public static void main(String arg[])
{
String s="Java is programming language";
System.out.println(s.startsWith("Java"));
}
}
```

Output

true

endsWith()

endsWith(): This method return true if string is end with given another string, otherwise it returns false.

Example

```
class StringHandling
{
public static void main(String arg[])
{
String s="Java is programming language";
System.out.println(s.endsWith("language"));
}
}
```

Output

true

substring()

substring(): This method is used to get the part of given string.

Example

```
class StringHandling
{
public static void main(String arg[])
```

```
{  
String s="Java is programming language";  
System.out.println(s.substring(8)); // 8 is  
starting index  
}  
}
```

Output

programming language

Example

```
class StringHandling  
{  
public static void main(String arg[])  
{  
String s="Java is programming language";  
System.out.println(s.substring(8, 12));  
}  
}
```

Output

prog

indexOf()

indexOf(): This method is used find the index value of given string. It always gives starting index value of first occurrence of string.

Example

```
class StringHandling  
{  
public static void main(String arg[])  
{  
String s="Java is programming language";
```

```
System.out.println(s.indexOf("programming")
);
}
}
```

Output

8

lastIndexOf()

lastIndexOf(): This method used to return the starting index value of last occurrence of the given string.

Example

```
class StringHandling
{
public static void main(String arg[])
{
String s1="Java is programming language";
String s2="Java is good programming
language";
System.out.println(s1.lastIndexOf("programm
ing"));
System.out.println(s2.lastIndexOf("programm
ing"));
}
}
```

Output

8

13

trim()

trim(): This method remove space which are available before starting of string and after ending of string.

Example

```
class StringHandling
{
public static void main(String arg[])
{
String s="      Java is programming language
";
System.out.println(s.trim());
}
}
```

Output

Java is programming language

split()

split(): This method is used to divide the given string into number of parts based on delimiter (special symbols like @ space ,).

Example

```
class StringHandling
{
public static void main(String arg[])
{
String s="contact@tutorial4us.com";
String[] s1=s.split("@"); // divide string
based on @
for(String c:s1) // foreach loop
{
System.out.println(c);
}
}
}
```

Output

contact
@tutorial4us.com

replace()

replace(): This method is used to return a duplicate string by replacing old character with new character.

Note: In this method data of original string will never be modify.

Example

```
class StringHandling
{
public static void main(String arg[])
{
String s1="java";
String s2=s1.replace('j', 'k');
System.out.println(s2);
}
}
```

Output

kava

StringBuffer Class in Java

It is a predefined class in java.lang package can be used to handle the String, whose object is mutable that means content can be modify.

StringBuffer class is working with thread safe mechanism that means multiple thread are not allowed simultaneously to perform operation of StringBuffer.

StringBuffer class object is **mutable** that means when we create an object of StringBulder class it can be change.

Example StringBuffer

```
class StringHandling
{
public static void main(String arg[])
{
StringBuffer sb=new StringBuffer("java");
sb.append("software");
System.out.println(sb);
}
}
```

Output

javasoftware

Explanation: Here we can changes in the existing object of StringBuffer class so output is javasoftware.

Difference Between String and StringBuffer

String

The data which enclosed within double quote (" ") is by default treated as String class.

String class object is immutable

When we create an object of String class by default no additional character memory space is created.

StringBuffer

The data which enclosed within double quote (" ") is not by default treated as StringBuffer class

StringBuffer class object is mutable

When we create an object of StringBuffer class by default we get 16 additional character memory space.

Similarities Between String and StringBuffer

- Both of them are belongs to public final. so that they never participates in inheritance that is is-A relationship is not possible but they can always participates in As-A and Uses-A relationship.
- We can not override the method of String and StringBuffer.

Methods of StringBuffer class

reverse()

reverse(): This method is used to reverse the given string and also the new value is replaced by the old string.

Example

```
class StringHandling
{
public static void main(String arg[])
{
StringBuffer sb=new StringBuffer("java
code");
System.out.println(sb.reverse());
}
}
```

Output

edoc avaj

insert()

insert(): This method is used to insert either string or character or integer or real constant or boolean value at a specific index value of given string.

Example

```
class StringHandling
{
public static void main(String arg[])
{
StringBuffer sb=new StringBuffer("this is
my java code");
System.out.println(sb.insert(11, "first
")) ;
}
}
```

Output

this is my first java code

append()

append(): This method is used to add the new string at the end of original string.

Example

```
class StringHandling
{
public static void main(String arg[])
{
StringBuffer sb=new StringBuffer("java is
easy");
System.out.println(sb.append(" to learn"));
}
}
```

Output

java is easy to learn

replace()

replace() This method is used to replace any old string with new string based on index value.

Example

```
class StringHandling
{
public static void main(String arg[])
{
StringBuffer sb=new StringBuffer("This is
my code");
System.out.println(sb.replace(8, 10,
"java"));
}
}
```

Output

This is java code

Explanation: In above example java string is replaced with old string (my) which is available between 8 to 10 index value.

deleteCharAt()

deleteCharAt(): This method is used to delete a character at given index value.

Example

```
class StringHandling
{
public static void main(String arg[])
{
StringBuffer sb=new StringBuffer("java");
System.out.println(sb.deleteCharAt(3));
}
}
```

Output

jav

delete()

delete(): This method is used to delete string from given string based on index value.

Example

```
class StringHandling
{
public static void main(String arg[])
{
StringBuffer sb=new StringBuffer("java is
easy to learn");
StringBuffer s;
s=sb.delete(8, 13);
System.out.println(sb);
}
}
```

Output

java is to learn

Explanation: In above example string will be deleted which is existing between 8 and 13 index value.

toString()

toString(): This method is used to convert mutable string value into immutable string.

Example

```
class StringHandling
{
public static void main(String arg[])
```

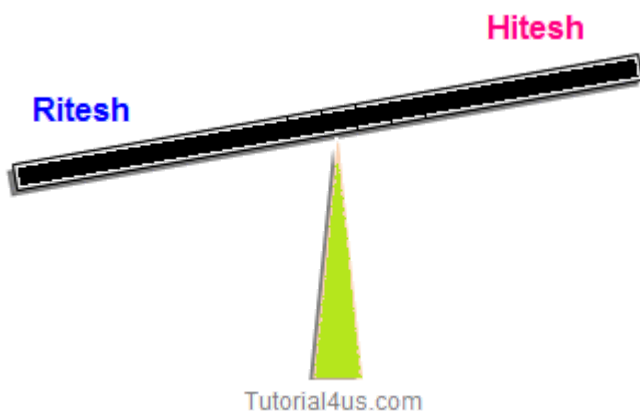
```
{  
StringBuffer sb=new StringBuffer("java");  
String s=sb.toString();  
System.out.println(s);  
s.concat("code");  
}  
}
```

Output

java

String Compare in Java

There are three way to compare string object in java:



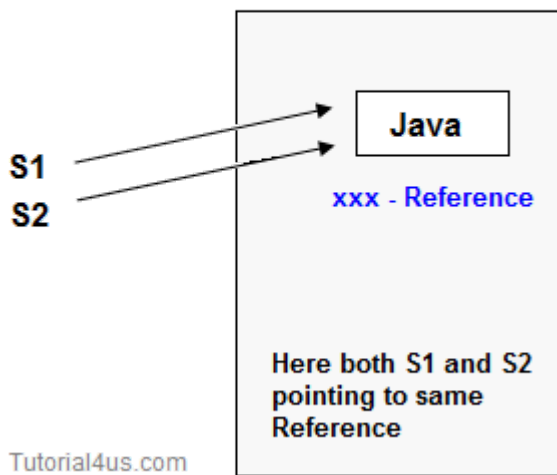
- By equals() method
- By == operator
- By compareTo() method

equals() Method in Java

equals() method always used to comparing contents of both source and destination String. It return true if both string are same in meaning and case otherwise it returns false. It is case sensitive method.

String s1 = "Java";

String s2 = "Java";



Example

```
class StringHandling
{
public static void main(String arg[])
{
String s1="Hitesh";
String s2="Raddy";
String s3="Hitesh";
System.out.println("Compare String:
"+s1.equals(s2));
System.out.println("Compare String:
"+s1.equals(s3));
}
}
```

Output

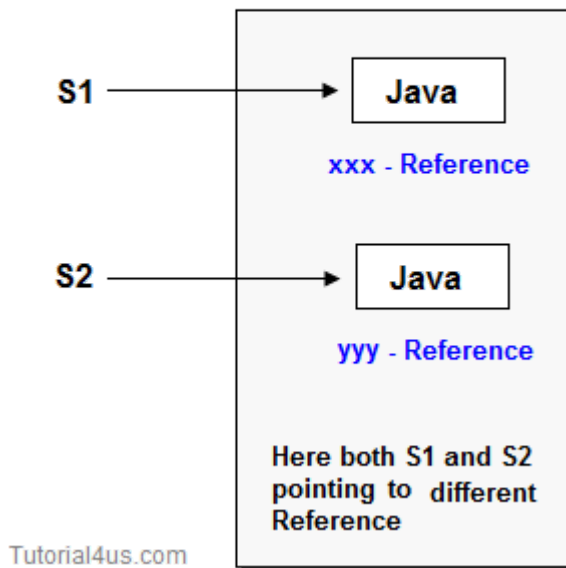
Compare String: false
Compare String: true

== or Double Equals to Operator in Java

== Operator is always used for comparing references of both source and destination objects but not their contents.

```
String s1 = new String ("Java");
```

```
String s2 = new String ("Java");
```



Example

```
class StringHandling
{
public static void main(String arg[])
{
String s1=new String("java");
String s2=new String("java");
if(s1==s2)
{
System.out.println("Strings are same");
}
else
{
System.out.println("Strings are not same");
}
}
}
```

Output

Strings are not same

compareTo() Method in Java

compareTo() method can be used to compare two string by taking Unicode values. It returns 0 if the string are same otherwise returns either +ve or -ve integer.

Example

```
class StringHandling
{
public static void main(String arg[])
{
String s1="Hitesh";
String s2="Raddy";
int i;
i=s1.compareTo(s2);
if(i==0)
{
System.out.println("Strings are same");
}
else
{
System.out.println("Strings are not same");
}
}
}
```

Output

Strings are not same

Difference between equals() method and == operator

equals() method always used to comparing contents of both source and destination String.

== Operator is always used for comparing references of both source and destination objects but not their contents.

String Concatenation

There are two way to concat string object in java:

- By + (string concatenation) operator
- By concat() method

By + operator

Using Java string concatenation operator (+) you can combined two or more strings.

Example

```
class StringHandling
{
public static void main(String arg[])
{
String s= "Java" + "Code";
System.out.println(s);
}
}
```

Output

JavaCode

By concat() method

concat() method is used to combined two strings.

Example

```
class StringHandling
{
public static void main(String arg[])
```

```
{  
String s1="Java";  
String s2="Code";  
String s3=s1.concat(s2);  
System.out.println(s3);  
}  
}
```

Output

JavaCode

Applet in Java

Applet is a predefined class in **java.applet** package used to design distributed application. It is a client side technology. Applets are run on web browser.

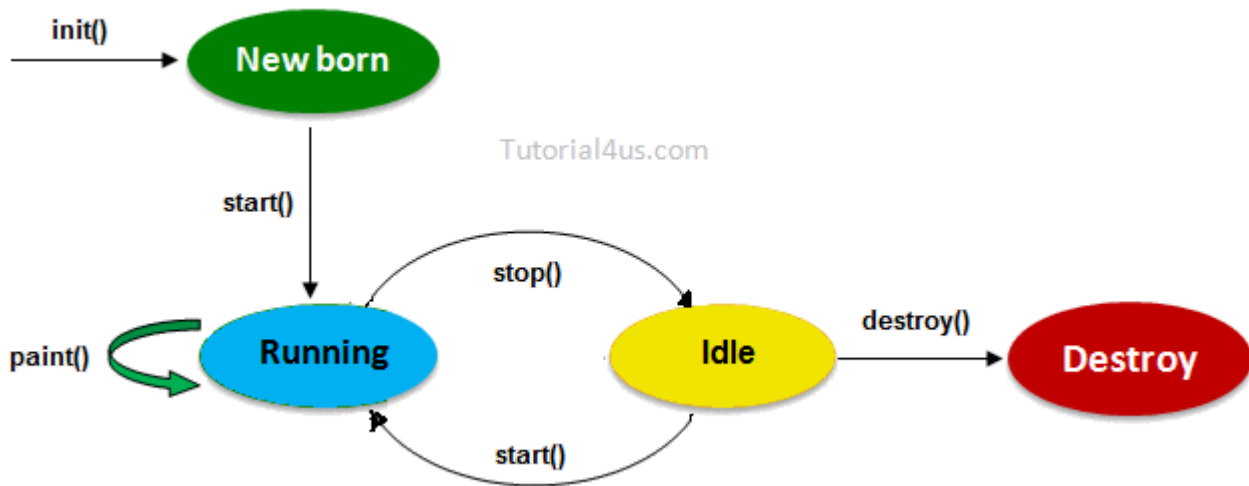
Advantage of Applet

- Applets are supported by most web browsers.
- Applets works on client side so less response time.
- **Secured:** No access to the local machine and can only access the server it came from.
- Easy to develop applet, just extends applet class.
- To run applets, it requires the Java plug-in at client side.
- Android, do not run Java applets.
- Some applets require a specific JRE. If it required new JRE then it take more time to download new JRE.

Life cycle of applet

- init()
- start()
- stop

- destroy



init(): Which will be executed whenever an applet program start loading, it contains the logic to initiate the applet properties.

start(): It will be executed whenever the applet program starts running.

stop(): Which will be executed whenever the applet window or browser is minimized.

destroy(): It will be executed whenever the applet window or browser is going to be closed (at the time of destroying the applet program permanently).

Design applet program

We can design our own applet program by extending applet class in the user defined class.

Syntax

```

class className extends Applet
{
    .....
    // override lifecycle methods
    .....
}
  
```

```
}
```

Note: Whenever an applet program is running `init()` and `start()` will be executed one after another, but `stop()` and `destroy()` will be executed if the browser is minimized and closed by the end user, respectively.

Note: Applet program may or may not contain life cycle methods.

Running of applet programs

Applet program can run in two ways.

- Using html (in the web browser)
- Using appletviewer tool (in applet window)

Running of applet using html

In general no Java program can directly execute on the web browser except markup language like html, xml etc.

Html support a predefined tag called `<applet>` to load the applet program on the browser window.

Syntax

```
<applet code="udc.class">  
height="100px"  
width="100px"  
</applet>
```

Example of applet program to run applet using html

Java code, JavaApp.java

```
import java.applet.*;  
import java.awt.*;  
public class JavaApp extends Applet
```

```

{
public void paint(Graphics g)
{
Font f=new Font("Arial",Font.BOLD,30);
g.setFont(f);
setForeground(Color.red);
setBackground(Color.white);
g.drawString("Student",200,200);
}
}

```

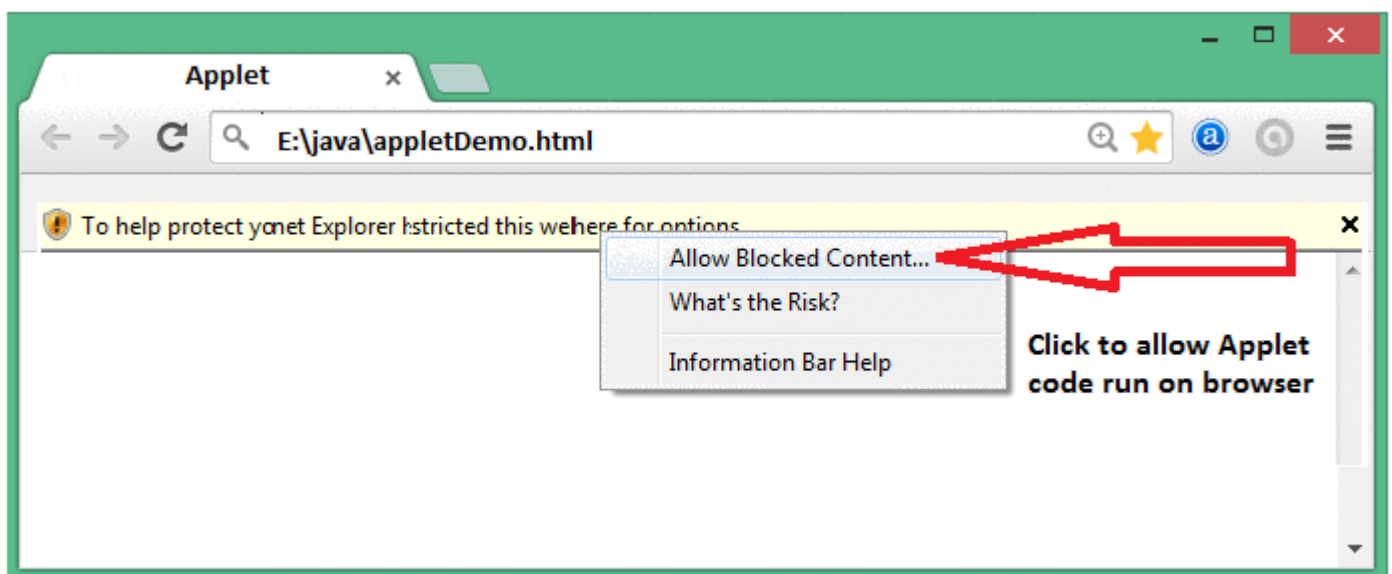
Html code, myapplet.html

```

<html>
<title> AppletEx</Title>
<body>
<applet code="JavaApp.class"
        height="70%"
        width="80%">
</applet>
</body>
</html>

```

If applet code not run on browser then allow blocked contents.



Running of applet using appletviewer

Some browser does not support **<applet>** tag so that Sun Microsystems was introduced a special tool called **appletviewer** to run the applet program.

In this Scenario Java program should contain **<applet>** tag in the commented lines so that appletviewer tools can run the current applet program.

Example of Applet

```
import java.applet.*;
import java.awt.*;

/*<applet code="LifeApp.class"
height="500",width="800">
</applet>*/

public class LifeApp extends Applet
{
    String s= " ";
    public void init()
    {
        s=s+ " int ";
    }
    public void start()
    {
        s=s+ "start ";
    }
    public void stop()
    {
        s=s+ "stop ";
    }
    public void destroy()
```

```

    {
        s=s+ " destory ";
    }
    public void paint(Graphics g)
    {
        Font f=new
Font("Arial",Font.BOLD,30);
        setBackgroundColor(Color."red");
        g.setFont(f);
        g.drawString(s,200,250);
    }
}

```

Execution of applet program

```

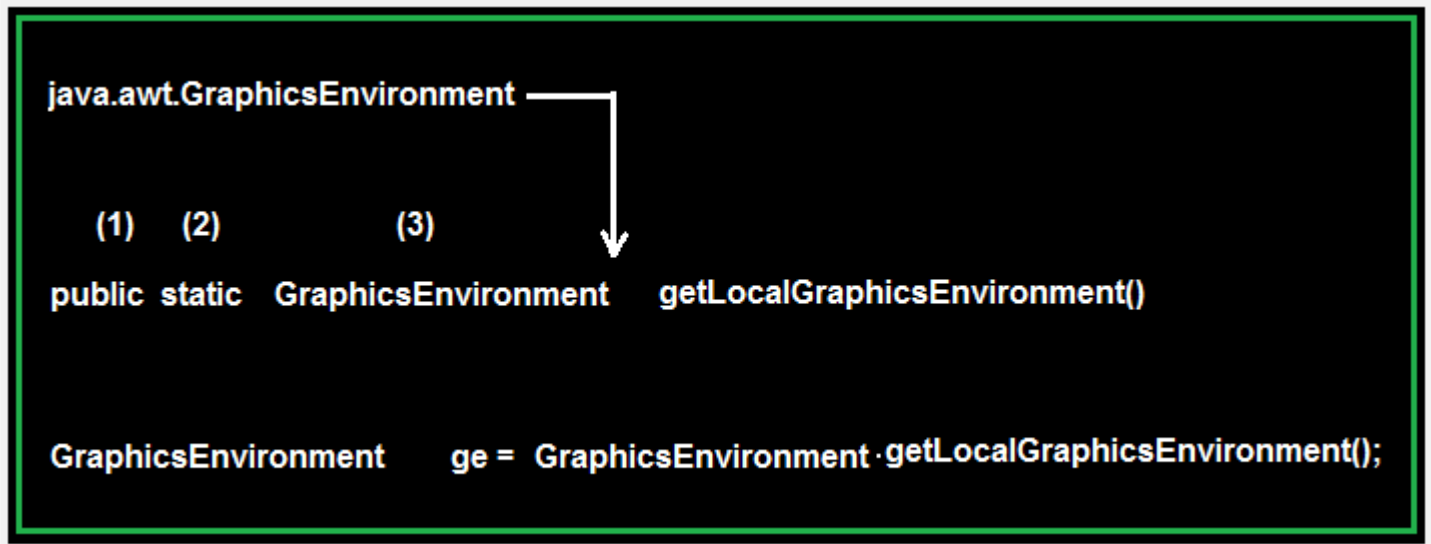
javac LifeApp.java
appletviewer LifeApp.java

```

Note: init() always execute only once at the time of loading applet window and also it will be executed if the applet is restarted.

Factory Method in Java

A factory method is one whose return type is similar to the class name in which class is present. The purpose of factory method is to create an object without using new operator.



Rules for writing factory method

1. The return type of the factory method must be similar to class name in which class it presents.
2. Every factory method in java is static.
3. The access specifier of the factory method must be public.

Socket Programming in Java

Networking is a concept of connecting two or more computing devices together so that we can share resources like printer, scanner, memory.

In Networking application mainly two programs are running one is **Client program** and another is **Server program**. In Core java **Client program** can be design using **Socket** class and **Server program** can be design using **ServerSocket** class.

Both Socket and ServerSocket classes are predefined in **java.net** package

Advantage of Network Programming

The main advantage of network Programming is sharing of data and resources, some more advantages are;

- Sharing resources like printer, Scanner.
- Centralize software management, Software install on only one system and used in multiple system.
- Sharing of data due to this reduce redundancy of application.
- Burden on the developer can be reduced.
- Wastage of memory can be reduced because no need to install same application on every system.
- Time consuming process to develop application is reduced.

Terms used in Socket Programming

Port number: It is unique identification value represents residing position of a server in the computer. It is four digit +ve number.

Port Name: It is a valid user defined name to know about client system, the default port name for any local computer is **localhost**.. every Port name should accompany with port no some value which is given at Server programming.

Socket class

Socket class are used for design a client program, it have some constructor and methods which are used in designing client program.

Constructor: Socket class is having a constructor through this Client program can request to server to get connection.

Syntax to call Socket() Constructor

```
Socket s=new Socket("localhost", 8080);
```

// localhost -- port name and 8080 -- port number

Note: If given port name is invalid then `UnknownHostException` will be raised.

Method of Socket class

- `public InputStream getInputStream()`
- `public OutputStream getOutputStream()`
- `public synchronized void close()`

getOutputStream ()

This method take the permission to write the data from client program to server program and server program to client program which returns `OutputStream` class object.

Syntax

```
Socket s=new Socket("localhost", 8080);
OutputStream os=new s.getOutputStream();
DataOutputStream dos=new
DataOutputStream(os);
```

getInputStream()

This method is used to take the permission to read data from client system by the server or from the server system by the client which returns `InputStream` class object.

Syntax

```
Socket s=new Socket("localhost", 8080);
InputStream is=new s.getInputStream();
DataInputStream dis=new
DataInputStream(is);
```

close()

This method is used to request for closing or terminating an object of socket class or it is used to close client request.

Syntax

```
Socket s=new Socket("localhost", 8080);  
s.close();
```

ServerSocket class

The ServerSocket class can be used to create a server socket. ServerSocket object is used to establish the communication with clients.

ServerSocket class are used for design a server program, it has some constructor and methods which are used in designing server program.

Constructor: ServerSocket class contain a constructor used to create a separate port number to run the server program.

Syntax to call ServerSocket() Constructor

```
ServerSocket ss=new ServerSocket(8080);  
// 8080 -- port number
```

Method of ServerSocket class

- public Socket accept()
- public InputStream getInputStream()
- public OutputStream getOutputStream()
- public synchronized void close()

accept(): Used to accept the client request it returns class reference.

Syntax

```
Socket s=new Socket("localhost", 8080);  
ServerSocket ss=new ServerSocket(8080);  
Socket s=ss.accept();
```

Rules to design server program

- Every server program should run accepted port number (any 4 digit +ve numeric value) It can set by relating an object for server socket class (In any used defined java program).

Syntax

```
ServerSocket ss=new ServerSocket(8080);
```

- Accept client request.
- Read input data from client using InputStream class.
- Perform valid business logic operation
- Send response (writing output data) back to client using OutputStream class.
- close or terminate client request.

Rules to design client program

- Obtain connection to the server from the client program (any user defined class) by passing port number and port name in the socket class.

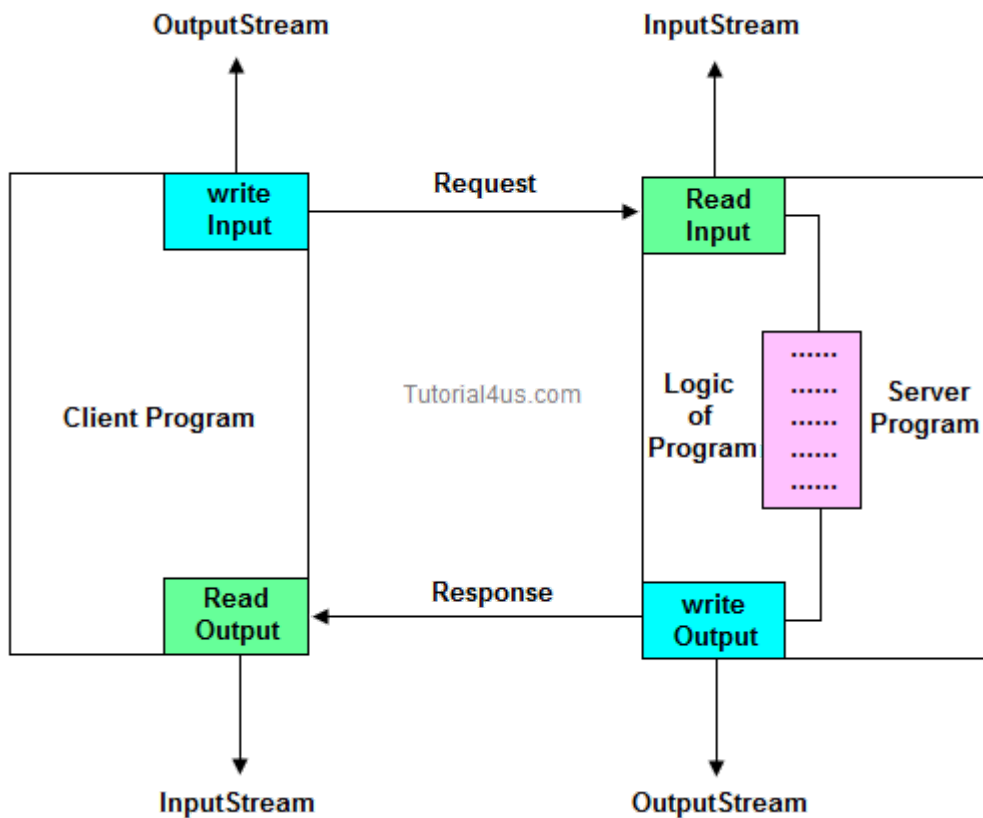
Syntax

```
Socket s=new Socket(8080, "localhost");  
// 8080 is port number and localhost is  
port name
```

- Send request (writing input data) to the server using OutputStream class.

- Read output data from the server using InputStream class.
- Display output data

Note: close the connection is optional.



Server Code

// Saved by Server.java

```

import java.net.*;
import java.io.*;

class Server
{
public static void main(String[] args)
{
try
{
int pno=Integer.parseInt(args[0]);
ServerSocket ss=new ServerSocket(pno);

```

```

System.out.println("server is ready to
accept client request");
Socket s1=ss.accept();
InputStream is=s1.getInputStream();
DataInputStream dis=new
DataInputStream(is);
int n=dis.readInt();
System.out.println("Value from client :
"+n);
int res=n*n;
OutputStream os=s1.getOutputStream();
DataOutputStream dos=new
DataOutputStream(os);
dos.writeInt(res);
s1.close();
}
catch (Exception e)
{
System.out.println(e);
}
}
}

```

Client Code

```
// Saved by Client.java
```

```

import java.net.*;
import java.io.*;
import java.util.*;

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

```

```

try
{
    String pname=args[0];
    int pno=Integer.parseInt(args[1]);
    Socket s=new Socket(pname,pno);
    System.out.println("client obtained
connection from server");
    System.out.println("Enter a number ");
    Scanner sn=new Scanner(System.in);
    int data=sn.nextInt();
    OutputStream os=s.getOutputStream();
    DataOutputStream dos=new
DataOutputStream(os);
    dos.writeInt(data);
    InputStream is=s.getInputStream();
    DataInputStream dis=new
DataInputStream(is);
    int res=dis.readInt();
    System.out.println("Result from server :
"+res);
}
catch (Exception e)
{
    System.out.println(e);
}
}
}

```

Steps to run above program

- First open two command prompt window separately.
- Compile Client program on one command prompt and compile server program on other command prompt.
- First run Server program and pass valid port number.

- Second run client program and pass valid port name and post number (which is already passed at server).
- Finally give the request to server from client (from client command prompt).
- Now you can get response from server.

Compile and run server program

```

C:\Windows\system32\cmd.exe
Tutorial4us.com

C:\java code>javac Server.java
C:\java code>java Server 1234
server is ready to accept clint request
Value from client : 4
C:\java code>
  
```

First compile Server code

Run Server program and pass valied port number
Here 1234 is port number

Request from client as a value to calculate square

Compile and run client program

```

C:\Windows\system32\cmd.exe
Tutorial4us.com

C:\java code>javac Client.java
C:\java code>java Client localhost 1234
clint obtailed connection from server
Enter a number
4
Result from server : 16
C:\java code>
  
```

First compile Client code

Run Client code and pass port name and port number
Here localhost is port name and 1234 is port number

Response from Server as a output

Limitation of J2SE network programming

- Using this concept you can share resource locally but not globally.
- Using this concept you can not develop internet based application.
- Using this concept you can develop only half-duplex application.

- Using this concept you can not get service from universal protocols like http, ftp etc...
- Using this concept you can not get services from universal server software like tomcat, weblogic etc..

To overcome all these limitation use Servlet and JSP technology.

AWT IN JAVA

Any user interact with java program in two different way, CUI (Command user Interface) and GUI (Graphical user Interface). In this tutorial we discuss about how to user interact with java code with GUI components.

Event Handling

Changing the state of an object is known as an event. The process of handling the request in GUI screen is known as **event handling** (event represent an action). It will be changes component to component.

AWT Introduction

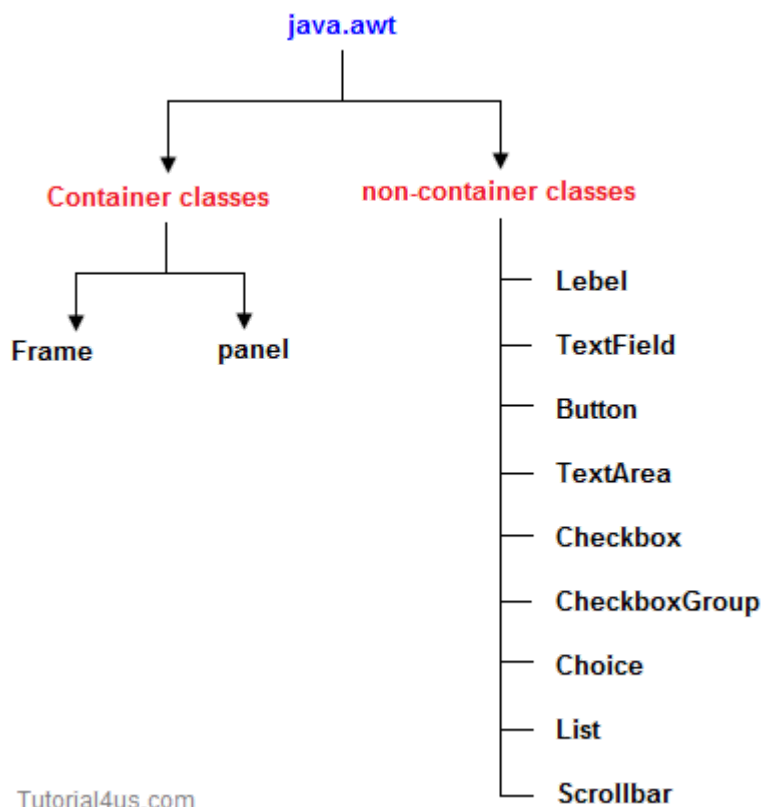
AWT stand for **abstract window toolkits**. Any user interact with java program in two different way;

- CUI (Command user Interface)
- GUI (Graphical user Interface)

If any user interact with java program by passing same commands through command prompt is known as **CUI**, if any user interact with java program through a graphical window known as **GUI**.

In core java GUI can be design using some predefined classes. All these classes are defined in **java.awt** package.

The predefined classes of awt package are classified into following types;



Container classes

These are the predefined classes in java.awt package which can be used **to display all non-container classes** to the end user in the frame of window. container classes are; frame, panel.

non-container classes

These are the predefined classes used **to design the input field** so that end user can provide input value to communicate with java program, these are also treated as GUI component. non-container classes are; Label, Button, List etc...

non-container classes

These are the predefined classes used to design the input field so that end user can provide input value to communicate with java program, these are also treated as GUI component.

Label

It can be any user defined name used to identify input field like textbox, textarea etc.

Example

```
Label l1=new Label("uname");  
Label l2=new Label("password");
```

TextField

This class can be used to design textbox.

Example

```
TextField tf=new TextField();
```

Example

```
TextField tf=new TextField(40);
```

Button

This class can be used to create a push button.

Example

```
Button b1=new Button("submit");  
Button b2=new Button("cancel");
```

TextArea

This class can be used to design a textarea, which will accept number of characters in rows and columns formate.

Example

```
TextArea ta=new TextArea(5, 10);  
// here 5 is no. of rows and 10 is no. of  
column
```

Note: In above example at a time we can give the contains in textarea is in 5 rows and 9 column (n-1 columns).

Checkbox

This class can be used to design multi selection checkbox.

Example

```
Checkbox cb1=new Checkbox(".net");  
Checkbox cb2=new Checkbox("Java");  
Checkbox cb3=new Checkbox("html");  
Checkbox cb4=new Checkbox("php");
```

CheckboxGroup

This class can be used to design radio button. Radio button is used for single selection.

Example

```
CheckboxGroup cbg=new CheckboxGroup();  
Checkbox cb=new Checkbox("male", cbg,  
"true");  
Checkbox cb=new Checkbox("female", cbg,  
"false");
```

Note: Under one group many number of radio button can exist but only one can be selected at a time.

Choice

This class can be used to design a drop down box with more options and supports single selection.

Example

```
Choice c=new Choice();  
c.add(20);  
c.add(21);  
c.add(22);  
c.add(23);
```

List

This class can be used to design a list box with multiple options and support multi selection.

Example

```
List l=new List(3);  
l.add("goa");  
l.add("delhi");  
l.add("pune");
```

Note: By holding **ctrl** button multiple options can be selected.

Scrollbar

This class can be used to display a scrollbar on the window.

Example

```
Scrollbar sb=new Scrollbar(type of  
scrollbar, initialposition, thumb width,  
minnum value, maximum value);
```

- Initial position represent very starting point or position of thumb.

- Thumb width represent the size of thumb which is scroll on scrollbar
- Minimum and maximum value represent boundaries of scrollbar

Type of scrollbar

There are two types of scrollbar are available;

- scrollbar.vertical
- scrollbar.horizontal

Example

```
Scrollbar sb=new
Scrollbar(Scrollbar.HORIZONTAL, 10, 5, 0,
100);
```

Awt Frame

Frame

It is a predefined class used to create a container with the title bar and body part.

Example

```
Frame f=new Frame();
```

Mostly used methods

setTitle()

It is used to display user defined message on title bar.

Example

```
Frame f=new Frame();
f.setTitle("myframe");
```

setBackground()

It is used to set background or image of frame.

Example

```
Frame f=new Frame();  
f.setBackground(Color.red);
```

Example

```
Frame f=new Frame();  
f.setBackground("c:\\image\\myimage.png");
```

setForeground()

It is used to set the foreground text color.

Example

```
Frame f=new Frame();  
f.setForeground(Color.red);
```

setSize()

It is used to set the width and height for frame.

Example

```
Frame f=new Frame();  
f.setSize(400,300);
```

setVisible()

It is used to make the frame as visible to end user.

Example

```
Frame f=new Frame();  
f.setVisible(true);
```


Note: You can write setVisible(true) or setVisible(false), if it is true then it is visible otherwise not visible.

setLayout()

It is used to set any layout to the frame. You can also set null layout it means no any layout apply on frame.

Example

```
Frame f=new Frame();  
f.setLayout(new FlowLayout());
```

Note: Layout is a logical container used to arrange the gui components in a specific order

add()

It is used to add non-container components (Button, List) to the frame.

Example

```
Frame f=new Frame();  
Button b=new Button("Click");  
f.add(b);
```

Explanation: In above code we add button on frame using f.add(b), here b is the object of Button class..

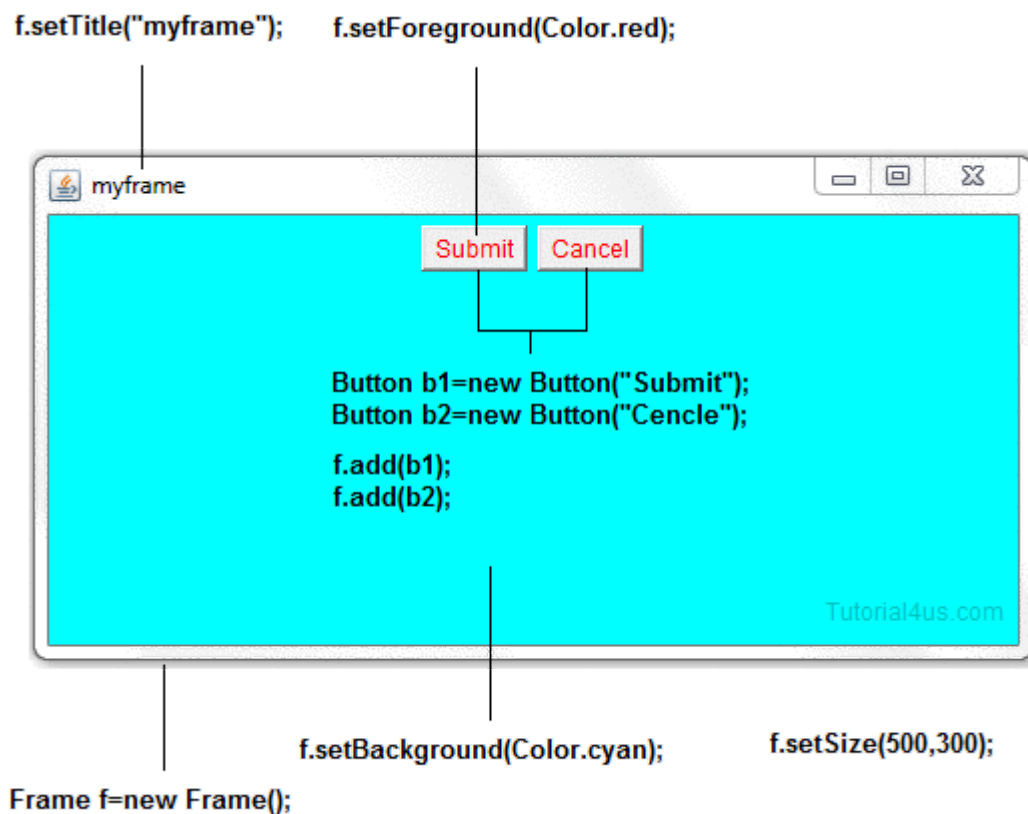
Example of Frame

```
import java.awt.*;  
  
class FrameDemo  
{  
    public static void main(String[] args)  
    {  
        Frame f=new Frame();  
    }  
}
```

```

f.setTitle("myframe");
f.setBackground(Color.cyan);
f.setForeground(Color.red);
f.setLayout(new FlowLayout());
Button b1=new Button("Submit");
Button b2=new Button("Cancel");
f.add(b1);
f.add(b2);
f.setSize(500,300);
f.setVisible(true);
}
}

```



Example of Frame

```

import java.awt.*;

class AddEmployee extends Frame

```

```

{

Label
user_id,fn,ln,ftn,DOB,sex,country,mobile,address;
TextField ID,f_name, l_name, Father1,
Father2, dob, add1, mobile_no;
Choice c;
Button b1,b2,b3,b4;
CheckboxGroup cg1;
Checkbox male,female;

public AddEmployee()
{

setTitle("Employee Detail");
setBackground(Color.cyan);
setLayout(null);

fn=new Label("Emp Name");
ftn=new Label("Father Name:");
DOB=new Label("Date of birth:");
sex=new Label("Sex:");
address=new Label("Address:");
country=new Label("Country:");
mobile=new Label("Mobile no:");

Font f=new Font("Arial", Font.BOLD, 15);
fn.setFont(f);
ftn.setFont(f);
DOB.setFont(f);
sex.setFont(f);
address.setFont(f);
country.setFont(f);

```

```

mobile.setFont(f);
fn.setBounds(20,50,100,40);
ftn.setBounds(20,100,100,40);
DOB.setBounds(20,150,100,40);
sex.setBounds(20,200,100,40);
address.setBounds(20,250,100,50);
country.setBounds(20,300,100,50);
mobile.setBounds(20,350,100,40);
add(fn);
add(ftn);
add(DOB);
add(sex);
add(address);
add(country);
add(mobile);

f_name=new TextField("",20);
Father1=new TextField("",20);
dob=new TextField("Day/Month/Year",20);
add1=new TextField("",40);
mobile_no=new TextField("",20);

f_name.setBounds(200,60,200,20);
Father1.setBounds(200,110,200,20);

dob.setBounds(200,160,140,20);
add1.setBounds(200,260,200,20);
mobile_no.setBounds(200,360,130,20);

add(f_name);
add(Father1);
add(dob);

cg1=new CheckboxGroup();

```

```
male=new Checkbox("Male", cg1, true);
female=new Checkbox("Female", cg1, false);
add(male);
add(female);
male.setBounds(200,200,50,50);
female.setBounds(300,200,60,50);

add(add1);

c=new Choice();

c.addItem("City");
c.addItem("New Delhi");
c.addItem("Raipur");
c.addItem("Chandigarh");
c.addItem("Dehradun");
c.addItem("Patna");
c.addItem("Dispur");
c.addItem("Other");
c.setBounds(200,310,130,20);
add(c);
add(c);
add(c);
add(c);
add(c);
add(c);
add(c);
add(c);
add(c);
add(c);
add(c);
add(c);
```

```

add(mobile_no);

Font fa=new Font("Arial", Font.BOLD, 15);
b1=new Button("Submit");
b2=new Button("Exit");
b3=new Button("AddNew");

b1.setBounds(170,430,80,30);
b2.setBounds(270,430,80,30);
b3.setBounds(370,430,80,30);
b1.setBackground(Color.pink);
b2.setBackground(Color.pink);
b3.setBackground(Color.pink);

add(b1);
add(b2);
add(b3);

b1.setFont(fa);
b2.setFont(fa);
b3.setFont(fa);

setSize(600,500);
setVisible(true);

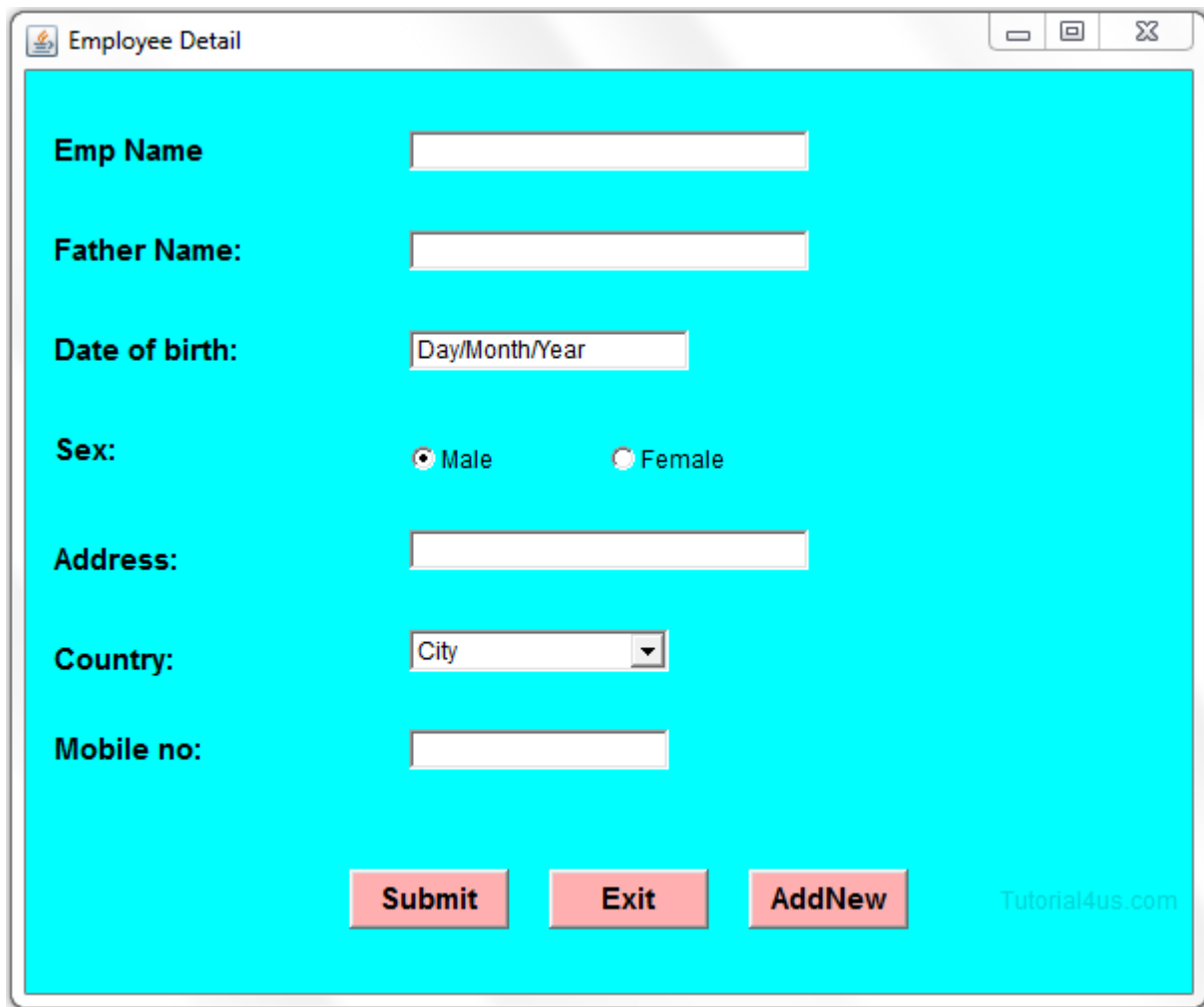
} //cons

} // class

class Employee
{
public static void main(String s[])
{
AddEmployee obj=new AddEmployee();

```

```
}  
}
```



Awt

Panel

It is a predefined class used to provide a logical container to hold various GUI component. Panel always should exist as a part of frame.

Note: Frame is always visible to end user where as panel is not visible to end user.

Panel is a derived class of container class so you can use all the methods which is used in frame.

Syntax

```
Panel p=new Panel();  
p.setBackground(Color.red);
```

```
p.setSize(400,300);
```

Example frame and panel

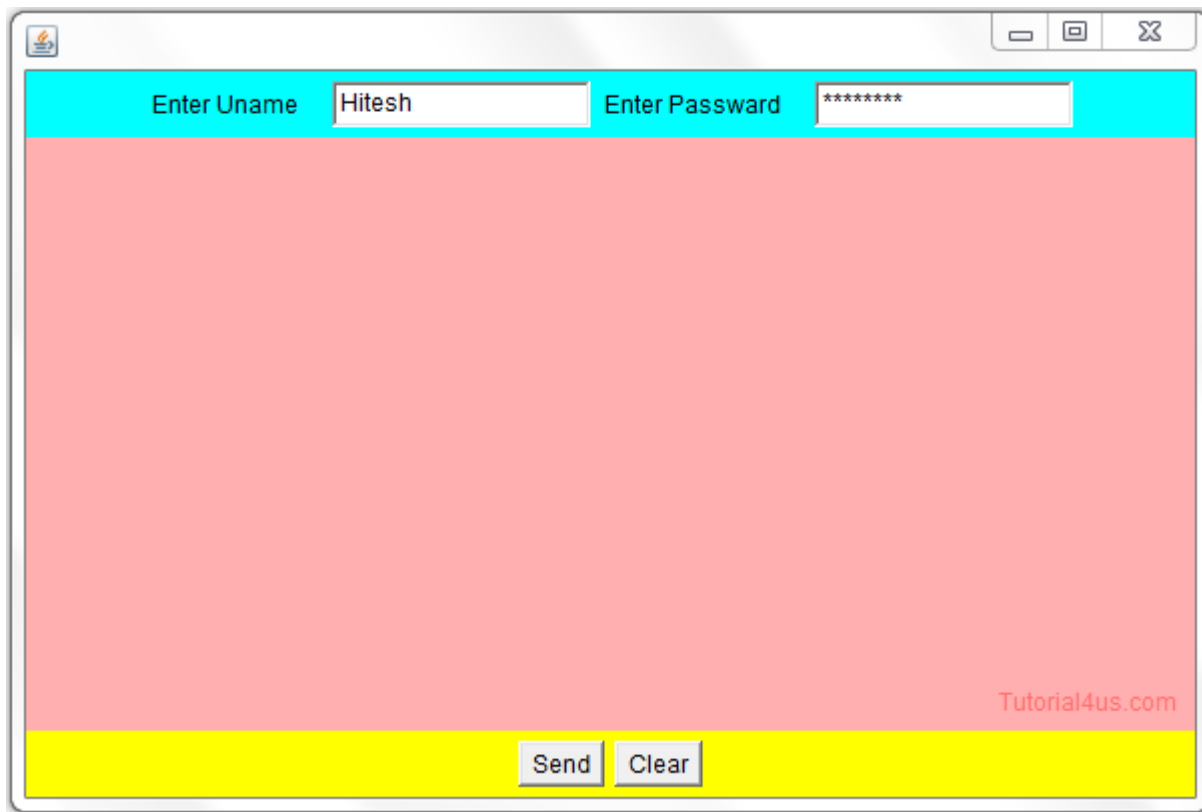
```
import java.awt.*;
class PanelFrame
{
    PanelFrame()
    {
        Frame f=new Frame();
        f.setSize(600,400);
        f.setBackground(Color.pink);
        f.setLayout(new BorderLayout());

        Panel p1=new Panel();
        p1.setBackground(Color.cyan);
        Label l1 =new Label("Enter Uname");
        TextField tf1=new TextField(15);
        Label l2=new Label("Enter Password");
        TextField tf2=new TextField(15);
        p1.add(l1);
        p1.add(tf1);
        p1.add(l2);
        p1.add(tf2);
        f.add("North",p1);

        Panel p2=new Panel();
        p2.setBackground(Color.yellow);
        Button b1=new Button("Send");
        Button b2=new Button("Clear");
        p2.add(b1);
        p2.add(b2);
        f.add("South",p2);
        f.setVisible(true);
    }
}
```



```
public static void main(String[] args)
{
    PanelFrame pf=new PanelFrame();
}
}
```



Awt

Layout Management

Layout is a logical container used to arrange the GUI component in proper order within the frame, in java.awt package some of the layout is existing as predefined classes as shown below;

- FlowLayout
- BorderLayout
- GridLayout
- GridBagLayout
- CardLayout

FlowLayout

This layout is used to arrange the GUI components in a sequential flow (that means one after another in horizontal way)

You can also set flow layout of components like flow from left, flow from right.

FlowLayout Left

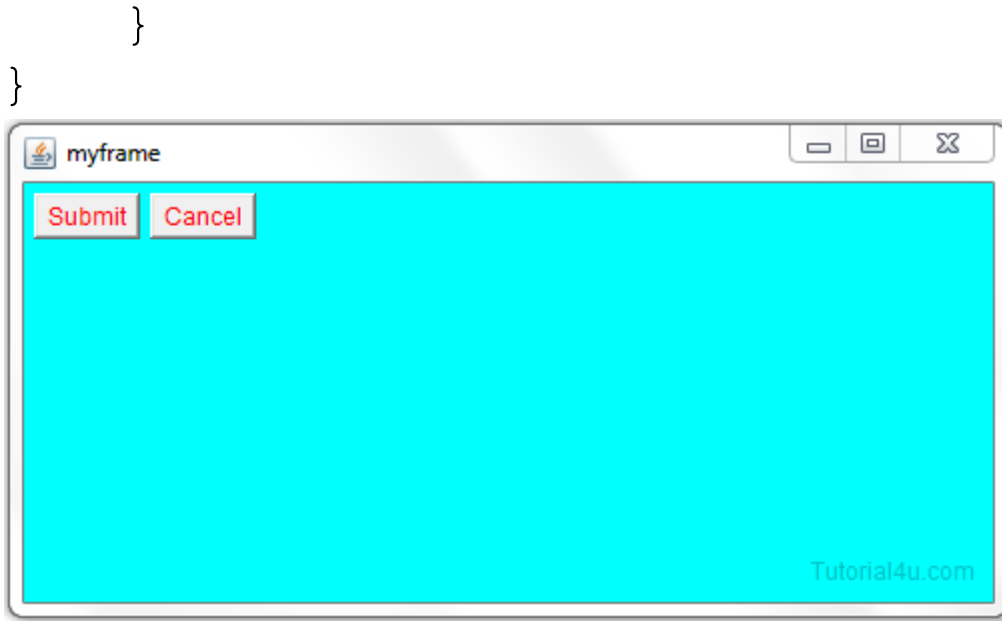
```
Frame f=new Frame();  
f.setLayout(new  
FlowLayout(FlowLayout.LEFT));
```

FlowLayout Right

```
Frame f=new Frame();  
f.setLayout(new  
FlowLayout(FlowLayout.RIGHT));
```

Example of FlowLayout

```
import java.awt.*;  
  
class FlowLayoutDemo  
{  
    public static void main(String[] args)  
    {  
        Frame f=new Frame();  
        f.setTitle("myframe");  
        f.setBackground(Color.cyan);  
        f.setForeground(Color.red);  
        f.setLayout(new  
FlowLayout(FlowLayout.LEFT));  
        Button b1=new Button("Submit");  
        Button b2=new Button("Cancel");  
        f.add(b1);  
        f.add(b2);  
        f.setSize(500,300);  
        f.setVisible(true);  
    }  
}
```



BoarderLayout

This layout is used to arrange the GUI components in S directions of frame as shown below.

Example

```

import java.awt.*;

class BorderDemo
{
    public static void main(String[] args)
    {
        Frame f=new Frame();
        f.setTitle("myframe");
        f.setSize(500,400);
        f.setBackground(Color.white);

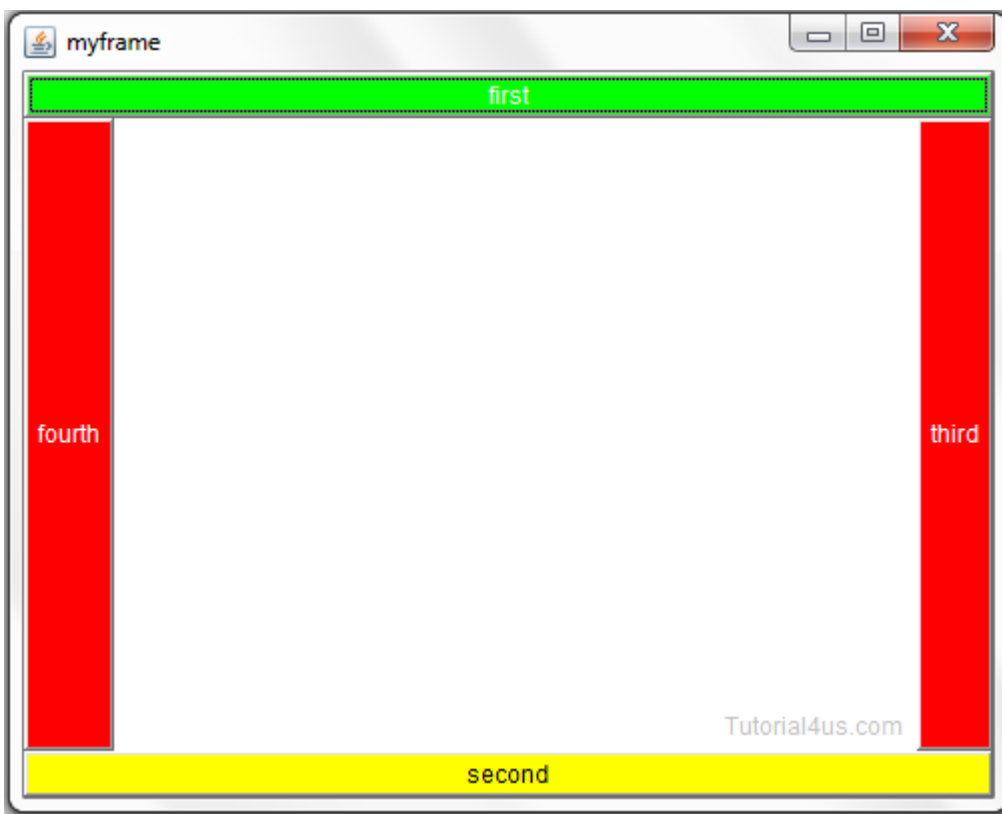
        f.setLayout(new BorderLayout());
        Button b1=new Button("first");
        b1.setBackground(Color.green);
        b1.setForeground(Color.white);
        Button b2=new Button("second");
        b2.setBackground(Color.yellow);
    }
}

```

```

b2.setForeground(Color.black);
Button b3=new Button("third");
b3.setBackground(Color.red);
b3.setForeground(Color.white);
Button b4=new Button("fourth");
b4.setBackground(Color.red);
b4.setForeground(Color.white);
/*Button b5=new Button("center");
b5.setBackground(Color.cyan);
b5.setForeground(Color.white);*/
f.add("North",b1);
f.add("South",b2);
f.add("East",b3);
f.add("West",b4);
//f.add(b5);
f.setVisible(true);
}
}

```



GridLayout

This layout is used to arrange the GUI components in the table format.

Example of GridLayout

```
import java.awt.*;
class Grid extends Frame
{
    Grid()
    {
        Frame f=new Frame();
        f.setTitle("myframe");
        f.setSize(500,400);
        f.setBackground(Color.white);

        f.setLayout(new GridLayout(2,2));
        Button b1=new Button("one");
        b1.setBackground(Color.green);
        b1.setForeground(Color.white);

        Button b2=new Button("two");
        b2.setBackground(Color.yellow);
        b2.setForeground(Color.black);

        Button b3=new Button("three");
        b3.setBackground(Color.red);
        b3.setForeground(Color.white);

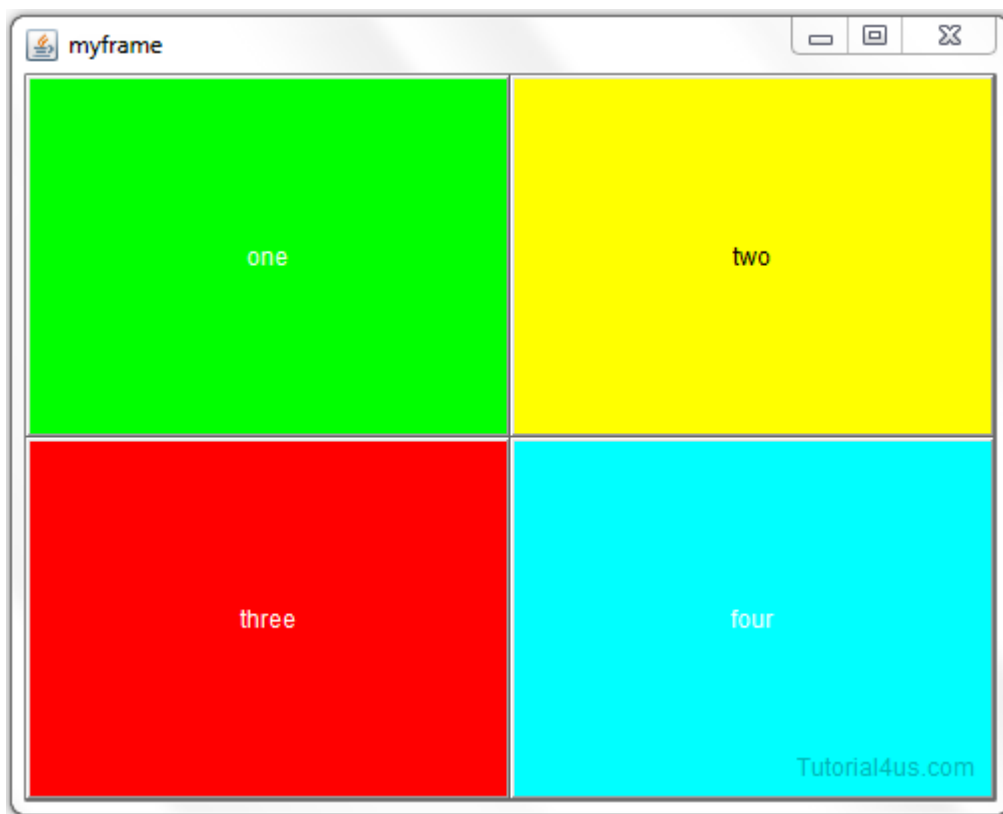
        Button b4=new Button("four");
        b4.setBackground(Color.cyan);
        b4.setForeground(Color.white);
        /*Button b5=new Button("center");
        b5.setBackground(Color.cyan);
        b5.setForeground(Color.white);*/
        f.add("one",b1);
```

```

        f.add("two",b2);
        f.add("three",b3);
        f.add("four",b4);
        //f.add(b5);
        f.setVisible(true);

    }
};
class GridDemo
{
    public static void main(String [] args)
    {
        Grid g1=new Grid();
    }
}

```



CardLayout

This layout is used to arrange multiple cards within the same frame.

Syntax

```
Frame f=new Frame();  
f.setLayout(new CardLayout());
```

GridbagLayout

This layout is used to arrange the GUI components in un-order table structured.

Rules to design gui screen

- Create a frame with specific size
- Set layout to frame
- Add gui component (non-container component) to the frame

Create a frame with specific size

Example of Frame

```
import java.awt.*;  
  
class FrameDemo  
{  
    public static void main(String[] args)  
    {  
        Frame f=new Frame();  
        f.setTitle("myframe");  
        f.setSize(500,300);  
        f.setVisible(true);  
    }  
}
```

Set layout to frame

Example of Frame

```
import java.awt.*;

class FrameDemo
{
public static void main(String[] args)
{
Frame f=new Frame();
f.setTitle("myframe");
f.setLayout(new FlowLayout()); // set
layout
f.setSize(500,300);
f.setVisible(true);
}
}
```

Add gui component (non-container component) to the frame

Example of Frame

```
import java.awt.*;

class FrameDemo
{
public static void main(String[] args)
{
Frame f=new Frame();
f.setTitle("myframe");
f.setLayout(new FlowLayout());
Button b1=new Button("Submit");
Button b2=new Button("Cancel");
f.add(b1); // add button component
f.add(b2); // add button component
f.setSize(500,300);
f.setVisible(true);
}
```

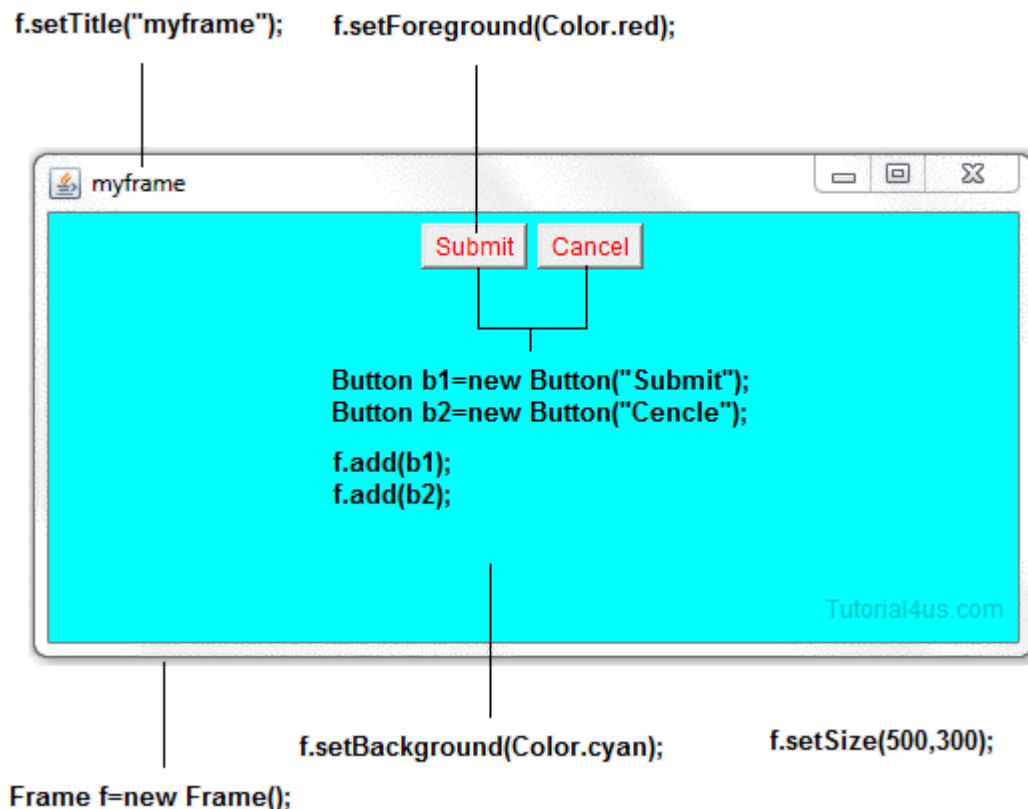


```
}  
}
```

Comple Example of design Gui Screen

Example GUI Screen

```
import java.awt.*;  
  
class FrameDemo  
{  
public static void main(String[] args)  
{  
Frame f=new Frame();  
f.setTitle("myframe");  
f.setBackground(Color.cyan);  
f.setForeground(Color.red);  
f.setLayout(new FlowLayout());  
Button b1=new Button("Submit");  
Button b2=new Button("Cancel");  
f.add(b1);  
f.add(b2);  
f.setSize(500,300);  
f.setVisible(true);  
}  
}
```



Event Handling in AWT

In general you can not perform any operation on dummy GUI screen even any button click or select any item. To perform some operation on these dummy GUI screen you need some predefined classes and interfaces. All these type of classes and interfaces are available in **java.awt.event** package.

Changing the state of an object is known as an **event**.

The process of handling the request in GUI screen is known as **event handling** (event represent an action). It will be changes component to component.

Note: In event handling mechanism event represent an action class and Listener represent an interface. Listener interface always contains abstract methods so here you need to write your own logic.

Event classes and Listener interfaces

Event Classes

Listener Interfaces

ActionEvent

ActionListener

MouseEvent

MouseListener and MouseMotionListener

MouseEvent 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:

- Implement the Listener interface and overrides its methods
- Register the component with the Listener

Syntax to Handle the Event

Example

```
class className implements XXXListener
{
    .....
    .....
}
addcomponentobject.addXXXListener(this);
.....
// override abstract method of given
interface and write proper logic
public void methodName(XXXEvent e)
{
    .....
    .....
}
.....
```

```
}
```

Event Handling for Button

Event handling for button component you need to use `ActionEvent` class and `ActionListener` interface.

`ActionEvent` class and `ActionListener` interface is associated with button component. If any button is clicked operation will be performed by writing the logic in `actionPerform()`.

GUI Component	Event class	Listener Interface	Method (abstract method)
Button	<code>ActionEvent</code>	<code>ActionListener</code>	<code>public void actionPerformed(ActionEvent e)</code>

Example of ActionEvent

```
import java.awt.*;
import java.awt.event.*;

class A implements ActionListener
{
    Frame f;
    Button b1,b2,b3,b4;
    A()
    {
        f=new Frame();
        f.setSize(500,500);
        f.setLayout(new BorderLayout());
        Panel p=new Panel();
        p.setBackground(Color.cyan);
    }
}
```

```

b1=new Button("Red");
b2=new Button("green");
b3=new Button("Blue");
b4=new Button("Exit");
p.add(b1);
p.add(b2);
p.add(b3);
p.add(b4);
f.add("North",p);
b1.addActionListener(this);
b2.addActionListener(this);
b3.addActionListener(this);
b4.addActionListener(this);
f.setVisible(true);
}

public void actionPerformed(ActionEvent e)
{
try
{
if(e.getSource().equals(b1))
{
f.setBackground(Color.red);
}
else if(e.getSource().equals(b2))
{
f.setBackground(Color.green);
}
else if(e.getSource().equals(b3))
{
f.setBackground(Color.blue);
}
else if(e.getSource().equals(b4))
{

```

```
System.exit(0);  
}  
}  
catch (Exception ec)  
{  
System.out.println(ec);  
}  
}  
};
```

```
class ActionEventEx  
{  
public static void main(String[] args)  
{  
A a1=new A();  
}  
}
```



In above example when you click on button then change background color of frame.

Event Handling for TextField

TextEvent class and TextListener interface are associated with both textfield and textarea. If we want to performed any operation, if the value of textfield and textarea are changed than the logic should be written in the textValueChanged().

GUI Component	Event class	Listener Interface	Method (abstract method)
			public void
TextField	TextEvent	TextListener	textValueChanged(TextEvent e)

Example of TextEvent

```
import java.awt.*;  
import java.awt.event.*;  
  
class TextEventEx extends Frame implements  
TextListener  
{  
    TextField tf1,tf2,tf3;  
    Label l1,l2,l3;  
    String nm="",xc="",yc="";  
    int x,y;  
    Panel p1;  
    TextEventEx()  
    {  
        setBackground(Color.cyan);  
        p1=new Panel();  
        p1.setBackground(Color.yellow);  
        l1=new Label("Name");  
        l2=new Label("X-axies");
```



```
l3=new Label("y-axes");
```

```
tf1=new TextField(15);
```

```
tf2=new TextField(5);
```

```
tf3=new TextField(5);
```

```
p1.add(l1);
```

```
p1.add(tf1);
```

```
p1.add(l2);
```

```
p1.add(tf2);
```

```
p1.add(l3);
```

```
p1.add(tf3);
```

```
add("North",p1);
```

```
tf1.addTextListener(this);
```

```
tf2.addTextListener(this);
```

```
tf3.addTextListener(this);
```

```
}
```

```
public void textValueChanged(TextEvent e)
```

```
{
```

```
nm=tf1.getText();
```

```
xc=tf2.getText();
```

```
x=Integer.parseInt(xc);
```

```
yc=tf3.getText();
```

```
y=Integer.parseInt(yc);
```

```
repaint();
```

```
}
```

```
public void paint(Graphics g)
```

```
{
```

```
setFont(new
```

```
Font("TimesRoman",Font.BOLD,30));
```

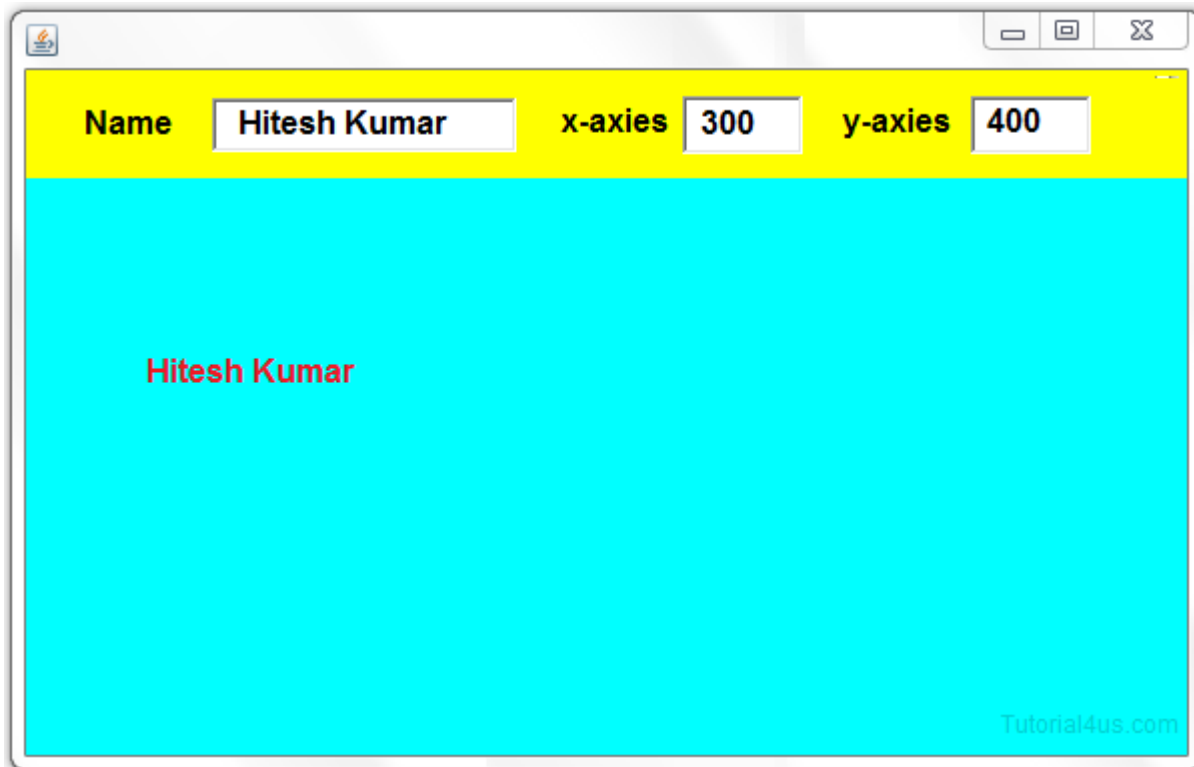
```
g.setColor(Color.red);
```

```

g.drawString("Name"+nm,x,y);
}

public static void main(String args[])
{
TextEventEx t=new TextEventEx();
Toolkit tx=Toolkit .getDefaultToolkit();
t.setSize(tx.getScreenSize());
t.setVisible(true);
}
}

```



Event

Handling for Mouse

For handling event for mouse you need MouseEvent class and MouseListener interface.

GUI Component Event class Listener Interface

Mouse MouseEvent MouseListener

Method (abstract method)

mousePressed(MouseEvent e): This method will be execute whenever mouse button is pressed (not released).

mouseReleased(MouseEvent e): This method will be execute whenever mouse button is only released (if already it is pressed).

mouseClicked(MouseEvent e): This method will be execute whenever mouse button is both pressed and released.

mouseEntered(MouseEvent e): This method will be execute whenever mouse cursor position is placed on specific location or component.

mouseExited(MouseEvent e): This method will be execute whenever mouse cursor position is taken back from any location or component.

Example MouseEvent

```
import java.awt.*;
import java.awt.event.*;

class MouseEventEx extends Frame implements
MouseListener
{
    int x=100,y=200;
    int count=0;

    MouseEventEx ()
    {
        addMouseListener(this);
    }

    public void paint(Graphics g)
```

```

{
count++;
if (count<=3)
{
g.setColor (Color.cyan) ;
}
else
{
g.setColor (Color.red) ;
}
setFont (new
Font ("TimesRoman", Font.BOLD, 30) ) ;
g.drawString ("Hello", x, y) ;
}

public void mouseClicked(MouseEvent e)
{
x=e.getX() ;
y=e.getY() ;
repaint() ;
}
public void mousePresed(MouseEvent e)
{
}
public void mouseReleased(MouseEvent e)
{
}
public void mousePressed(MouseEvent e)
{
}
public void mouseEntered(MouseEvent e)
{
}
public void mouseExited(MouseEvent e)

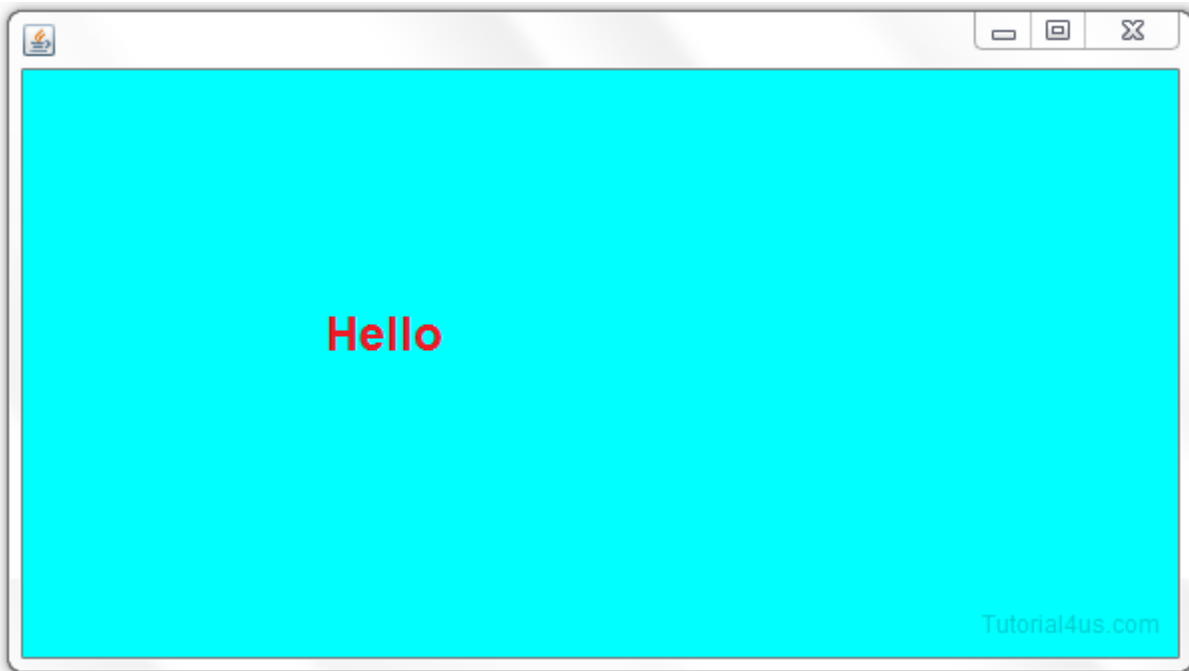
```

```

{
}

public static void main(String[] args)
{
MouseEventEx om=new MouseEventEx();
Toolkit tk=Toolkit.getDefaultToolkit();
om.setSize(tk.getScreenSize());
om.setBackground(Color.cyan);
om.setVisible(true);
}
}

```



Event

Handling for RadioButton, List, Choice

For handling event for RadioButton, List, Choice, checkbox you need to use ItemEvent class and ItemListener interface.

ItemEvent class and ItemListener interface are associated with radio button, checkbox list, choice. This logic can be written in itemStateChanged() whenever we want to perform operation while changed the option in those component.

GUI Component	Event class	Listener Interface	Method (abstract method)
RadioButton , Checkbox, List, Choice	ItemEvent	ItemListener	public void itemStateChanged(ItemEvent e)

Example of ItemEvent

```
import java.awt.*;
import java.awt.event.*;

class A
{
    Frame f;
    Label statusLabel;
    A()
    {
        f=new Frame();
        f.setSize(500,300);
        f.setLayout(new BorderLayout());
        Panel p=new Panel();
        p.setBackground(Color.cyan);

        statusLabel = new Label();
        statusLabel.setAlignment(Label.CENTER);
        statusLabel.setSize(350,100);

        Checkbox chkJava = new Checkbox("Java");
        Checkbox chkHtml = new Checkbox("Html");
        Checkbox chkPHP = new Checkbox("PHP");

        p.add(chkJava);
        p.add(chkHtml);
```

```

p.add(chkPHP);
f.add("North",p);

chkJava.addItemListener(new
CustomItemListener());
chkHtml.addItemListener(new
CustomItemListener());
chkPHP.addItemListener(new
CustomItemListener());

f.add(statusLabel);
f.setVisible(true);

}

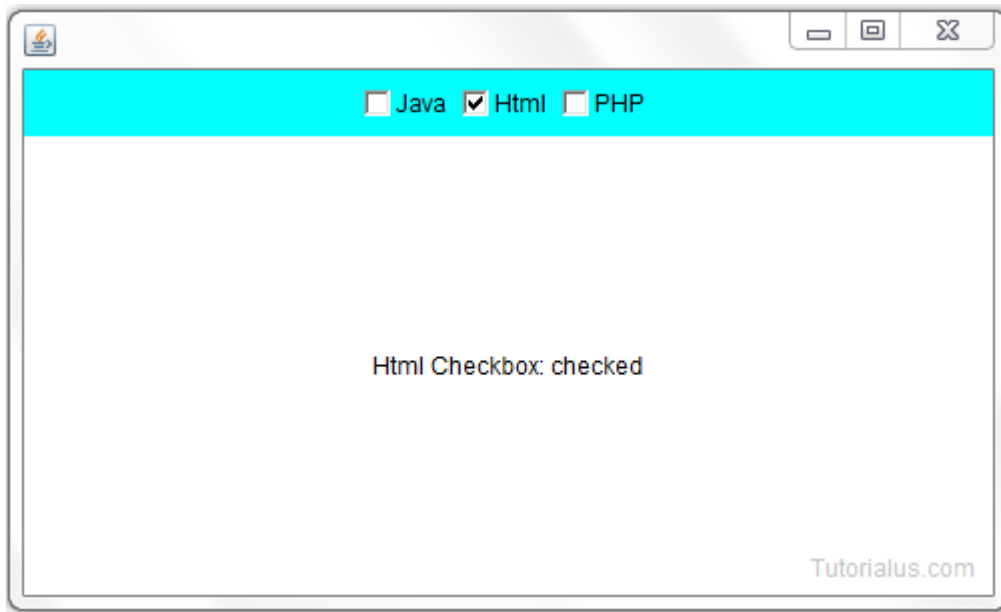
class CustomItemListener implements
ItemListener {
    public void itemStateChanged(ItemEvent
e) {
        statusLabel.setText(e.getItem()
        +" Checkbox: "
        +
        (e.getStateChange()==1?"checked":"unchecked
        "));
    }
}

class ItemEventEx
{

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

```

}



Event handling

for Scrollbar

AdjustmentEvent class and AdjustmentListener interface are used for handling event for scrollbar.

This event class and Listener interface are associated with Scrollbar component adjustmentValueChanged() contains the logic to perform any operation if scrollbar position is changed.

GUI Compon ent	Event class	Listener Interface	Method (abstract method)
Scrollbar	AdjustmentE vent	AdjustmentLis tener	public void adjustmentValueChanged(I temEvent e)

Example of AdjustmentEvent

```
import java.awt.*;  
import java.awt.event.*;
```



```

class AdjustmentEventEx extends Frame
implements AdjustmentListener
{
int rval=0,gval=0,bval=0;
Scrollbar sr,sb,sg;
Panel ps;
AdjustmentEventEx()
{
    ps=new Panel();
    Label l1=new Label("Red");
    Label l2=new Label("Green");
    Label l3=new Label("Blue");
    sr=new
Scrollbar(Scrollbar.HORIZONTAL,0,5,0,255);
    sg=new
Scrollbar(Scrollbar.HORIZONTAL,0,5,0,255);
    sb=new
Scrollbar(Scrollbar.HORIZONTAL,0,5,0,255);
    ps.add(l1);
    ps.add(sr);
    ps.add(l2);
    ps.add(sg);
    ps.add(l3);
    ps.add(sb);
    add("South",ps);
    sr.addAdjustmentListener(this);
    sg.addAdjustmentListener(this);
    sb.addAdjustmentListener(this);
    setSize(400,400);
    setVisible(true);
}
    public void adjustmentValueChanged(
AdjustmentEvent e)
{

```

```

        if (e.getSource().equals(sr))
        {
            rval=sr.getValue();
            setBackground(new
Color(rval,bval,gval));
        }
        if (e.getSource().equals(sg))
        {
            gval=sg.getValue();
            setBackground(new
Color(rval,gval,bval));
        }
        if (e.getSource().equals(sb))
        {
            bval=sb.getValue();
            setBackground(new
Color(rval,gval,bval));
        }
    }

    public static void main(String[] args)
    {
        AdjustmentEventEx ae=new
AdjustmentEventEx();
    }
}

```

Event Handling for Window

Event handling for window you need WindowEvent and WindowListener interface.

Event class Listener Interface

WindowEvent WindowListener

Method (abstract method)

public void windowOpened(WindowEvent e): This method will be execute if the window is open.

public void windowClosing(WindowEvent e): This method will be execute if the window is going to be closed.

public void windowClosed(WindowEvent e): This method will be execute if the window is already closed.

public void windowActivate(WindowEvent e): This method will be execute if the window is in selected state.

public void windowDeactivated(WindowEvent e): This method will be execute if the window is not in selected state.

Example of ItemEvent

```
import java.awt.*;
import java.awt.event.*;

class A extends Frame
{
    A()
    {
        setTitle("Insurance Managment Sysytem");
        setBackground(Color.cyan);
        setLayout(null);

        addWindowListener(new WindowAdapter()
        {
            public void windowClosing(WindowEvent we)
            {
```

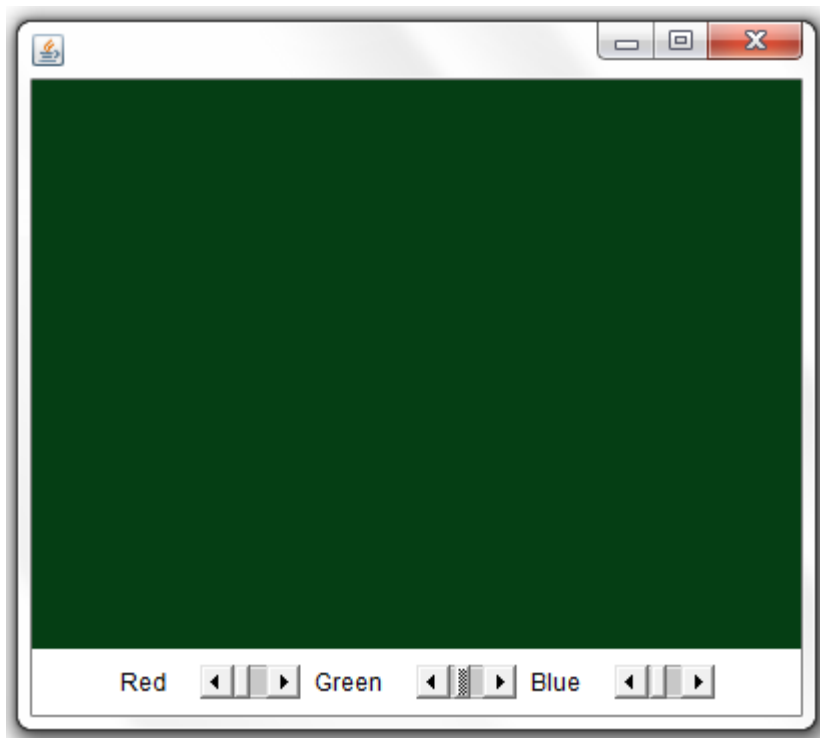
```

        System.exit(0);
    }
}

setSize(500,300);
setVisible(true);
} //constructor
} // class

class WindowEventEx
{
public static void main(String s[])
{
A obj=new A();
}
}

```



- **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.

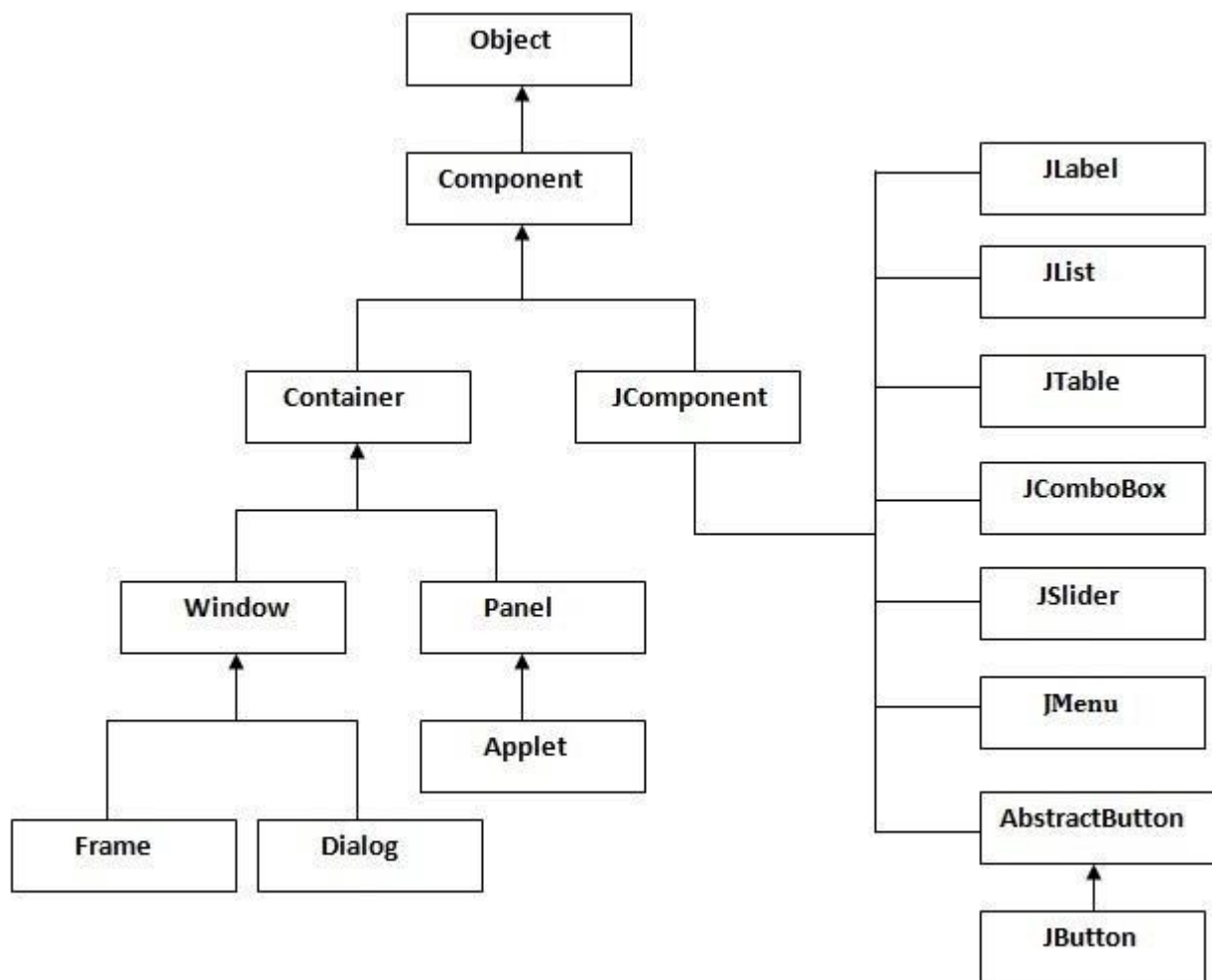
- Unlike AWT, Java Swing provides platform-independent and lightweight components.
- The javax.swing package provides classes for java swing API such as JButton, JTextField, JTextArea, JRadioButton, JCheckbox, JMenu, JColorChooser etc.
-
- **Difference between AWT and Swing**
- There are many differences between java awt and swing that are given below.

No.	Java AWT	Java Swing
1)	AWT components are platform-dependent .	Java swing components are platform-independent .
2)	AWT components are heavyweight .	Swing components are lightweight .
3)	AWT doesn't support pluggable look and feel .	Swing supports pluggable look and feel . Swing provides more powerful components such as tables, lists, scrollpanes, colorchooser, tabbedpane etc.
4)	AWT provides less components than Swing.	
5)	AWT doesn't follows MVC (Model View Controller) where model represents data, view represents presentation and controller acts as an interface between model and view.	Swing follows MVC .
●	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

<code>public void add(Component c)</code>	add a component on another component.
<code>public void setSize(int width,int height)</code>	sets size of the component.
<code>public void setLayout(LayoutManager m)</code>	sets the layout manager for the component.
<code>public void setVisible(boolean b)</code>	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

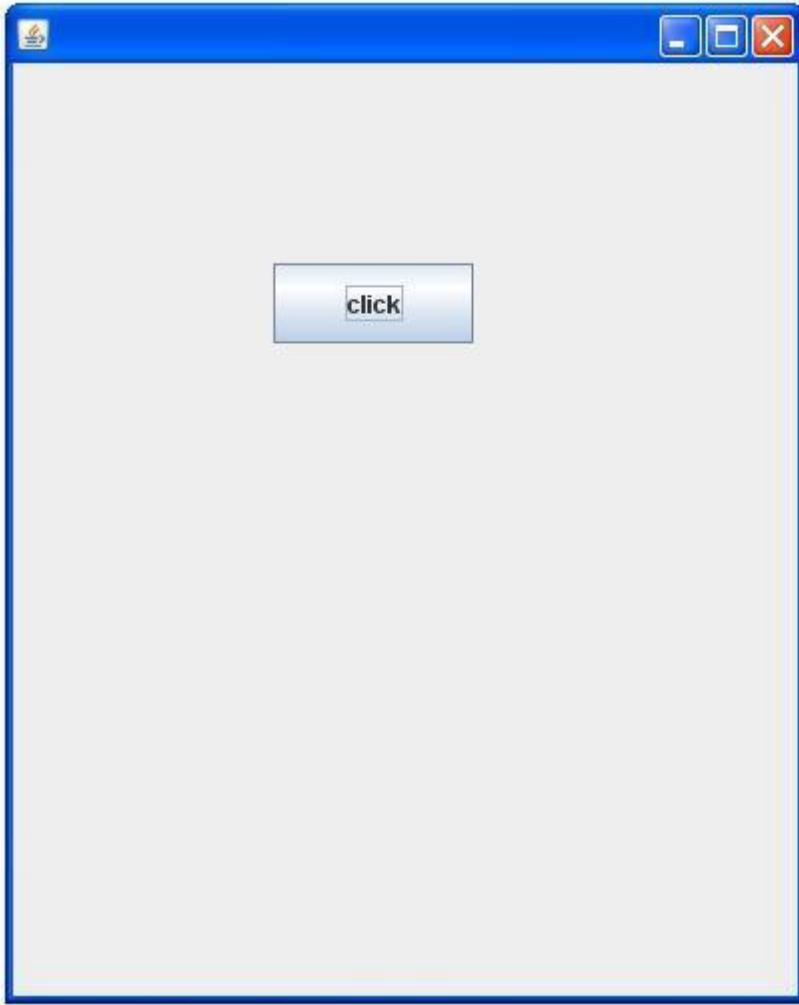
Let's see a simple swing example where we are creating one button and adding it on the `JFrame` object inside the `main()` method.

File: `FirstSwingExample.java`

1. `import javax.swing.*;`
2. `public class FirstSwingExample {`

```
3. public static void main(String[] args) {
4. JFrame f=new JFrame();//creating instance of JFrame
5.
6. JButton b=new JButton("click");//creating instance of JButton
   n
7. b.setBounds(130,100,100, 40);//x axis, y axis, width, height

8.
9. f.add(b);//adding button in JFrame
10.
11. f.setSize(400,500);//400 width and 500 height
12. f.setLayout(null);//using no layout managers
13. f.setVisible(true);//making the frame visible
14. }
15. }
```

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

```
1. import javax.swing.*;  
2. public class Simple {  
3. JFrame f;  
4. Simple(){  
5. f=new JFrame();//creating instance of JFrame  
6.
```

```

7. JButton b=new JButton("click");//creating instance of JButton
   n
8. b.setBounds(130,100,100, 40);
9.
10. f.add(b);//adding button in JFrame
11.
12. f.setSize(400,500);//400 width and 500 height
13. f.setLayout(null);//using no layout managers
14. f.setVisible(true);//making the frame visible
15. }
16.
17. public static void main(String[] args) {
18.     new Simple();
19. }
20. }

```

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`

```

1. import javax.swing.*;
2. public class Simple2 extends JFrame{//inheriting JFrame
3. JFrame f;
4. Simple2(){
5. JButton b=new JButton("click");//create button
6. b.setBounds(130,100,100, 40);

```

```
7.  
8.add(b);//adding button on frame  
9.setSize(400,500);  
10. setLayout(null);  
11. setVisible(true);  
12. }  
13. public static void main(String[] args) {  
14. new Simple2();  
15. }}
```

- JButton class
- JRadioButton class
- JTextArea class
- JComboBox class
- JTable class
- JColorChooser class
- JProgressBar class
- JSlider class
- Digital Watch
- Graphics in swing
- Displaying image
- Edit menu code for Notepad
- OpenFileDialog Box
- Notepad
- Puzzle Game
- Pic Puzzle Game
- Tic Tac Toe Game
- BorderLayout
- GridLayout
- FlowLayout
- CardLayout

Java JButton

The JButton class is used to create a labeled button that has platform independent implementation. The application result in some action when the button is pushed. It inherits AbstractButton class.

JButton class declaration

Let's see the declaration for javax.swing.JButton class.

1. public class JButton extends AbstractButton implements Accessible

Commonly used Constructors:

Constructor	Description
JButton()	It creates a button with no text and icon.
JButton(String s)	It creates a button with the specified text.
JButton(Icon i)	It creates a button with the specified icon object.

Commonly used Methods of AbstractButton class:

Methods	Description
void setText(String s)	It is used to set specified text on button
String getText()	It is used to return the text of the button.

`void setEnabled(boolean b)`

It is used to enable or disable the button.

`void setIcon(Icon b)`

It is used to set the specified Icon on the button.

`Icon getIcon()`

It is used to get the Icon of the button.

`void setMnemonic(int a)`

It is used to set the mnemonic on the button.

`void`

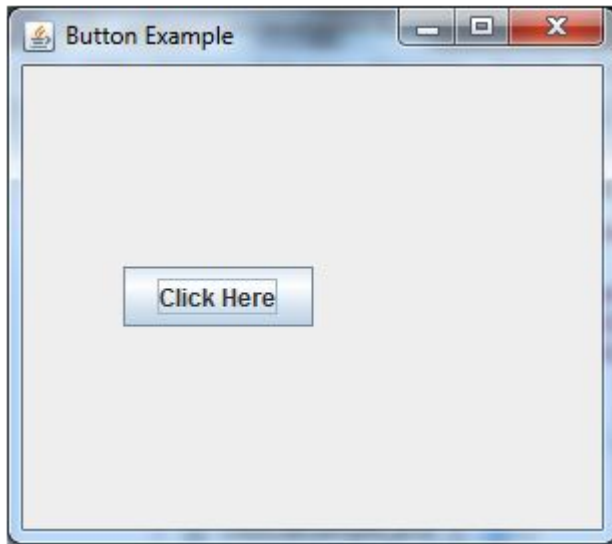
It is used to add the action

`addActionListener(ActionListener a)` listener to this object.

Java JButton Example

```
1. import javax.swing.*;  
2. public class ButtonExample {  
3.     public static void main(String[] args) {  
4.         JFrame f=new JFrame("Button Example");  
5.         JButton b=new JButton("Click Here");  
6.         b.setBounds(50,100,95,30);  
7.         f.add(b);  
8.         f.setSize(400,400);  
9.         f.setLayout(null);  
10.        f.setVisible(true);  
11.    }  
12. }
```

Output:



Java JButton Example with ActionListener

```
1. import java.awt.event.*;
2. import javax.swing.*;
3. public class ButtonExample {
4.     public static void main(String[] args) {
5.         JFrame f=new JFrame("Button Example");
6.         final JTextField tf=new JTextField();
7.         tf.setBounds(50,50, 150,20);
8.         JButton b=new JButton("Click Here");
9.         b.setBounds(50,100,95,30);
10.        b.addActionListener(new ActionListener(){
11.            public void actionPerformed(ActionEvent e){
12.                tf.setText("Welcome to Javatpoint.");
13.            }
14.        });
15.        f.add(b);f.add(tf);
16.        f.setSize(400,400);
17.        f.setLayout(null);
18.        f.setVisible(true);
19.    }
```

20. }

Output:

Java JTable

The JTable class is used to display data in tabular form. It is composed of rows and columns.

JTable class declaration

Let's see the declaration for javax.swing.JTable class.

Commonly used Constructors:

Constructor	Description
JTable()	Creates a table with empty cells.
JTable(Object[][] rows, Object[] columns)	Creates a table with the specified data.

Java JTable Example

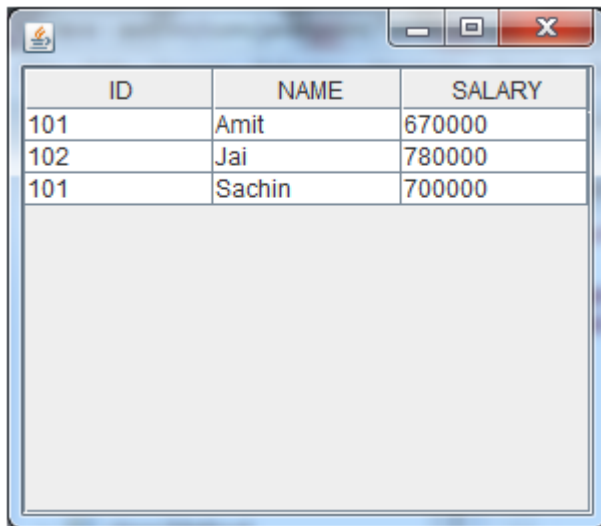
```
1. import javax.swing.*;  
2. public class TableExample {  
3.     JFrame f;  
4.     TableExample(){  
5.         f=new JFrame();  
6.         String data[][]={ {"101","Amit","670000"},
```

```

7.          {"102","Jai","780000"},
8.          {"101","Sachin","700000"};
9.  String column[]={"ID","NAME","SALARY"};
10.  JTable jt=new JTable(data,column);
11.  jt.setBounds(30,40,200,300);
12.  JScrollPane sp=new JScrollPane(jt);
13.  f.add(sp);
14.  f.setSize(300,400);
15.  f.setVisible(true);
16. }
17. public static void main(String[] args) {
18.     new TableExample();
19. }
20. }

```

Output:



ID	NAME	SALARY
101	Amit	670000
102	Jai	780000
101	Sachin	700000

Java JTable Example with ListSelectionListener

```

1. import javax.swing.*;
2. import javax.swing.event.*;
3. public class TableExample {

```



```

4.    public static void main(String[] a) {
5.        JFrame f = new JFrame("Table Example");
6.        String data[][]={ {"101","Amit","670000"},
7.                               {"102","Jai","78000
    0"},
8.                               {"101","Sachin","70
    0000"}}};
9.        String column[]={"ID","NAME","SALARY"};

10.           final JTable jt=new JTable(data,column);
11.        jt.setCellSelectionEnabled(true);
12.        ListSelectionModel select= jt.getSelectionModel();

13.        select.setSelectionMode(ListSelectionModel.SINGL
    E_SELECTION);
14.        select.addListSelectionListener(new ListSelectionLi
    stener() {
15.            public void valueChanged(ListSelectionEvent e) {
16.                String Data = null;
17.                int[] row = jt.getSelectedRows();
18.                int[] columns = jt.getSelectedColumns();
19.                for (int i = 0; i < row.length; i++) {
20.                    for (int j = 0; j < columns.length; j++) {
21.                        Data = (String) jt.getValueAt(row[i], columns[j]
    );
22.                    } }
23.                System.out.println("Table element selected is: "
    + Data);
24.            }
25.        });
26.        JScrollPane sp=new JScrollPane(jt);

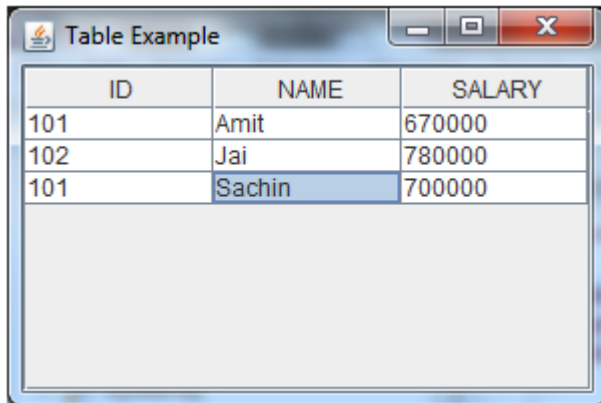
```

```

27.      f.add(sp);
28.      f.setSize(300, 200);
29.      f.setVisible(true);
30.  }
31.  }

```

Output:



ID	NAME	SALARY
101	Amit	670000
102	Jai	780000
101	Sachin	700000

If you select an element in column **NAME**, name of the element will be displayed on the console:

1. Table element selected is: Sachin

Java JList

The object of JList class represents a list of text items. The list of text items can be set up so that the user can choose either one item or multiple items. It inherits JComponent class.

JList class declaration

Let's see the declaration for javax.swing.JList class.

1. public class JList extends JComponent implements Scrollable, Accessible

Commonly used Constructors:

Constructor	Description
JList()	Creates a JList with an empty, read-only, model.
JList(ary[] listData)	Creates a JList that displays the elements in the specified array.
JList(ListModel<ary> dataModel)	Creates a JList that displays elements from the specified, non-null, model.

Commonly used Methods:

Methods	Description
Void addListSelectionListener(ListSelectionListener listener)	It is used to add a listener to the list, to be notified each time a change to the selection occurs.
int getSelectedIndex()	It is used to return the smallest selected cell index.
ListModel getModel()	It is used to return the data model that holds

a list of items
displayed by the
JList component.

It is used to
create a
read-only
ListModel from
an array of
objects.

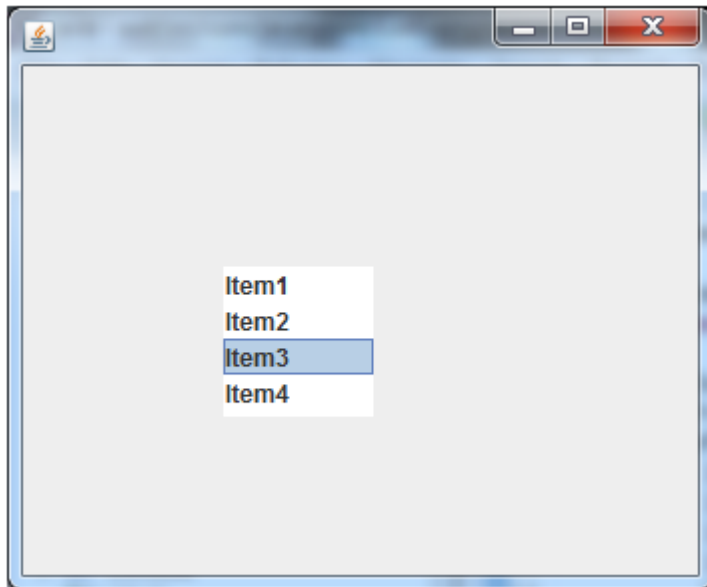
```
void setListData(Object[] listData)
```

Java JList Example

```
1. import javax.swing.*;  
2. public class ListExample  
3. {  
4.     ListExample(){  
5.         JFrame f= new JFrame();  
6.         DefaultListModel<String> l1 = new DefaultListModel<>(  
7.             );  
8.         l1.addElement("Item1");  
9.         l1.addElement("Item2");  
10.        l1.addElement("Item3");  
11.        l1.addElement("Item4");  
12.        JList<String> list = new JList<>(l1);  
13.        list.setBounds(100,100, 75,75);  
14.        f.add(list);  
15.        f.setSize(400,400);  
16.        f.setLayout(null);  
17.        f.setVisible(true);  
18.    }
```

```
18. public static void main(String args[])
19.     {
20.         new ListExample();
21.     }
```

Output:



Java JList Example with ActionListener

```
1. import javax.swing.*;
2. import java.awt.event.*;
3. public class ListExample
4. {
5.     ListExample(){
6.         JFrame f= new JFrame();
7.         final JLabel label = new JLabel();
8.         label.setSize(500,100);
9.         JButton b=new JButton("Show");
10.        b.setBounds(200,150,80,30);
11.        final DefaultListModel<String> l1 = new DefaultListM
        odel<>();
```

```

12.    l1.addElement("C");
13.    l1.addElement("C++");
14.    l1.addElement("Java");
15.    l1.addElement("PHP");
16.    final JList<String> list1 = new JList<>(l1);
17.    list1.setBounds(100,100, 75,75);
18.    DefaultListModel<String> l2 = new DefaultListModel
    <>();
19.    l2.addElement("Turbo C++");
20.    l2.addElement("Struts");
21.    l2.addElement("Spring");
22.    l2.addElement("YII");
23.    final JList<String> list2 = new JList<>(l2);
24.    list2.setBounds(100,200, 75,75);
25.    f.add(list1); f.add(list2); f.add(b); f.add(label);
26.    f.setSize(450,450);
27.    f.setLayout(null);
28.    f.setVisible(true);
29.    b.addActionListener(new ActionListener() {
30.        public void actionPerformed(ActionEvent e) {
31.            String data = "";
32.            if (list1.getSelectedIndex() != -1) {
33.                data = "Programming language Selected: " + lis
t1.getSelectedValue();
34.                label.setText(data);
35.            }
36.            if(list2.getSelectedIndex() != -1){
37.                data += ", FrameWork Selected: ";
38.                for(Object frame :list2.getSelectedValues()){
39.                    data += frame + " ";
40.                }

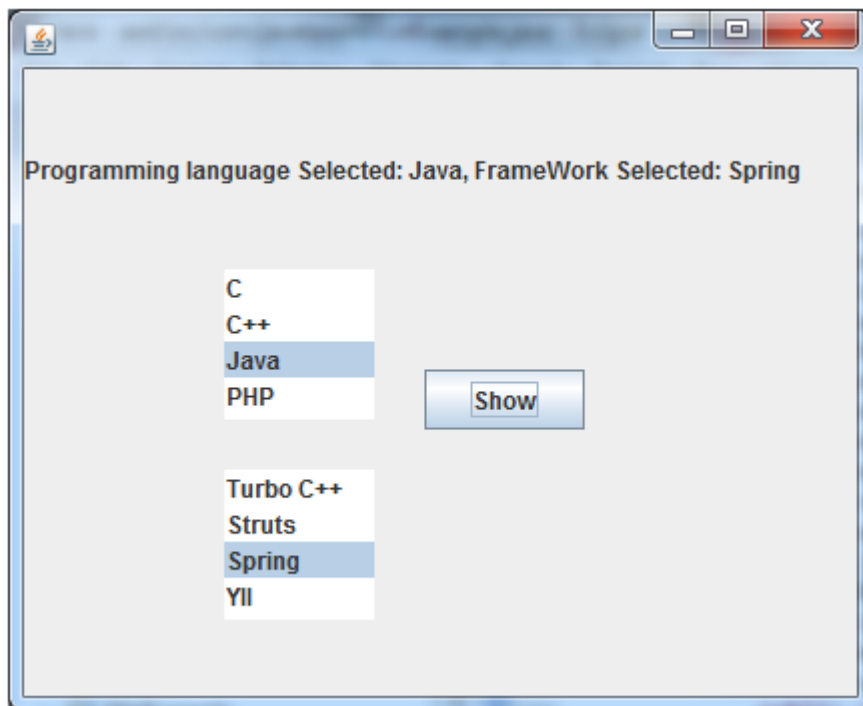
```

```

41.         }
42.         label.setText(data);
43.     }
44. });
45. }
46. public static void main(String args[])
47. {
48.     new ListExample();
49. }

```

Output:



Java JList

The object of JList class represents a list of text items. The list of text items can be set up so that the user can choose either one item or multiple items. It inherits JComponent class.

JList class declaration

Let's see the declaration for javax.swing.JList class.

1. public class JList extends JComponent implements Scrollable, Accessible

Commonly used Constructors:

Constructor	Description
JList()	Creates a JList with an empty, read-only, model.
JList(ary[] listData)	Creates a JList that displays the elements in the specified array.
JList(ListModel<ary> dataModel)	Creates a JList that displays elements from the specified, non-null, model.

Commonly used Methods:

Methods	Description
Void addListSelectionListener(ListSelectionListener listener)	It is used to add a listener to the list, to be notified each time a change to the selection occurs.
int getSelectedIndex()	It is used to return the smallest selected cell index.

ListModel getModel()

It is used to return the data model that holds a list of items displayed by the JList component.

void setListData(Object[] listData)

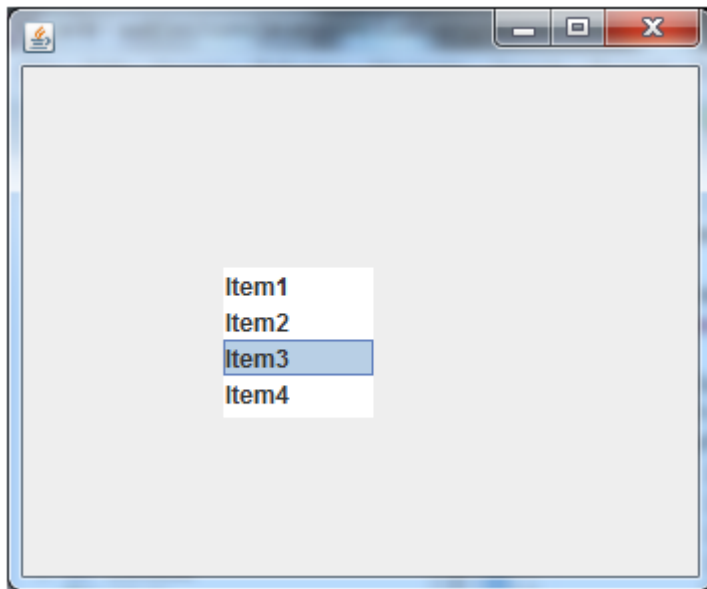
It is used to create a read-only ListModel from an array of objects.

Java JList Example

```
1. import javax.swing.*;
2. public class ListExample
3. {
4.     ListExample(){
5.         JFrame f= new JFrame();
6.         DefaultListModel<String> l1 = new DefaultListModel<>(
7.             );
8.         l1.addElement("Item1");
9.         l1.addElement("Item2");
10.        l1.addElement("Item3");
11.        l1.addElement("Item4");
12.        JList<String> list = new JList<>(l1);
13.        list.setBounds(100,100, 75,75);
14.        f.add(list);
```

```
14.      f.setSize(400,400);
15.      f.setLayout(null);
16.      f.setVisible(true);
17.  }
18.  public static void main(String args[])
19.  {
20.      new ListExample();
21.  }}
```

Output:



Java JList Example with ActionListener

```
1. import javax.swing.*;
2. import java.awt.event.*;
3. public class ListExample
4. {
5.     ListExample(){
6.         JFrame f= new JFrame();
7.         final JLabel label = new JLabel();
8.         label.setSize(500,100);
```

```

9.      JButton b=new JButton("Show");
10.     b.setBounds(200,150,80,30);
11.     final DefaultListModel<String> l1 = new DefaultListM
        odel<>();
12.     l1.addElement("C");
13.     l1.addElement("C++");
14.     l1.addElement("Java");
15.     l1.addElement("PHP");
16.     final JList<String> list1 = new JList<>(l1);
17.     list1.setBounds(100,100, 75,75);
18.     DefaultListModel<String> l2 = new DefaultListModel
        <>();
19.     l2.addElement("Turbo C++");
20.     l2.addElement("Struts");
21.     l2.addElement("Spring");
22.     l2.addElement("YII");
23.     final JList<String> list2 = new JList<>(l2);
24.     list2.setBounds(100,200, 75,75);
25.     f.add(list1); f.add(list2); f.add(b); f.add(label);
26.     f.setSize(450,450);
27.     f.setLayout(null);
28.     f.setVisible(true);
29.     b.addActionListener(new ActionListener() {
30.         public void actionPerformed(ActionEvent e) {
31.             String data = "";
32.             if (list1.getSelectedIndex() != -1) {
33.                 data = "Programming language Selected: " + lis
                    t1.getSelectedValue();
34.                 label.setText(data);
35.             }
36.             if(list2.getSelectedIndex() != -1){

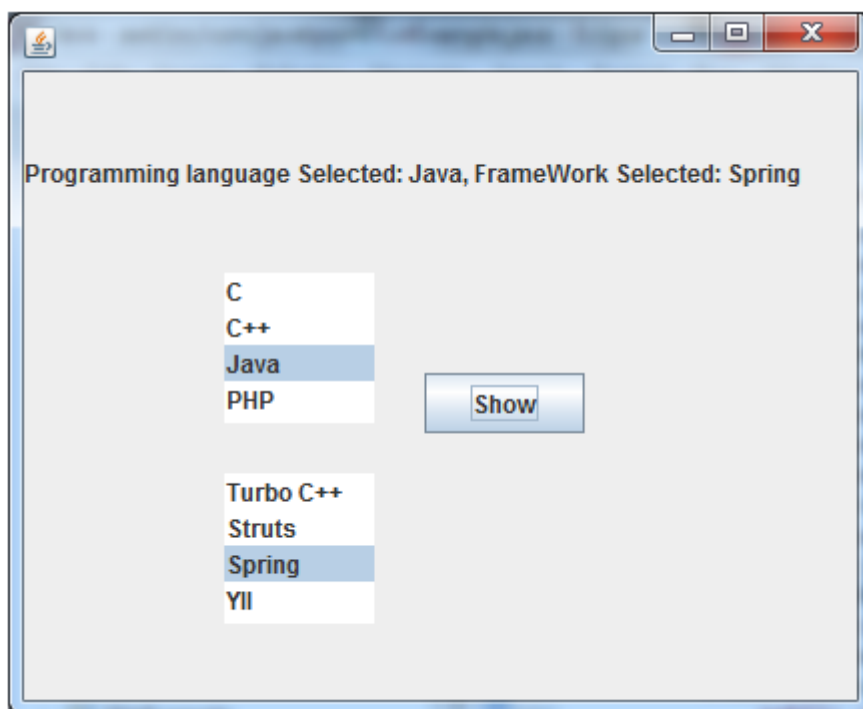
```

```

37.         data += ", FrameWork Selected: ";
38.         for(Object frame :list2.getSelectedValues()){
39.             data += frame + " ";
40.         }
41.     }
42.     label.setText(data);
43. }
44. });
45. }
46. public static void main(String args[])
47. {
48.     new ListExample();
49. }

```

Output:



Java JOptionPane

The JOptionPane class is used to provide standard dialog boxes such as message dialog box, confirm dialog box and input dialog

box. These dialog boxes are used to display information or get input from the user. The JOptionPane class inherits JComponent class.

JOptionPane class declaration

1. public class JOptionPane extends JComponent implements Accessible

Common Constructors of JOptionPane class

Constructor	Description
JOptionPane()	It is used to create a JOptionPane with a test message.
JOptionPane(Object message)	It is used to create an instance of JOptionPane to display a message.
JOptionPane(Object message, int messageType)	It is used to create an instance of JOptionPane to display a message with specified message type and default options.

Common Methods of JOptionPane class

Methods	Description
JDialog createDialog(String title)	It is used to create and return a new parentless JDialog with the specified title.

static void showMessageDialog(Component parentComponent, Object message)	It is used to create an information-message dialog titled "Message".
static void showMessageDialog(Component parentComponent, Object message, String title, int messageType)	It is used to create a message dialog with given title and messageType.
static int showConfirmDialog(Component parentComponent, Object message)	It is used to create a dialog with the options Yes, No and Cancel; with the title, Select an Option.
static String showInputDialog(Component parentComponent, Object message)	It is used to show a question-message dialog requesting input from the user parented to parentComponent.
void setInputValue(Object newValue)	It is used to set the input value that was selected or input by the user.

Java JOptionPane Example: showMessageDialog()

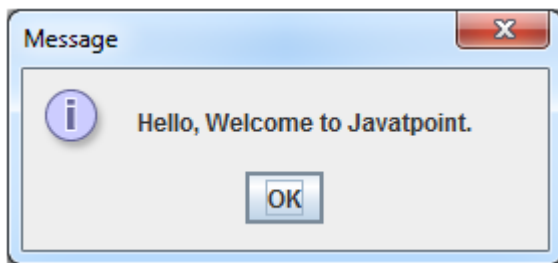
1. import javax.swing.*;
2. public class OptionPaneExample {
3. JFrame f;
4. OptionPaneExample(){

```

5.  f=new JFrame();
6.  JOptionPane.showMessageDialog(f,"Hello, Welcome to Javatpoint.");
7. }
8. public static void main(String[] args) {
9.    new OptionPaneExample();
10. }
11. }

```

Output:



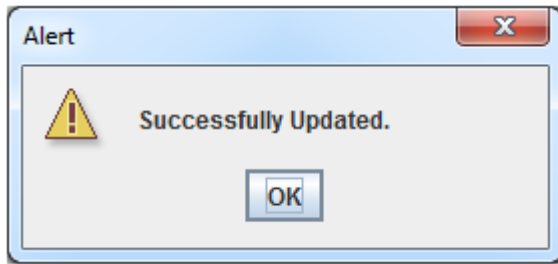
Java JOptionPane Example: showMessageDialog()

```

1. import javax.swing.*;
2. public class OptionPaneExample {
3. JFrame f;
4. OptionPaneExample(){
5.    f=new JFrame();
6.    JOptionPane.showMessageDialog(f,"Successfully Updated
    .","Alert",JOptionPane.WARNING_MESSAGE);
7. }
8. public static void main(String[] args) {
9.    new OptionPaneExample();
10. }
11. }

```

Output:



Java JScrollbar

The object of JScrollbar class is used to add horizontal and vertical scrollbar. It is an implementation of a scrollbar. It inherits JComponent class.

JScrollbar class declaration

Let's see the declaration for javax.swing.JScrollbar class.

1. public class JScrollbar extends JComponent implements Adjustable, Accessible

Commonly used Constructors:

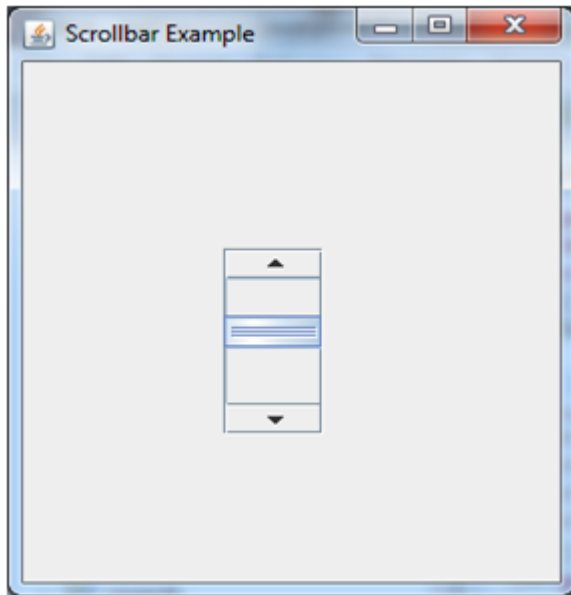
Constructor	Description
JScrollbar()	Creates a vertical scrollbar with the initial values.
JScrollbar(int orientation)	Creates a scrollbar with the specified orientation and the initial values.

JScrollBar(int orientation, int value, int extent, int min, int max)
Creates a scrollbar with the specified orientation, value, extent, minimum, and maximum.

Java JScrollBar Example

```
1. import javax.swing.*;
2. class ScrollBarExample
3. {
4. ScrollBarExample(){
5.     JFrame f= new JFrame("Scrollbar Example");
6.     JScrollBar s=new JScrollBar();
7.     s.setBounds(100,100, 50,100);
8.     f.add(s);
9.     f.setSize(400,400);
10.    f.setLayout(null);
11.    f.setVisible(true);
12. }
13. public static void main(String args[])
14. {
15.     new ScrollBarExample();
16. }}
```

Output:

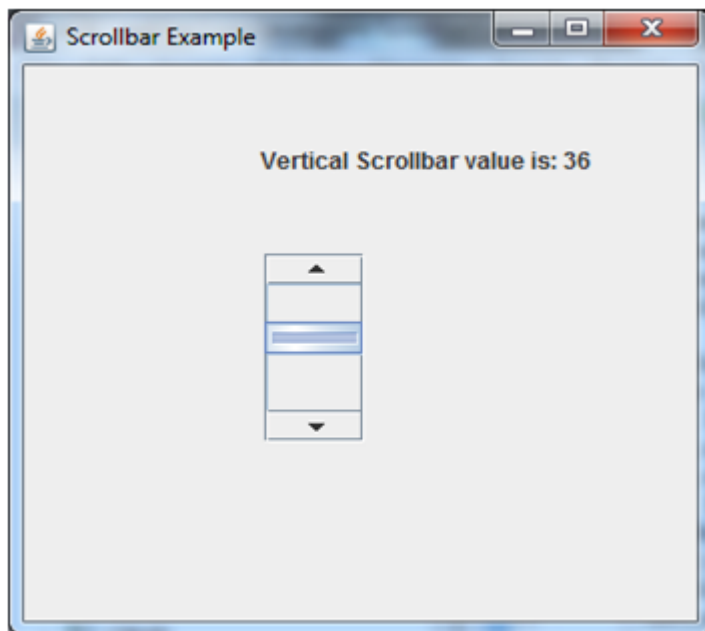


Java JScrollBar Example with AdjustmentListener

```
1. import javax.swing.*;
2. import java.awt.event.*;
3. class ScrollBarExample
4. {
5.     ScrollBarExample(){
6.         JFrame f= new JFrame("Scrollbar Example");
7.         final JLabel label = new JLabel();
8.         label.setHorizontalAlignment(JLabel.CENTER);
9.         label.setSize(400,100);
10.        final JScrollBar s=new JScrollBar();
11.        s.setBounds(100,100, 50,100);
12.        f.add(s); f.add(label);
13.        f.setSize(400,400);
14.        f.setLayout(null);
15.        f.setVisible(true);
16.        s.addAdjustmentListener(new AdjustmentListener() {
17.            public void adjustmentValueChanged(AdjustmentEvent
               e) {
```

```
18.     label.setText("Vertical Scrollbar value is:"+ s.getValue(  
    ));  
19. }  
20. });  
21. }  
22. public static void main(String args[])  
23. {  
24.     new ScrollBarExample();  
25. }}
```

Output:



Java JMenuBar, JMenu and JMenuItem

The JMenuBar class is used to display menubar on the window or frame. It may have several menus.

The object of JMenu class is a pull down menu component which is displayed from the menu bar. It inherits the JMenuItem class.

The object of JMenuItem class adds a simple labeled menu item. The items used in a menu must belong to the JMenuItem or any of its subclass.

JMenuBar class declaration

1. public class JMenuBar extends JComponent implements MenuElement, Accessible

JMenu class declaration

1. public class JMenu extends JMenuItem implements MenuElement, Accessible

JMenuItem class declaration

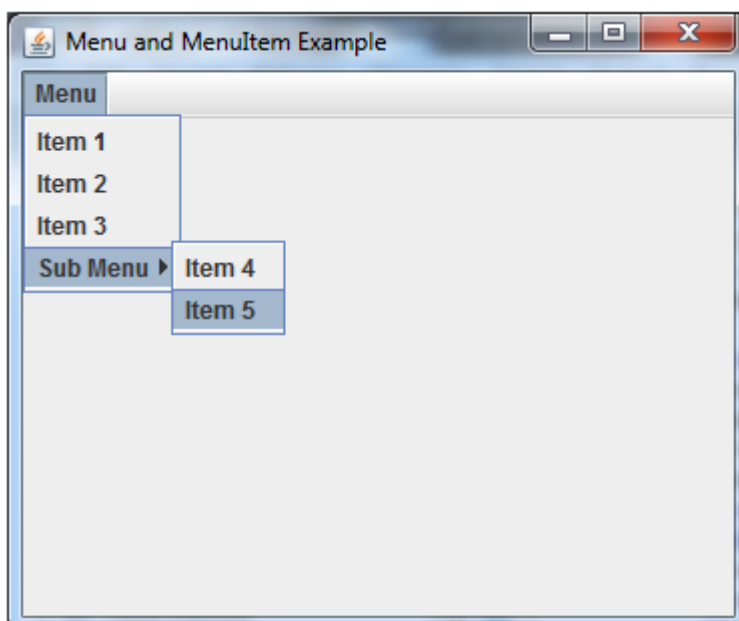
1. public class JMenuItem extends AbstractButton implements Accessible, MenuElement

Java JMenuItem and JMenu Example

1. import javax.swing.*;
2. class MenuExample
3. {
4. JMenu menu, submenu;
5. JMenuItem i1, i2, i3, i4, i5;
6. MenuExample(){
7. JFrame f= new JFrame("Menu and MenuItem Example");
8. JMenuBar mb=new JMenuBar();
9. menu=new JMenu("Menu");

```
10. submenu=new JMenu("Sub Menu");
11. i1=new JMenuItem("Item 1");
12. i2=new JMenuItem("Item 2");
13. i3=new JMenuItem("Item 3");
14. i4=new JMenuItem("Item 4");
15. i5=new JMenuItem("Item 5");
16. menu.add(i1); menu.add(i2); menu.add(i3);
17. submenu.add(i4); submenu.add(i5);
18. menu.add(submenu);
19. mb.add(menu);
20. f.setJMenuBar(mb);
21. f.setSize(400,400);
22. f.setLayout(null);
23. f.setVisible(true);
24. }
25. public static void main(String args[])
26. {
27. new MenuExample();
28. }}
```

Output:

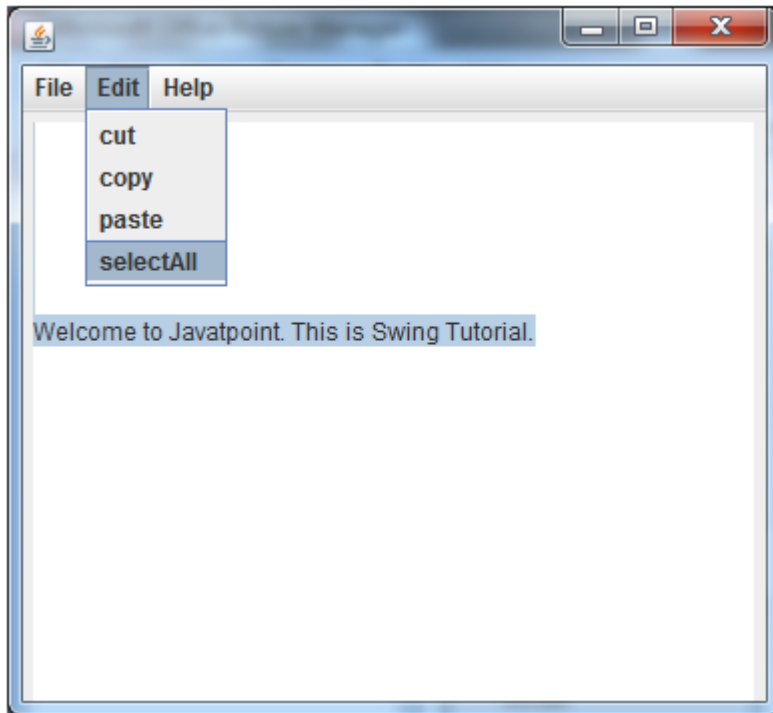


Example of creating Edit menu for Notepad:

```
1. import javax.swing.*;
2. import java.awt.event.*;
3. public class MenuExample implements ActionListener{
4. JFrame f;
5. JMenuBar mb;
6. JMenu file,edit,help;
7. JMenuItem cut,copy,paste,selectAll;
8. JTextArea ta;
9. MenuExample(){
10. f=new JFrame();
11. cut=new JMenuItem("cut");
12. copy=new JMenuItem("copy");
13. paste=new JMenuItem("paste");
14. selectAll=new JMenuItem("selectAll");
15. cut.addActionListener(this);
16. copy.addActionListener(this);
17. paste.addActionListener(this);
18. selectAll.addActionListener(this);
19. mb=new JMenuBar();
20. file=new JMenu("File");
21. edit=new JMenu("Edit");
22. help=new JMenu("Help");
23. edit.add(cut);edit.add(copy);edit.add(paste);edit.add(selectAll);
24. mb.add(file);mb.add(edit);mb.add(help);
25. ta=new JTextArea();
26. ta.setBounds(5,5,360,320);
27. f.add(mb);f.add(ta);
```

```
28. f.setJMenuBar(mb);
29. f.setLayout(null);
30. f.setSize(400,400);
31. f.setVisible(true);
32. }
33. public void actionPerformed(ActionEvent e) {
34.     if(e.getSource()==cut)
35.         ta.cut();
36.     if(e.getSource()==paste)
37.         ta.paste();
38.     if(e.getSource()==copy)
39.         ta.copy();
40.     if(e.getSource()==selectAll)
41.         ta.selectAll();
42. }
43. public static void main(String[] args) {
44.     new MenuExample();
45. }
46. }
```

Output:



Java JPopupMenu

PopupMenu can be dynamically popped up at specific position within a component. It inherits the JComponent class.

JPopupMenu class declaration

Let's see the declaration for javax.swing.JPopupMenu class.

1. public class JPopupMenu extends JComponent implements Accessible, MenuElement

Commonly used Constructors:

Constructor	Description
JPopupMenu()	Constructs a JPopupMenu without an "invoker".

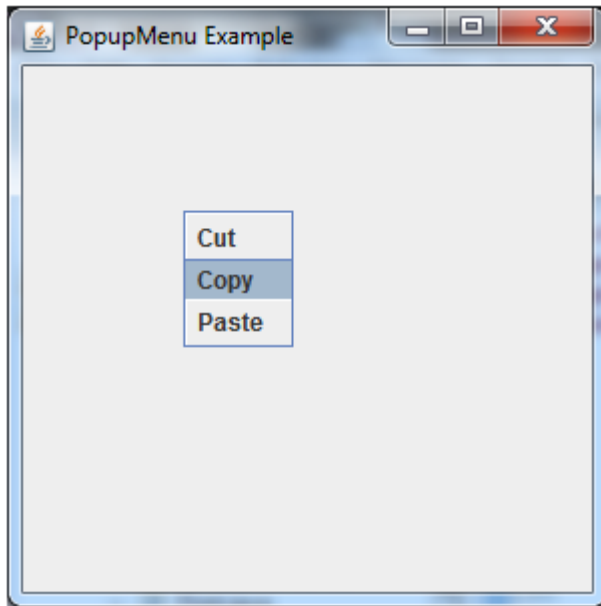
JPopupMenu(String label)	Constructs a JPopupMenu with the specified title.
--------------------------	---

Java JPopupMenu Example

```
1. import javax.swing.*;
2. import java.awt.event.*;
3. class PopupMenuExample
4. {
5.     PopupMenuExample(){
6.         final JFrame f= new JFrame("PopupMenu Example");
7.         final JPopupMenu popupmenu = new JPopupMenu("Edit");
8.         JMenuItem cut = new JMenuItem("Cut");
9.         JMenuItem copy = new JMenuItem("Copy");
10.        JMenuItem paste = new JMenuItem("Paste");
11.        popupmenu.add(cut); popupmenu.add(copy); popupmenu.add(paste);
12.        f.addMouseListener(new MouseAdapter() {
13.            public void mouseClicked(MouseEvent e) {
14.                popupmenu.show(f , e.getX(), e.getY());
15.            }
16.        });
17.        f.add(popupmenu);
18.        f.setSize(300,300);
19.        f.setLayout(null);
20.        f.setVisible(true);
21.    }
22.    public static void main(String args[])
23.    {
```

```
24.     new PopupMenuExample();
25. }}
```

Output:



Java JPopupMenu Example with MouseListener and ActionListener

```
1. import javax.swing.*;
2. import java.awt.event.*;
3. class PopupMenuExample
4. {
5.     PopupMenuExample(){
6.         final JFrame f= new JFrame("PopupMenu Example");
7.         final JLabel label = new JLabel();
8.         label.setHorizontalAlignment(JLabel.CENTER);
9.         label.setSize(400,100);
10.        final JPopupMenu popupmenu = new JPopupMenu(
    "Edit");
11.        JMenuItem cut = new JMenuItem("Cut");
```

```

12.      JMenuItem copy = new JMenuItem("Copy");
13.      JMenuItem paste = new JMenuItem("Paste");
14.      popupmenu.add(cut); popupmenu.add(copy); popu
    pmenu.add(paste);
15.      f.addMouseListener(new MouseAdapter() {
16.          public void mouseClicked(MouseEvent e) {
17.              popupmenu.show(f , e.getX(), e.getY());
18.          }
19.      });
20.      cut.addActionListener(new ActionListener(){
21.          public void actionPerformed(ActionEvent e) {
22.              label.setText("cut MenuItem clicked.");
23.          }
24.      });
25.      copy.addActionListener(new ActionListener(){
26.          public void actionPerformed(ActionEvent e) {

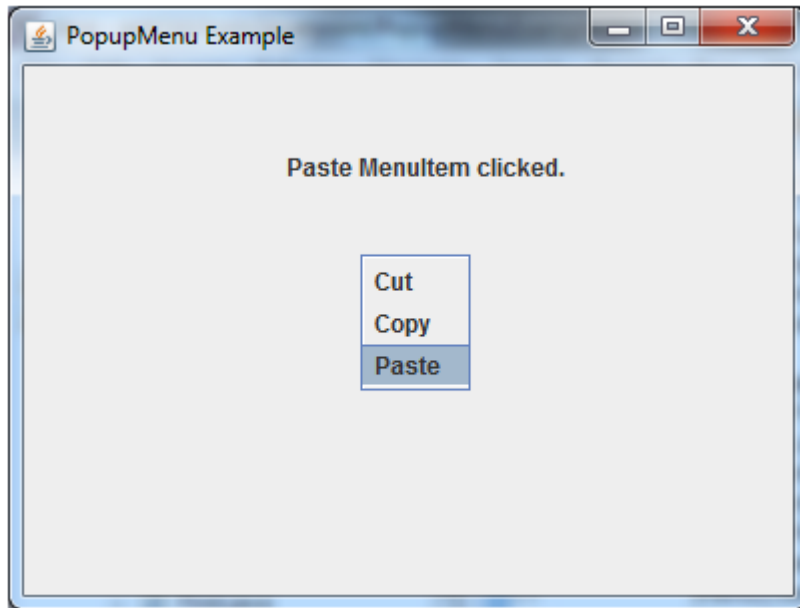
27.              label.setText("copy MenuItem clicked.");
28.          }
29.      });
30.      paste.addActionListener(new ActionListener(){
31.          public void actionPerformed(ActionEvent e) {

32.              label.setText("paste MenuItem clicked.");
33.          }
34.      });
35.      f.add(label); f.add(popupmenu);
36.      f.setSize(400,400);
37.      f.setLayout(null);
38.      f.setVisible(true);
39.  }

```

```
40. public static void main(String args[])
41. {
42.     new PopupMenuExample();
43. }
44. }
```

Output:



Collection Framework in Java

It is a collection of collection classes in the java API. It can be used to handle the data structure in java language or collection classes in java API used to manage the data very efficiently like inserting, deleting, updating, retrieving, sorting the data etc.

It is one of the standardized mechanisms which allow us to group multiple values either of same types or different type or both the types in a single variable with dynamic size in nature. This single variable is known as collection variable.

All the operations that we want to perform on a data such as searching, sorting, insertion, deletion etc. Before storing or

moving the data to permanent memory location, can be performed by Java Collection Framework.

Basic Aim

The basic aim of collection framework is to improve the performance of java based application.

Prerequisites

Before learning of This Tutorial you need to learn basic of Java Programming Like oops concept, Exception handling, object Type casting.

What is Collection Framework

What is Collection ?

A **Collection** is simply an object that groups multiple elements into a single unit.

What is Framework ?

A **Framework** provides ready-made architecture and represents set of classes and interface.

What is Collection Framework ?

It is a unified architecture for representing and manipulating collections. All collections frameworks contain the following:

- Interfaces
- Interface Implementer Classes
- Algorithms

It is a collection of classes in the java API. It can used to handle the data structure in java language or collection classes in java

API used to manage the data very efficiently like inserting, deleting, updating, retrieving, sorting the data etc.

It is one of the standardized mechanisms which allow us to group multiple values either of same types or different type or both the types in a single variable with dynamic size in nature. This single variable is known as collection framework variable.

Where use Collection Framework ?

All the operations that we want to perform on a data such as searching, sorting, insertion, deletion etc. Before storing or moving the data to permanent memory location, first performed by Java Collection Framework.



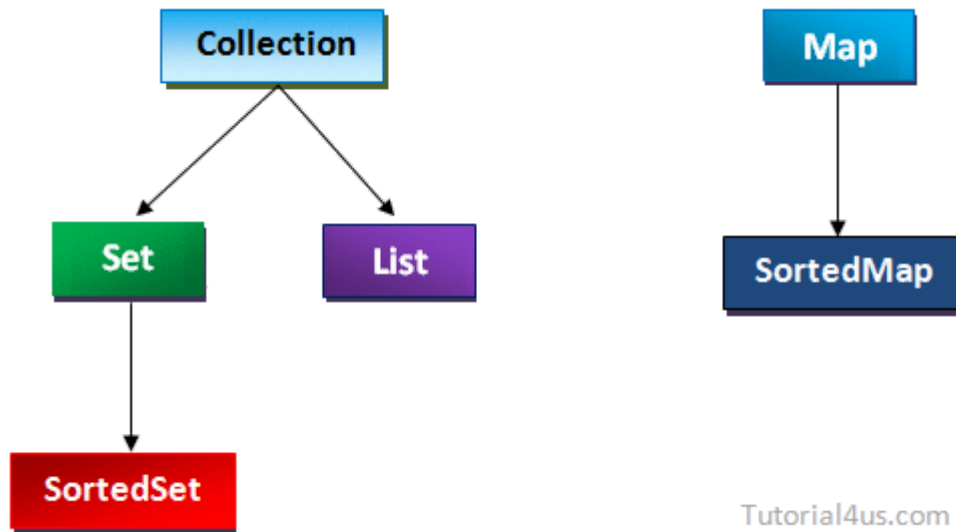
Features	Collection	List	Set	SortedSet
Extends	Collection is top most Collection Framework	List extends Collection	Set extends Collection	SortedSet extends Set
Allow to Store Elements	Store duplicate Elements	Store duplicate Elements	Store Unique Elements	Store Unique Elements
Display Elements	Random Order	Sequential Order	Random Order	Sorted Order
Add Elements	Only at End	Beginning, Middle End	Only at End	Beginning, Middle End
Retrieve Elements	Forward Direction Backward Direction	Forward, Random Backward Direction	Forward Direction	Forward, Random Backward Direction
Tutorial4us.com				

In above images you can see features of all collection classes and interfaces.

Collection Framework

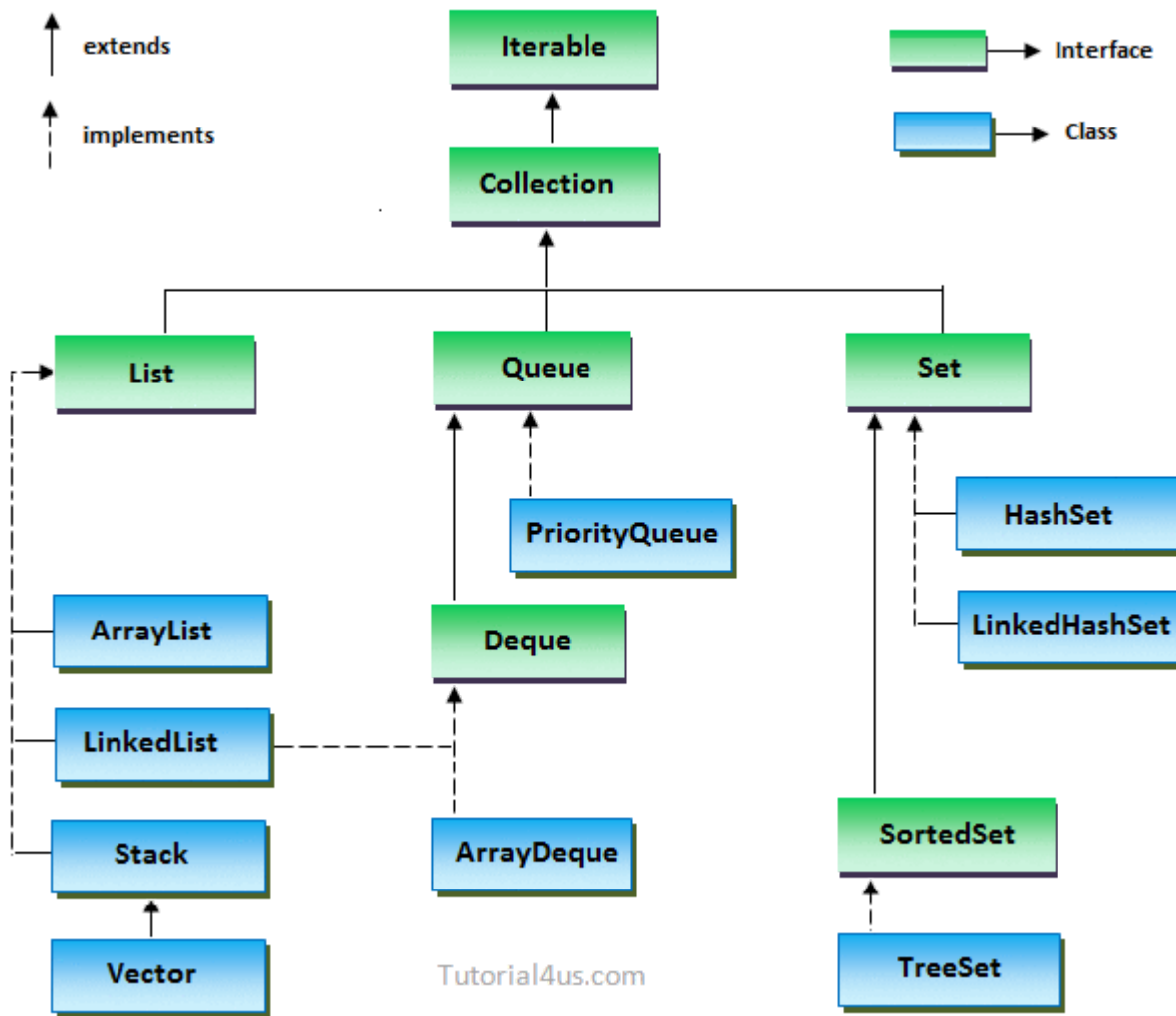
It is one of the standardized mechanisms which allow us to group multiple values either of same types or different type or both the types in a single variable with dynamic size in nature. This single variable is known as collection framework variable.

Collection Framework have collection of classes and interface where Collection interface is top most collection framework. In below images it is defined clearly.



Collection Framework Hierarchy

Almost all collections in Java are derived from the `java.util.Collection` interface



Difference Between Array and Collection

The main advantage of Collection Framework are Collections are grow-able in nature, hence based on our requirement we can increase or decrease the size. Some others advantage are given below;

- Collection is re-sizable or dynamically draw-able memory.
- Provides useful data structures in the form of predefined classes that reduces programming affords.
- It support to store heterogeneous elements or object.
- It provides higher performance.
- It provides Extendability (depends on incoming flow of data,if the size of collection framework variable is increasing than the collection framework variable is containing Extendability feature).

- It provides adaptability facility(The process of adding the content of one collection framework variable to another collection framework either in the beginning or in the ending or in the middle in known as adaptability).
- It is one of the algorithmic oriented.
- It provides in-built sorting technique.
- It provides in-built searching technique.
- It provides higher preliminary concepts of Data Structure such as:- Stack,Queue,LinkedList,Trees ..etc.

Difference between Array and Collection

Array	Collection
1 Arrays are fixed in size and hence once we created an array we are not allowed to increase or decrease the size based on our requirement.	Collections are grow-able in nature and hence based on our requirement we can increase or decrease the size.
2 Arrays can hold both primitives as well as objects.	Collections can hold only objects but not primitive.
3 Performance point of view arrays faster than collection	Performance point of view collections are slower than array
4 Arrays can hold only homogeneous elements.	Collections can hold both homogeneous and heterogeneous elements.

5	Memory point of view arrays are not recommended to use.	Memory point of view collections are recommended to use.
6	For any requirement, there is no ready method available.	For every requirement ready made method support is available.

Note: Above difference is same for difference between Array and any Collection object (like Vector, ArrayList, LinkedList).

Collection Framework API

Collection Framework API is a java API that is contains all Interfaces and classes of Collection Framework which is used for perform searching, sorting, insertion, manipulation, deletion operations.

Package

- java.util package

Interfaces

- Collection
- Set
- List
- Queue
- Deque

Classes

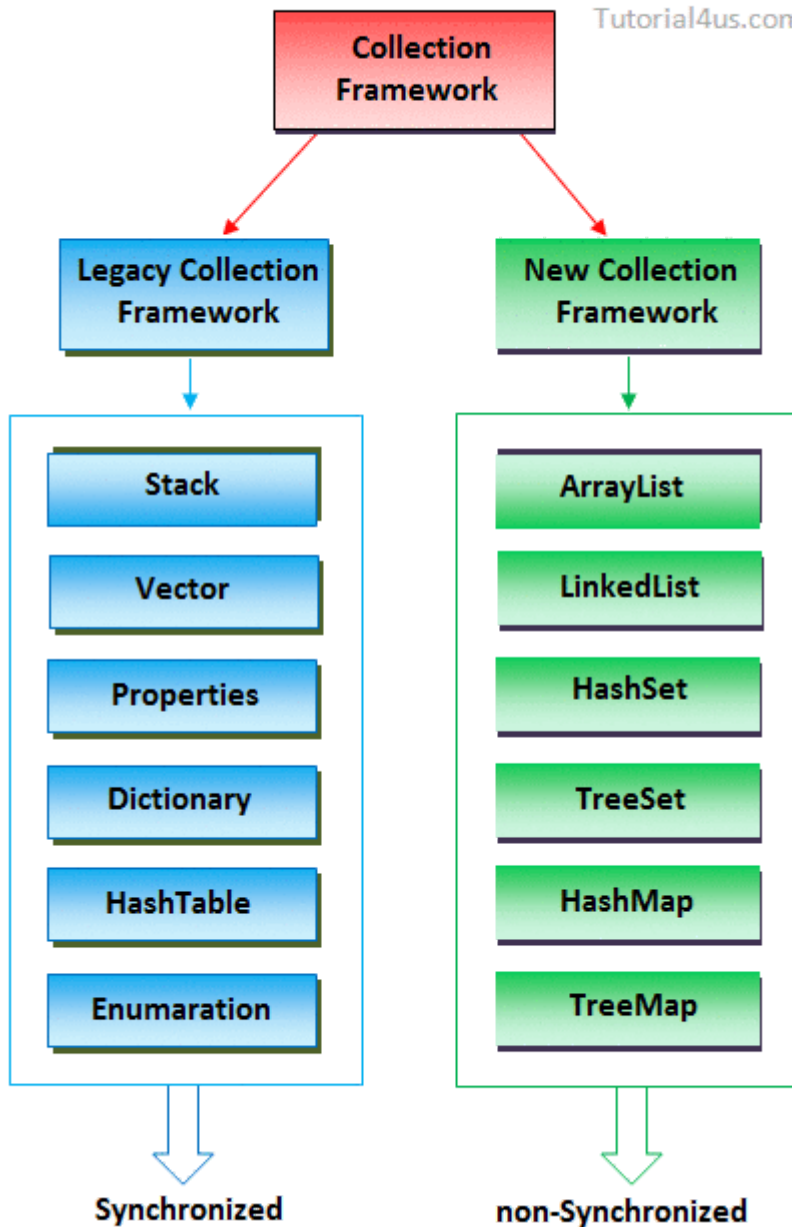
- ArrayList
- Vector

- LinkedList
- PriorityQueue
- HashSet
- LinkedHashSet
- TreeSet

Types of Collection Framework

In the initial version of java the concept of Collection Framework was known as data structures. At that time this concept is unable to full fill all requirement of java application so sun micro-system add some new classes and interfaces in collection framework. Here i will show you old (Legacy Collection Framework) and new (New Collection Framework) both collection framework. Collection Framework are mainly divide in two types they are;

- Legacy Collection Framework
- New Collection Framework



Legacy Collection Framework

Legacy Collection Framework classes and interfaces are Synchronized and for retrieving data from Legacy Collection Framework we use Enumeration. Classes and interfaces are;

- Stack
- Vector
- Directory
- HashTable

- Properties
- Enumeration

Note: Legacy Collection Framework are Synchronized because of this we can not use Legacy Collection Framework for web based applications.

New Collection Framework

New Collection Framework classes and interfaces are non-Synchronized and for retrieving data from new Collection Framework you can use foreach loop, iterator, listiterator. Classes and interfaces are;

- ArrayList
- LinkedList
- HashSet
- TreeSet
- HashMap
- TreeMap

Note: All new Collection Framework are not Synchronized.

Data Retrieving Technique form Collection Framework

Technique to retrieve elements from Collection object

Java support following technique to retrieve the elements from any collection object.

1. foreach loop
2. Iterator interface
3. ListIterator interface
4. Enumeration interface

foreach

It is similar to for loop used to retrieve elements of collection objects (until the last element)

Syntax

```
for (datatype variable:collection-object)
{
    . . . . .
    . . . . .
}
```

Iterator interface

It is one of the predefined interface present in java.util.* package. The purpose of this interface is that to extract or retrieve the elements of collection variable only in forward direction but not in backward direction. By default an object of iterator is pointing just before the first element of any collection framework variable.

Methods of Iterator Interface

- public boolean hasNext()
- public object next()

ListIterator interface

It is one of the predefined interface present in java.util.* package. The ListIterator interface object is always used for retrieving the data from any collection framework either forward direction or backward direction or in both direction. Like Iterator interface object, ListIterator interface object is pointing just before the first element of any collection framework variable.

Methods of ListIterator Interface

- public boolean hasNext()
- public object next()
- public boolean nestPrevious()
- public object previous()

Enumeration interface

It is one of the predefined interface and whose object is always used for retrieving the data from any legacy collection framework variable only in forward direction but not in backward direction. Like Iterator interface object, Enumeration Interface object is pointing just before the first element of any legacy collection framework variable.

The functionality of Enumeration is more or less similar to Iterator Interface but Enumeration Interface object belongs to synchronized and Iterator Interface object belong to non-synchronized.

Methods of Enumeration Interface

- public boolean hasMoreElements()
- public object nextElement()

Foreach Loop in Java

It is similar to for loop used to retrieve elements of collection objects (until the last element)

Syntax

```
for (datatype variable:collection-object)
{
    . . . . .
    . . . . .
}
```


The above looping statement executed repeatedly several number of time until the elements are available in the collection object, the loop will be terminated if no elements found.

Note: foreach loop always traversing in forward direction.

Example of for...each Loop

```
import java.util.*;

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

        ArrayList<Integer> al=new
        ArrayList<Integer>();    // creating
        arraylist
        al.add(10);
        al.add(20);
        al.add(30);

        for(int i : al)
        {
            System.out.println(i);
        }
    }
}
```

Output

```
10
20
30
```

Iterator Interface in Java

It is one of the predefined interface present in `java.util.*` package. The purpose of this interface is that to extract or retrieve the elements of collection variable only in forward direction but not in backward direction. By default an object of iterator is pointing just before the first element of any collection framework variable.

Points to Remember

- Iterator allows to remove the elements from collection object.
- Iterator is not Synchronized.
- Using Iterator elements of collection can be access only in forward direction.
- Iterator can be available to all the collection classes.
- Every collection class contains `Iterator()` and that returns Iterator object, using this object reference elements can be retrieved from collection.

Methods of Iterator Interface

- `public boolean hasNext()`
- `public Object next()`
- `remove()`

`public boolean hasNext()`

This method return true provided by Iterator interface object is having next element otherwise it returns false.

public object next()

This method is used for retrieving next element of any collection framework variable provided public boolean hasNext(). If next elements are available then this method returns true other wise it return false.

public object remove()

Remove from collection the last element returned by Iterator.

Example of Iterator

```
import java.util.*;

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

ArrayList<Integer> al=new
ArrayList<Integer>();    // creating
arraylist
al.add(10);
al.add(20);
al.add(30);

Iterator itr=al.iterator();    // getting
Iterator from arraylist to traverse
elements
while(itr.hasNext())
{
System.out.println(itr.next());
}
}
```

```
}
```

Output

```
10  
20  
30
```

ListIterator Interface in Java

It is one of the predefined interface present in `java.util.*` package. The `ListIterator` interface object is always used for retrieving the data from any collection framework either forward direction or backward direction or in both direction. Like `Iterator` interface object, `ListIterator` interface object is pointing just before the first element of any collection framework variable.

Methods of ListIterator Interface

- **public boolean hasNext():** Returns true if the `ListIterator` has more elements when traversing the list in the forward direction.
- **public object next():** Return the next elements in the list.
- **public boolean hasprevious():** Return true if the `ListIterator` has more elements when traversing in list in reverse direction.
- **public object previous():** Return the previous elements in the list.
- **public object previousIndex():** Return the index of the element that could be returned by subsequent call to `previous` method.
- **public object add(element):** Insert the specified element into the list.
- **public object set(element):** Replace the element returned by `next` or `previous` elements with the new element.

- **public object remove():** Remove from the list the last elements that was returned by next or previous methods.

Difference between Iterator and ListIterator

Iterator

ListIterator

1 Iterator is one of the super Interface for ListIterator
 1 ListIterator is one of sub-Interface of Iterator

2 This Interface is allow to retrieve the element only in forward direction
 This Interface is allow to retrieve the element in both direction either in forward and backward direction.

Example of ListIterator

```
import java.util.*;

class ListIteratorDemo
{
    public static void main(String args[])
    {
        LinkedList<Integer> ll=new
        LinkedList<Integer>(); // creating
        arraylist
        ll.add(10);
        ll.add(20);
        ll.add(30);

        ListIterator li=al.listIterator(); //
        getting Iterator from arraylist to traverse
        elements
        System.out.println("Forward Direction");
        while(li.hasNext())
```

```
{  
System.out.println(li.next());  
}  
System.out.println("Reverse Direction");  
while (li.hasPrevious())  
{  
System.out.println(li.previous());  
}  
}  
}
```

Output

```
Forward Direction  
10  
20  
30  
Reverse Direction  
30  
20  
10
```

Enumeration Interface in Java

It is one of the predefined interface and whose object is always used for retrieving the data from any legacy collection framework variable (like vector, stack, HasTable etc.) only in forward direction but not in backward direction. Like Iterator interface object, Enumeration Interface object is pointing just before the first element of any legacy collection framework variable.

The functionality of Enumeration is more or less similar to Iterator Interface but Enumeration Interface object belongs to synchronized and Iterator Interface object belong to non-synchronized.

Points to Remember

- Enumeration does not support addition, removing and replacing of elements.
- Enumeration is Synchronized.
- Using Enumeration elements of legacy collection can be access only in forward direction.
- Every legacy class contains following methods to work with enumeration and it returns enumeration object.

Methods of Enumeration Interface

- **public boolean hasMoreElements():** Return true if Enumeration contains more elements otherwise returns false.
- **public object nextElement():** Returns the next elements of Enumeration.

Create an object of Enumeration

Syntax

```
Vector v=new Vector();  
Enumeration e=v.elements();
```

Example of Enumeration

```
import java.util.ArrayList;  
  
class EnumerationDemo  
{  
public static void main(String args[])  
{  
  
Vector<Integer> al=new vector<Integer>();  
// creating Vector  
v.add(10);
```

```
v.add(20);  
v.add(30);  
  
Enumeration e=v.elements();  
System.out.println("Forword Direction");  
  
while(itr.e.hasMoreElements())  
{  
System.out.println(e.e.nextElement());  
}  
}
```

Output

```
Forword Direction  
10  
20  
30
```

List Interface in Java

List Interface is used to store multiple objects with duplicates that means it allows to store duplicate elements. List Interface are implemented in following collection classes.

- Stack
- Vector
- ArrayList
- LinkedList

Create object for List Implementer classes

Syntax

```
List_Implemented_class<generic_class>  
obj=new  
List_Implemented_class<generic_class>();
```

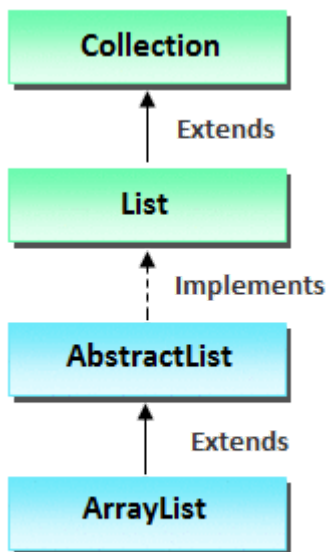
Example to create object of Stack

```
Stack<String> obj=new Stack<String>();
```

ArrayList in Java

ArrayList is a replacement of vector class, It is a new class used to store multiple objects.

In ArrayList the data is organizing in the form of cells. Cell values are storing in heap memory and cell address are storing in associative memory.



Tutorial4us.com

Points to Remember

- ArrayList class is not Synchronized.
- ArrayList class elements can be access randomly.
- In the ArrayList value will be stored in the same order as inserted.

- ArrayList class uses a dynamic array for storing the elements. It extends AbstractList class and implements List interface.
- ArrayList class can contain duplicate elements.
- ArrayList allows random access because array works at the index basis.

Note: ArrayList class contains same methods like vector.

Creating ArrayList is nothing but creating an object of ArrayList class.

Syntax

```
ArrayList al=new ArrayList();
```

ArrayList Constructor

ArrayList(): This constructor is used for creating an object of ArrayList class.

Syntax

```
ArrayList al=new ArrayList();
```

Advantages of ArrayList

- ArrayList based applications takes less memory space.
- Retrieving the data from ArrayList will take less time.
- Performance of ArrayList based applications is more.

Difference Between Vector and ArrayList

	Vector	ArrayList
1	Vector is legacy Collection Framework (old class).	ArrayList is new Collection Framework.

2 Vector is Synchronized by
default.

ArrayList is not Synchronized.

3 For retrieving elements from
Vector class can be use foreach
loop, iterator, listiterator and
enumeration.

For retrieving elements from
ArrayList class can be use
foreach loop, iterator and
listiterator.

Example of ArrayList

```
import java.util.ArrayList;

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

        ArrayList<Integer> al=new
        ArrayList<Integer>();    // creating
        arraylist
        al.add(10);
        al.add(20);
        al.add(30);

        Iterator itr=al.iterator();    // getting
        Iterator from arraylist to traverse
        elements
        while(itr.hasNext())
        {
            System.out.println(itr.next());
        }
    }
}
```

Output

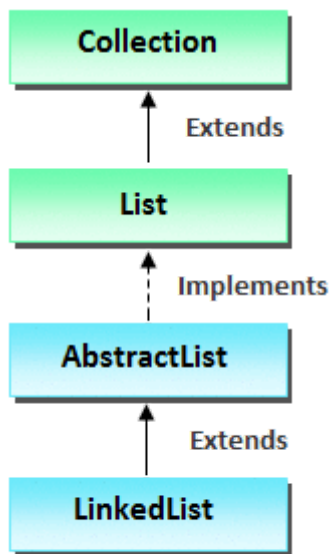
10
20
30

LinkedList class in Java

A LinkedList is a Collection of nodes. LinkedList class uses doubly linked list to store the elements. It extends the AbstractList class and implements List and Deque interfaces.



Hierarchy of LinkedList



Tutorial4us.com

Important points LinkedList

- A LinkedList is a Collection of nodes.
- LinkedList class is not Synchronized.
- LinkedList class can contain duplicate elements.
- LinkedList organize the data in the form of nodes.

Constructor of LinkedList

- **LinkedList():** is used for creating an object of LinkedList without specifying any number of nodes.
- **LinkedList(int):** is used for creating an object of ll by specifying the initial number of nodes to be created.

Syntax

```
LinkedList ll=new LinkedList();
```

Here ll represent object name and it can be treated as Collection framework variable and it allows us to store multiple values either of same type or different type or both the types.

Methods of LinkedList

- **public void addFirst(Object):** are used for adding the elements at first position of LinkedList.
- **public void addLast(Object):** are used for adding the elements at last position of LinkedList.
- **public Object getFirst():** are used for get first position elements of LinkedList.
- **public Object getLast():** are used for get last position elements of LinkedList.
- **public Object removedFirst():** are used for remove first position elements of LinkedList.
- **public Object removedLast():** are used for remove last position elements of LinkedList.

Advantages of LinkedList

- LinkedList class object allows us to organize the data and it may belong to both homogeneous or heterogeneous elements.
- LinkedList class object contains dynamic size in nature.

- LinkedList class object allows us to insert the data either in the beginning or in the ending or in the middle (dynamic insertion are allowed).

Limitations of LinkedList

- LinkedList class object takes more memory space (explicitly) memory space is created for Data part and Address part in heap memory.
- LinkedList class object takes more retrieval time(after processing data of the current node,to process the data of next node,internally JVM will retrieve address of next node from address part of current node which is one of the consuming process.).
- Less performance:

To avoid the problems of LinkedList we use the concept of ArrayList. Because ArrayList class developed by Sun developers by making use of associative memory.

Note: If we store anything in associative memory which will take negligible amount of space and if we retrieve anything from associative memory than it will take negligible amount of time.

Example of LinkedList

```
import java.util.*;
class LinkedListDemo
{
    public static void main(String[] args)
    {
        LinkedList<Integer> ll=new
LinkedList<Integer> ();

        // Add content
```

```

        ll.add(20);
        ll.add(30);
        ll.add(40);
        ll.add(50);
        Iterator itr=ll.iterator();
        while(itr.hasNext())
        {
            System.out.println(itr.next());
        }
    }
}

```

Output

```

20
30
40
50

```

Difference Between ArrayList and LinkedList

ArrayList and LinkedList both are implements List interface and maintains insertion order. ArrayList and LinkedList both are non synchronized classes and these are new collection framework classes.

Note: All new Collection Framework are non-Synchronized and all Legacy Collection Framework are Synchronized.

Difference between ArrayList and LinkedList

ArrayList	LinkedList
1 ArrayList internally uses dynamic array to store the elements.	LinkedList internally uses doubly linked list to store the elements.

<p>Manipulation with ArrayList is slow because it internally uses array. If any element is removed from the array, all the bits are shifted in memory.</p> <p>ArrayList class can act as a list only because it implements List only.</p> <p>ArrayList is better for storing and accessing data.</p>	<p>Manipulation with LinkedList is faster than ArrayList because it uses doubly linked list so no bit shifting is required in memory.</p> <p>LinkedList class can act as a list and queue both because it implements List and Deque interfaces.</p> <p>LinkedList is better for storing and accessing data.</p>
--	---

Example of ArrayList and LinkedList

```
import java.util.*;
class DemoArrayLinked
{
    public static void main(String args[])
    {
        List<String> al=new ArrayList<String>();
        //creating arraylist
        al.add("Mark");//adding object in
        arraylist
        al.add("Deo");
        al.add("Pitter");
        al.add("Porter");

        List<String> ll=new LinkedList<String>();
        //creating linkedlist
```



```

    ll.add("Smith");    //adding object in
linkedlist
    ll.add("Kater");
    ll.add("Parker");
    ll.add("Holly");

    System.out.println("Arraylist: "+al);

System.out.println(".....
..");
    System.out.println("Linkedlist: "+ll);
}
}

```

Output

```

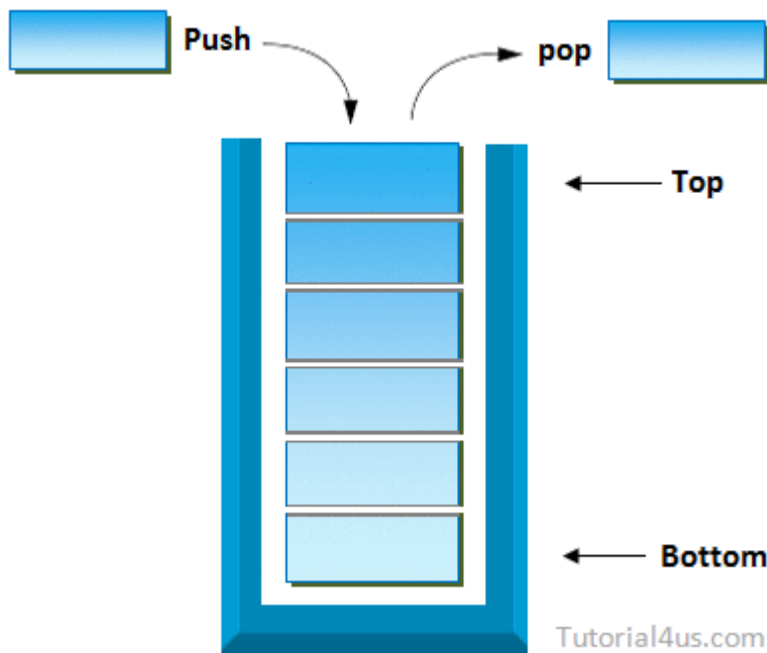
Arraylist: [Mark, Deo, Pitter, Porter]
.....
Linkedlist: [Smith, Kater, Parker, Holly]

```

Stack in Java

Stack is one of the sub-class of Vector class so that all the methods of Vector are inherited into Stack.

The concept of Stack of Data Structure is implemented in java and develop a pre-defined class called Stack.



Stack Important Points

- Stack class allow to store Heterogeneous elements.
- Stack work on Last in First out (LIFO) manner.
- Stack allow to store duplicate values.
- Stack class is Synchronized.
- Initial 10 memory location is create whenever object of stack is created and it is re-sizable.
- Stack also organizes the data in the form of cells like Vector.
- Stack is one of the sub-class of Vector.

Creating a Stack is nothing but creating an object of Stsck Class.

Syntax

```
Stack s=new Stack();
```

Constructors of Stack

- **Stack():** is used for creating an object of Stack.

Syntax

```
Stack s=new Stack();
```

Methods of Stack

- **public boolean empty():** is used for returns true provided Stack is empty.It returns false in case of Stack is non-empty.
- **public void push (Object):** is used for inserting the elements into the Stack.
- **public Object pop():** is used for removing Top Most elements from the Stack.
- **public Object peek():** is used for retrieving Top Most element from the Stack.
- **public int search(Object):** is used for searching an element in the Stack.If the element is found then it returns Stack relative position of that element otherwise it returns -1, -1 indicates search is unsuccessful and element is not found.

Example of Stack add Elements to Stack

```
import java.util.*;
class StackDemo
{
public static void main(String args[])
{
Stack s=new Stack();

//add the data to s
s.push(10);
s.push(20);
s.push(30);
s.push(40);
System.out.println("Stack data: "+s); //
[10, 20, 30, 40]
}
}
```

Output

Stack data: [10,20,30,40]

Example of Stack to add and remove Elements from Stack

```
import java.util.*;
class StackDemo
{
public static void main(String args[])
{
Stack s=new Stack();

// Add the data to s
s.push(10);
s.push(20);
s.push(30);
s.push(40);
System.out.println("Stack elements: "+s);
//[10,20,30,40]

// Remove the top most element
System.out.println("Delete element:
"+s.pop()); //40
System.out.println("Stack elements after
pop: "+s); // [10,20,30]
}
}
```

Output

Stack elements: [10,20,30,40]

Delete element: 40

Stack elements after pop: [10,20,30]

Complete Example of Stack

```
import java.util.*;
```

```

class StackDemo
{
public static void main(String args[])
{
Stack s=new Stack();
System.out.println("content of s="+s);
//[ ]
System.out.println("size of s="+s.size());
//10
System.out.println("Is empty?="+s.empty());
//true

//add the data to s
s.push(10);
s.push(20);
s.push(30);
s.push(40);
System.out.println("content of s="+s);
//[10,20,30,40]
System.out.println("size of s="+s.size());
//4
System.out.println("Is s empty
?=s.empty()"); //false
//remove the top most element
System.out.println("delete
element="+s.pop()); //40
System.out.println("content of s after
pop="+s); // [10,20,30]
//extract the top most element
System.out.println("top most
element="+s.peek()); //30
System.out.println("content of s after
peek="+s); // [10 20 30]
//Search the element 10 and 100

```

```
int srp=s.search(10);  
System.out.println("stack relative pos.of  
10 is="+srp);      //3  
int srp1=s.search(100);  
System.out.println("stack relative pos.of  
100 is="+srp1);    //-1  
}  
}
```

Output

Vector in Java

It is one of the Legacy Collection framework class. Vector class object organizes the data in the form of cells. Creating a Vector is nothing but creating an object of Vector class.

Syntax

```
Vector v=new Vector();
```

Vector Important Points

- The default capacity of the Vector v=10 cells
- Vector are Synchronized.
- Vector uses Enumeration interface to traverse the elements. It can be use Iterator also.
- Vector is one of the Legacy Collection framework class.
- Vector class object organizes the data in the form of cells.
- The default size of the Vector=0 (size is nothing but number of values available in the cells)
- In Vector cell values are storing in Heap Memory and cell address are storing in associative memory.

Constructor of vector

- **Vector():** Vector() is used for creating an object of Vector without specifying any number of cells to be created.
- **Vector(int):** Vector(int) is used for creating an object for Vector class by specifying number of cells to be created.

Methods of Vector

- **public int capacity():** are used for find the capacity of Vector.
- **public int size():** are used for find size of Vector class object.
- **public void addElement(Object):** are used for adding the elements to the Vector one-by-one.
- **public void addElementAt(int.Object):** are used for adding the elements to the Vector at specific existing position.
- **public Object removeElementAt(int):** are used for removing the elements of Vector on the basics of the position.
- **public void removeElement(Object):** are used for removing the elements of Vector on the basics of content.
- **public Enumeration elements():** are used for extracting all the elements of Vector class object

Syntax

```
Vector v=new Vector();  
System.out.println("capacity of  
v="+v.capacity()); // 10  
System.out.println("size of v="+v.size());  
// 0  
v.addElement(10);  
v.addElement(10.75f);  
System.out.println(v);           // [10,10.75]
```

```

v.addElementAt(1, 'A');
System.out.println(v);      // [10,A,10.75]
Object obj=v.removeElementAt(1);
System.out.println(obj);
// A
v.removeElement('A');
System.out.println(v);      // [10,10.75]

```

Example of Vector

```

import java.util.*;

class VectorDemo
{
    public static void main(String [] args)
    {
        Vector v=new Vector();

        // Add data to v
        v.addElement("Deo");
        v.addElement("Smith");
        v.addElement("Karter");
        v.addElement("Porter");

        // Extract the data
        Enumeration e=v.elements();
        while(e.hasMoreElements()){
            System.out.println(e.nextElement());
        }
    }
}

```

Output

```

Deo
Smith

```


Set Interface in Java

Set is an Interface in java API which extends Collection Interface. It is used to store multiple objects with unique that means it do not allows to store duplicate elements. Set Interface are implemented in following collection classes.

- HashSet
- TreeSet
- LinkedHashSet

Note: Set is based on Set in general mathematics which also do not store duplicate elements so here Set Interface do not allows to store duplicate elements.

Create object for Set Implementer classes

Syntax

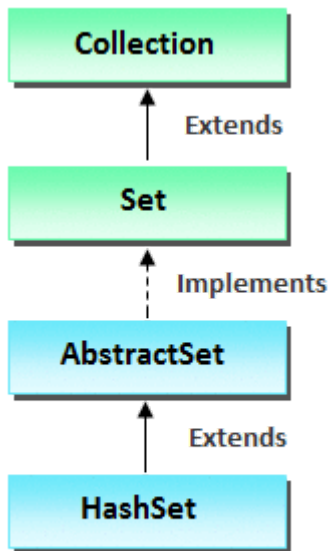
```
Set_Implemented_class<generic_class>  
obj=new  
Set_Implemented_class<generic_class>();
```

Example to create object of HashSet

```
HashSet<String> obj=new HashSet<String>();
```

HashSet in Collection Framework

HashSet is **Implementer** class of Set Interface. HashSet supports hashing mechanism to store the value that means all the elements are stored in un-sorted random order (the elements will not be in same order as inserted).



Tutorial4us.com

Points to Remember

- HashSet are not allows to store duplicate elements.
- HashSet allows to store heterogeneous elements.
- For retrieving elements from HashSet you can use foreach loop and iterator interface to retrieve the elements.
- HashSet is not Synchronized means multiple threads can work Sanctimoniously.
- HashSet allows to store null value.

Example of HashSet

```
import java.util.*;

class HashSetDemo
{
    public static void main(String args[])
    {
        HashSet<String> hs=new
HashSet<String> ();
        hs.add("Java");
        hs.add("C-lang");
        hs.add("C++");
    }
}
```

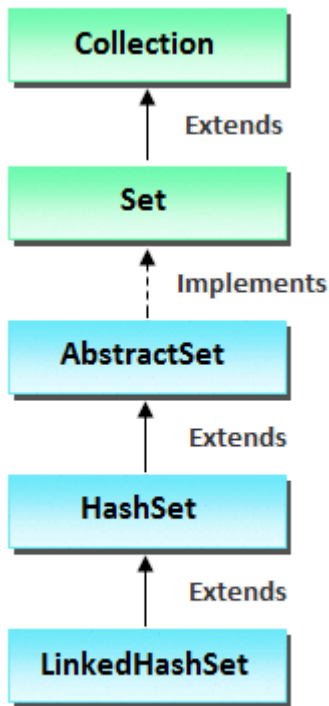
```
        hs.add("Java");  
        System.out.println(hs);  
        Iterator i=hs.iterator();  
        System.out.println("Forward Direction");  
  
        while(l.hasNext())  
        {  
            System.out.println(i.next());  
        }  
    }  
}
```

Output

Java
C-lang
C++

LinkedHashSet in Java

LinkedHashSet is **Implementer** class of Set Interface, Which supports hashing mechanism to store the value that means all the elements are stored in the same order as inserted (In unsorted format).



Tutorial4us.com

Points to Remember

- LinkedHashSet does not allow duplicate elements.
- LinkedHashSet allows to store heterogeneous elements
- LinkedHashSet is not Synchronized.
- For retrieving elements from LinkedHashSet you can use foreach loop and iterator interface.
- LinkedHashSet allows null values.

Example of LinkedHashSet

```
import java.util.*;

class LHashDemo
{
    public static void main(String args[])
    {
        LinkedHashSet<String> lhs=new
        LinkedHashSet<String>();
        lhs.add("Java");
        lhs.add("C-lang");
    }
}
```

```

        lhs.add("C++");
        lhs.add("Java");
        System.out.println(lhs);
        Iterator i=lhs.iterator();
        System.out.println("Forward Direction");

        while(l.hasNext())
        {
            System.out.println(i.next());
        }
    }
}

```

Output

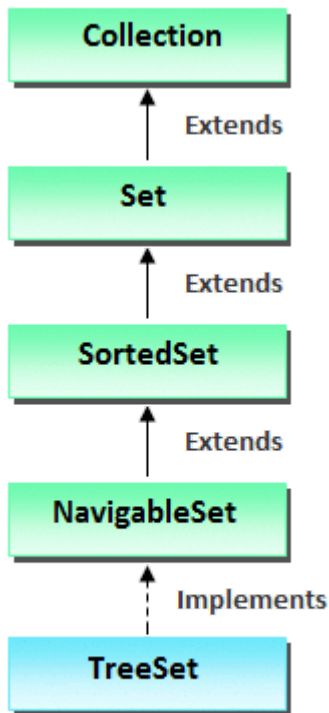
```

Java
C-lang
C++

```

Queue Interface in Java

Queue Interface defines the queue data structure, which stores elements in the order in which they are inserted. New additions go to the end of the line, and elements are removed from the front. It creates a first-in first-out system.



Tutorial4us.com

Methods of Queue Interface

- public boolean add(object)
- public boolean offer(object)
- public remove()
- public poll()
- public element()
- public peek()

Queue Interface can be implement using following classes;

- PriorityQueue
- ArrayDeque

PriorityQueue

Example of PriorityQueue

```
import java.util.*;  
class PriorityQueueDemo  
{
```

```

public static void main(String args[])
{

    PriorityQueue<String> queue=new
    PriorityQueue<String> ();

    // Add Elements
    queue.add("Deo");
    queue.add("Smith");
    queue.add("Piter");
    queue.add("Poter");

    System.out.println("Iterating the queue
    Elements:");
    Iterator itr=queue.iterator();
    while(itr.hasNext()){
        System.out.println(itr.next());
    }
}
}

```

Output

```

Iterating the queue Elements:
Deo
Smith
Piter
Porte

```

PriorityQueue

Example of PriorityQueue

```

import java.util.*;
class PriorityQueueDemo
{

```

```

public static void main(String args[])
{

    PriorityQueue<String> queue=new
    PriorityQueue<String> ();

    // Add Elements
    queue.add("Deo");
    queue.add("Smith");
    queue.add("Piter");
    queue.add("Poter");

    // Removing Elements
    queue.remove();
    queue.poll();

    System.out.println("Removing the queue
    Elements:");
    Iterator itr=queue.iterator();
    while(itr.hasNext()){
        System.out.println(itr.next());
    }
}
}

```

Output

```

Removing the queue Elements:
Piter
Porte

```

Map Interface in Java

Maps are defined by the `java.util.Map` interface in Java. Map is an Interface which store key, value in which both key and value are objects Key must be unique and value can be duplicate. Here

value are elements which store in Map. Map Interface is implemented in the following four classes.

- HashMap
- HashTable
- LinkedHashMap
- TreeMap

Create object for map Implementer classes

Syntax

```
Map_Implemented_class<generic_class1,  
generic_class2> obj=new  
Map_Implemented_class<generic_class1,  
generic_class2>();
```

Example to create object of HashMap

Example

```
HashMap<Integer, Float> obj=new  
HashMap<Integer, Float>();
```

Methods of Map Interface

public int size()

This method is used for finding the number of entries in any 2-D Collection framework variable.

public boolean isEmpty()

This method returns true provided 2-D Collection framework is empty (size=0).Returns false in the case of non-empty (size>0).

public void put(Object, Object)

This method is used for adding and modifying the entries of 2-D Collection framework variable. If the (key, value) pair is not existing than such (k,v)pair will be added to the 2-D Collection framework variable and it will be considered as Inserted Entry. In the (k,v) pair ,if the value of key is already present in 2-D collection framework variable than the existing value of value is replaced by new value of value.

Example

```
m.put(10, 7.5f);          //Inserted entery
m.put(20, 1.8f);
System.out.println(m);    // (10=7.5)
                           (20=1.8)
m.put(10, 9.5);           //Modified entery
System.out.println(m);    // (10=9.5)
                           (20=1.8)
```

public Object get(Object obj)

This method is used for obtaining the value of value by passing value of key

Example

```
Object vobj1=m.get(10);
Object vobj2=m.get(100);
```

public Set entrySet()

This method is used for extracting or retrieving all the entries of any 2-D Collection framework variable and hold them in the object of java.util.Set interface.

Example

```
System.out.println(m);
// { (10=7.5) (20=1.9) (30=1.5) }
```

```

Set s= m.entrySet();
//s={ (10=7.5) (20=1.9) (30=1.5) }
Iterator itr=s.iterator();
while(itr.hasNext())
{
Object mobj=itr.next();    //4
Map.Entry me=(Map.Entry)mobj;
Object kobj=me.getKey();
Object vobj=me.getValue();
Integer io=(Integer)kobj;
Float fo=(Float)vobj;
int acno=io.intValue();
float bal=fo.floatValue();
System.out.println(bal+"is the bal
of"+acno);
} //while

```

public set keySet()

This method is used for obtaining the set of keys from any 2-D Collection framework variable and pass those set of keys to the get() for obtaining the values of value.

Example

```

System.out.println(m);
//{ (10=7.5) (20=1.9) (30=1.5) }
Set s= m.keySet();
//s={ (10=7.5) (20=1.9) (30=1.5) }
Iterator itr=s.iterator();
while(itr.hasNext())
{
Object kobj=itr.next();
Object vobj=me.get(kobj);
Integer io=(Integer)kobj;

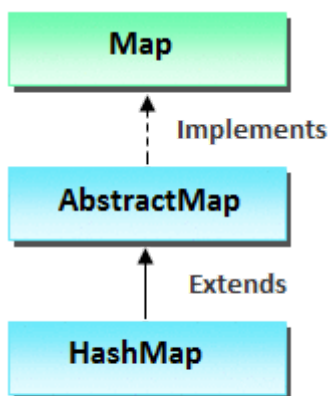
```

```
Float fo=(Float)vobj;  
int acno=io.intValue();  
float bal=fo.floatValue();  
System.out.println(bal+"is the bal  
of"+acno);  
} //while
```

Note: These all above methods are applicable for all Implementer classes of Map Interface.

HashMap in Java

HashMap are the **Implementer** class of Map Interface, which store the values based on the key.



Tutorial4us.com

Points to Remember

- HashMap are not allows to store duplicate elements.
- HashMap are new collection framework class.
- HashMap may have one null key and multiple null values.
- For retrieving elements from HashMap you can use foreach loop, Iterator Interface and ListIterator Interface to retrieve the elements.
- HashMap is not Synchronized means multiple threads can work Simultaneously.

Example of HashMap

```
import java.util.*;

class HashMapDemo
{
    public static void main(String args[])
    {
        HashMap<Integer,String> hm=new
HashMap<Integer,String>();
        hm.put(1,"Deo");
        hm.put(2,"zen");
        hm.put(3,"porter");
        hm.put(4,"piter");

        for(Map.Entry m:hm.entrySet())
        {
            System.out.println(m.getKey()+"
"+m.getValue());
        }
    }
}
```

Output

```
1 Deo
2 zen
3 porter
4 piter
```

Example of HashMap

```
import java.util.*;

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

```

{
    HashMap<Integer, Float> hm=new
HashMap<Integer, Float>();
    hm.add(10, 10.5);
    hm.add(20,20.5);
    hm.add(30,30.5);
    hm.add(40,40.5);
    hm.add(Null,50.5);
    hm.add(Null,60.5);    // only one null
allow
    System.out.println(hm);
    System.out.println(hm.values());
    System.out.println(hm.keySet());
    System.out.println(hm.get(20));
}
}

```

Output

```

[null=60.5, 20=20.5, 10=10.5, 40=40.5,
30=30.5]
[60.5, 20.5, 10.5, 40.5, 30.5]
[Null, 20, 10, 30, 40]
[20.5]

```

HashTable in Java

HashTable is **Implementer** class of Map interface and extends Dictionary class. HashTable does not allows null key and null values, these elements will be stored in a random order.

Points to Remember

- HashTable is a legacy class, which will uses hashing technique (the elements will be stored in unsorted or un-order format).

- HashTable stored the elements in key values formate.
- HashTable does not allows null keys and null values.
- HashTable is Synchronized.
- HashTable allows heterogeneous elements.
- For retrieve elements from HashTable you can use foreach loop, Iterator Interface and Enumeration.

Note: HashTable class also contains same methods like HashMap.

Difference between HashMap and HashTable

	HashMap	HashTable
1	HashMap is a new class in java API.	HashTable is a legacy class in java API.
2	HashMap is not Synchronized.	HashTable is Synchronized.
3	HashMap allows maximum one null key and multiple null value.	HashTable does not allows any null key and null values.

Limitations of HashTable

- Hashtable class object is enable to read the data from property file / resource bundle file.
- Hashtable class object is unable to develop flexible application.

Example of HashTable

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

```

    Hashtable<Integer,String> ht=new
    Hashtable<Integer,String>();
    ht.put(1,"Deo");
    ht.put(2,"zen");
    ht.put(3,"porter");
    ht.put(4,"piter");

    System.out.println(ht);
}
}

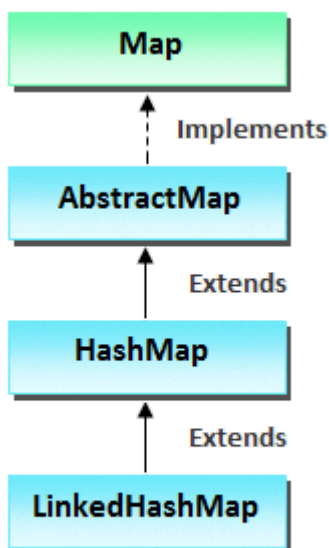
```

Output

[1=Deo, 2=zen, 3=porter ,4=piter]

LinkedHashMap in Java

LinkedHashMap class are **Implementer** class of Map Interface which contains values based on the key. It implements the Map interface and extends HashMap class.



Tutorial4us.com

Points to Remember

- LinkedHashMap contains only unique elements.

- LinkedHashMap have one null key or can have multiple null values.
- It is same as HashMap instead maintains insertion order.

Example of LinkedHashMap

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

        LinkedHashMap<Integer,String> tm=new
        LinkedHashMap<Integer,String>();
        lhm.put(1,"Deo");
        lhm.put(2,"zen");
        lhm.put(3,"porter");
        lhm.put(4,"piter");

        for(Map.Entry m:lhm.entrySet())
        {
            System.out.println(m.getKey()+"
"+m.getValue());
        }
    }
}
```

Output

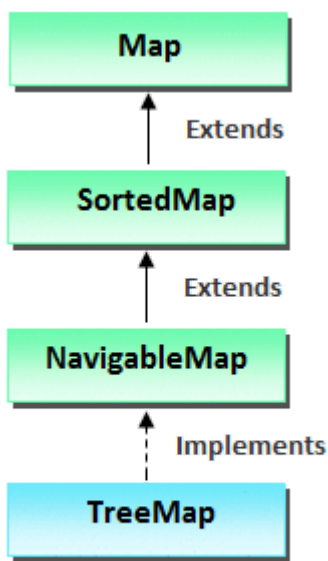
```
1 Deo
2 zen
3 porter
4 piter
```

TreeMap in Java

A TreeMap contains values based on the key. It implements the NavigableMap interface and extends AbstractMap class.

Points to Remember

- 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.



Tutorial4us.com

Example of TreeSet

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

```
        TreeMap<Integer,String> tm=new
TreeMap<Integer,String>();
        tm.put(1,"Deo");
        tm.put(2,"zen");
        tm.put(3,"porter");
```

```
tm.put(4, "piter");

for (Map.Entry m:tm.entrySet())
{
    System.out.println(m.getKey()+"
"+m.getValue());
}
}
```

Output

```
1, Deo
2, zen
3, porter
4, piter
```

Properties Files in Collection Framework

A property file is one type of the file which organizing the data in the form of (key, value) pair. A property file is a text file which is to be created in any editors like-notepad, Editpuls and etc. Property file should be saved on same file name with on extension .prop or .rbf.

Properties Files Important points

- A property file is a text file which is to be created in any editors like-notepad, Editpuls and etc.
- Property file should be saved on same file name with on extension .prop or .rbf.
- Properties class object organizes the data in the form of (key, value) pair and it displays in the same order in whichever order it is added.
- Property file always resides in secondary memory.

java.util.properties

It is one of the pre-defined sub-class of Hashtable class, so that all the methods of Hashtable are inherited into properties class. Creating Properties is nothing but creating an object of properties class.

Syntax

```
Properties p=new Properties();  
// Here p resides in heap memory.
```

Advantages of Properties class over Hashtable class

- Properties class object is able to read the data from properties/resource bundle file.
- Properties class always makes us to develop flexible application.

Constructor and Method of Properties class

Constructor perporties()

methods `public void load(FileInputStream);`
 `public String get Property(String);`

Constructor of properties class is used for creating an object of properties class.

Syntax

```
Properties p=new Properties();
```

public void load(FileInputStream); is used for loading the content of property file into properties class object by opening the properties file in read mode with the help of FileInputStream class.

Example

```
FileInputStream fis=new  
FileInputStream("student properties");  
p.load(fis);
```

public String get Property(String); is used for obtaining the value of value by passing the key.

Student.prop

```
Sname = porter  
Sno = 10  
Marks = 98.99
```

Example

```
String sno=p.getProperty("stno");  
String sname=p.getProperty("sname");  
String marks=p.getProperty("marks");
```

Student.prop

```
Sname = porter  
Sno = 10  
Marks = 98.99
```

Example of properties file

```
import java.util.*;  
import java.io.*;  
class Prop  
{  
public static void main(String []args)  
{  
try  
{  
Scanner s=new Scanner(System.in);
```

```

System.out.println("Enter the property file
name");
String pfile=s.nextLine();
FileInputStream fis=new FileInputStream();
Properties p=new Properties();
p.load(fis);
String sno=p.getProperty();
String sname=p.getProperty();
String marks=p.getProperty();
System.out.println("student number="+sno);
System.out.println("student name="+sname);
System.out.println("student marks="+marks);
fis.close();
}
catch(FileNotFoundException fe)
{
System.out.println("properties file does
not exists");
}
catch(Exception e)
{
System.out.println(e);
}
}
}

```