

Subject: Object Oriented Programming in Java

Chapter: Chapter 1 (Introduction to Java)

What is java?

Java is a high-level programming language originally developed by Sun Microsystems and released in 1995.

Java is a write-once, run-anywhere programming language developed by Sun Microsystems. It is similar to C and C++ but a lot easier.

History of Java

Java was originally developed by James Gosling with his colleagues at Sun Microsystems during the early 1990s. Initially, it was called as project 'Oak' which had implementation similar to C and C++. The name Java was later selected after enough brainstorming and is based on the name of an espresso bean. Java 1.0, the first version was released in 1995 with the tagline of 'write once, run anywhere'. Later, Sun Microsystems was acquired by Oracle. From there, there has been no looking back. The latest version of Java is Java 12 released in March 2019.

Features of Java

Object Oriented – In Java, everything is an Object. Java can be easily extended since it is based on the Object model.

Platform Independent – Unlike many other programming languages including C and C++, when Java is compiled, it is not compiled into platform specific machine, rather into platform independent byte code. This byte code is distributed over the web and interpreted by the Virtual Machine (JVM) on whichever platform it is being run on.

Simple – Java is designed to be easy to learn. If you understand the basic concept of OOP Java, it would be easy to master.

Secure – With Java's secure feature it enables to develop virus-free, tamper-free systems. Authentication techniques are based on public-key encryption.

Architecture-neutral – Java compiler generates an architecture-neutral object file format, which makes the compiled code executable on many processors, with the presence of Java runtime system.

Portable – Being architecture-neutral and having no implementation dependent aspects of the specification makes Java portable. Compiler in Java is written in ANSI C with a clean portability boundary, which is a POSIX subset.

Robust – Java makes an effort to eliminate error prone situations by emphasizing mainly on compile time error checking and runtime checking.

Multithreaded – With Java's multithreaded feature it is possible to write programs that can perform many tasks simultaneously. This design feature allows the developers to construct interactive applications that can run smoothly.

Interpreted – Java byte code is translated on the fly to native machine instructions and is not stored anywhere. The development process is more rapid and analytical since the linking is an incremental and light-weight process.

High Performance – With the use of Just-In-Time compilers, Java enables high performance.

Distributed – Java is designed for the distributed environment of the internet.

Dynamic – Java is considered to be more dynamic than C or C++ since it is designed to adapt to an evolving environment. Java programs can carry extensive amount of run-time information that can be used to verify and resolve accesses to objects on run-time.

Components of Java

JDK	JVM	JRE
JRE + development tools like interpreter(class loader), compiler (javac), jar files (package and archive) and javadocs.	The abstract machine where the java bytecodes are executed. Consists of <i>specification</i> document that describes JVM implementation, the actual <i>implementation</i> program and the <i>instance</i> of JVM (run-time) where you can run your main program.	Physical implementation (Runtime instance) of JVM. Contains the library packages and support files that JVM uses for running a program.

Application and Applet in java

An **applet** is a Java™ program designed to be included in an HTML Web document. You can write your Java applet and include it in an HTML page, much in the same way an image is included. When you use a Java-enabled browser to view an HTML page that contains an applet, the applet's code is transferred to your system and is run by the browser's Java virtual machine.

Applications are stand-alone programs that do not require the use of a browser. Java applications run by starting the Java interpreter from the command line and by specifying the file that contains the compiled application. Applications usually reside on the system on which they are deployed. Applications access resources on the system, and are restricted by the Java security model.

Byte Code

Bytecode, also termed **portable code** or **p-code**, is a form of instruction set designed for efficient execution by a software interpreter. Unlike human-readable source code, bytecodes are compact numeric codes, constants, and references (normally numeric addresses) that encode the result of compiler parsing and performing semantic analysis of things like type, scope, and nesting depths of program objects.\

Procedural Programming Vs. Object Oriented Programming:

PROCEDURAL ORIENTED PROGRAMMING	OBJECT ORIENTED PROGRAMMING
In procedural programming, program is divided into small parts called <i>functions</i> .	In object oriented programming, program is divided into small parts called <i>objects</i> .
Procedural programming follows <i>top down approach</i> .	Object oriented programming follows <i>bottom up approach</i> .
There is no access specifier in procedural programming.	Object oriented programming have access specifiers like private, public, protected etc.
Adding new data and function is not easy.	Adding new data and function is easy.
Procedural programming does not have any proper way for hiding data so it is <i>less secure</i> .	Object oriented programming provides data hiding so it is <i>more secure</i> .
In procedural programming, overloading is not possible.	Overloading is possible in object oriented programming.

In procedural programming, function is more important than data.	In object oriented programming, data is more important than function.
Procedural programming is based on <i>unreal world</i> .	Object oriented programming is based on <i>real world</i> .
Examples: C, FORTRAN, Pascal, Basic etc.	Examples: C++, Java, Python, C# etc.

Local Environment Setup

If you are still willing to set up your environment for Java programming language, then this section guides you on how to download and set up Java on your machine. Following are the steps to set up the environment.

Java SE is freely available from the link [Download Java](#). You can download and install a version based on your operating system. Once you installed Java on your machine, you will need to set environment variables to point to correct installation directories –

Setting Up the Path for Windows

Assuming you have installed Java in *c:\Program Files\java\jdk* directory –

- Right-click on 'My Computer' and select 'Properties'.
- Click the 'Environment variables' button under the 'Advanced' tab.
- Now, alter the 'Path' variable so that it also contains the path to the Java executable. Example, if the path is currently set to 'C:\WINDOWS\SYSTEM32', then change your path to read 'C:\WINDOWS\SYSTEM32;c:\Program Files\java\jdk\bin'.

Setting Up the Path for Linux, UNIX, Solaris, FreeBSD

Environment variable PATH should be set to point to where the Java binaries have been installed. Refer to your shell documentation, if you have trouble doing this.

Example, if you use *bash* as your shell, then you would add the following line to the end of your '.bashrc':
`export PATH = /path/to/java:$PATH`

Popular Java Editors

To write your Java programs, you will need a text editor. There are even more sophisticated IDEs available in the market. But for now, you can consider one of the following –

- **Notepad** – On Windows machine, you can use any simple text editor like Notepad (Recommended for this tutorial), TextPad.
- **Netbeans** – A Java IDE that is open-source and free which can be downloaded from <https://www.netbeans.org/index.html>.
- **Eclipse** – A Java IDE developed by the eclipse open-source community and can be downloaded from <https://www.eclipse.org/>.

What is Next?

Next chapter will teach you how to write and run your first Java program and some of the important basic syntaxes in Java needed for developing applications.

Programming code

A computer is merely a machine that can process a set of simple instructions very quickly. The set of instructions it processes is known as a “program”, and the instructions are known as “code”.

People who write computer programs are known as “programmers” or “coders”. Their programs have enabled computers to become useful in almost every area of modern life:

- **In the hand** – computers are found in cellphone devices for tasks such as communication via voice, text, and social media
- **In the home** – computers are found in household devices such as TV sets, gaming consoles, and washing machines
- **In the office** – computers are found in desktop devices for tasks such as word processing, payroll, and graphic design
- **In the store** – computers are found in retail devices such as automatic teller machines (ATMs) and bar code scanners
- **In the car** – computers are found in control devices for tasks such as engine management, anti-lock braking and security
- **In the sky** – computers are found in airplanes for piloting and in air traffic control centers for safe navigation

It is important that each computer program provides clear step-by-step instructions that the computer can execute without errors.

The coder must therefore break down the task required of the computer into simple unambiguous steps. For example, a program to move a mobile robot from indoors to outdoors must include instructions to have the robot locate a doorway and navigate around any obstacles. So the coder must always consider what possible unexpected difficulties a program may encounter.

Program instructions must be presented to the computer in a language it can understand. At the most basic level the computer can understand “machine code”, which moves items around in its memory to perform tasks. This type of obscure low-level code is incredibly tedious as it requires many lines of instruction to perform even a simple task.

Fortunately, over the years, many “high-level” programming languages have been developed that allow the coder to compose instructions in more human-readable form. These modern high-level programs are automatically translated into the machine code that the computer can understand by a “compiler” or by an “interpreter”. In order to become a coder you must typically learn at least one of these high-level programming languages:

- **C** – a powerful compiled language that is closely mapped to machine code and used to develop operating systems
- **C++** – an enhanced compiled language developing on C to provide classes for Object Oriented Programming (OOP)
- **C#** – a modern compiled language designed by Microsoft for the .NET framework and Common Language Infrastructure
- **Java** – a portable compiled language that is designed to run on any platform regardless of the hardware architecture
- **Python** – a dynamic interpreted language that allows both functional and Object Oriented Programming (OOP).

How Java Works?

Java's platform independence is achieved by the use of the Java Virtual Machine.

A Java program consists of one or more files with a .java extension – these are plain old text files

When a Java program is compiled the .java files are fed to a compiler which produces a .class file for each java file

The .class file contains Java bytecode. Bytecode is like machine language, but it is intended for the Java Virtual Machine not a specific chip such as a Pentium or PowerPC chip

To run a Java program the bytecode in a .class file is fed to an interpreter which converts the byte code to machine code for a specific chip (IA to machine code for a specific chip (IA-32, PowerPC)

Some people refer to the interpreter as "The Java Virtual Machine" (JVM)

The interpreter is platform specific because it takes the platform independent bytecode and produces machine language instructions for a particular chip

So a Java program could be run on any type of computer that has a JVM written for it.

First program "HelloWorld.java"

```
/**  
 * A simple program  
 */  
  
public class HelloWorld  
{  
    public static void main(String[] args)  
    {  
        System.out.println( System.out.println( HELLO  WORLD!" );  
    }  
}
```

Programming Errors and its Type

Error is an illegal operation performed by the user which results in abnormal working of the program.

Programming errors often remain undetected until the program is compiled or executed. Some of the errors inhibit the program from getting compiled or executed. Thus errors should be removed before compiling and executing.

The most common errors can be broadly classified as follows.

Syntax error /Compile errors

- caught at compile time.
- compiler did not understand or compiler does not allow

Runtime error

- something “Bad” happens at runtime. Java breaks these into Errors and Exceptions

Logic Error

- program compiles and runs, but does not do what you intended or want

Java Data Types

Identifiers in Java

- letters, digits, _, and \$ (don't use \$. Can confuse the runtime system)
- start with letter, _, or \$
- by convention:
 1. start with a letter
 2. variables and method names lowercase with internal variables and method names, lowercase with internal words capitalized e.g. honkingBigVariableName
 3. constants all caps with _ between internal words e.g. ANOTHER HONKING BIG IDENTIFIER ANOTHER _ HONKING _ BIG _ IDENTIFIER
 4. classes start with capital letter, internal words capitalized, all other lowercase e.g HonkingLongClassName
- Why? To differentiate identifiers that refer to classes from those that refer to variables

Data Types

- **Primitive Data Types**

– byte short int long float double boolean char

```
//dataType identifier;
```

```
int x;
```

```
int y = 10;
```

```
int z, zz;
```

```
int z, zz;
```

```
double a = 12.0;
```

```
boolean done = false, prime = true;
```

– stick with int for integers, double for real numbers

```
char mi = 'D';
```

- **Classes and Objects/ Non Primitive Data type**

– pre defined or user defined data types consisting of constructors, methods and fields (constants and fields (variables) which may be primitives or objects.)

Note: **import** is a reserved word packages and classes can be imported to another class another class does not actually import the code (unlike the C++ include preprocessor command) statement outside the class block

```
import java.util.ArrayList;
```

```
import java.awt.Rectangle;
```

```
public class Foo
```

```
{
```

```
// code for class Foo
```

```
}
```

Data Type	Characteristics	Range
<code>byte</code>	8 bit signed integer	-128 to 127
<code>short</code>	16 bit signed integer	-32768 to 32767
<code>int</code>	32 bit signed integer	-2,147,483,648 to 2,147,483,647
<code>long</code>	64 bit signed integer	-9,223,372,036,854,775,808 to -9,223,372,036,854,775,807
<code>float</code>	32 bit floating point number	$\pm 1.4\text{E-}45$ to $\pm 3.4028235\text{E+}38$
<code>double</code>	64 bit floating point number	$\pm 4.9\text{E-}324$ to $\pm 1.7976931348623157\text{E+}308$
<code>boolean</code>	<code>true</code> or <code>false</code>	NA, note Java booleans cannot be converted to or from other types
<code>char</code>	16 bit, Unicode	Unicode character, <code>\u0000</code> to <code>\uFFFF</code> Can mix with integer types

Creating and Using Objects

Data Type identifier;

Rectangle r1;

- new operator and specified constructor

`r1 = new Rectangle();`

`Rectangle r2 = new Rectangle();`

Behavior - with the dot operator `r2.setSize(10, 20);`

`String s2 = r2.toString();`

Built in Classes

Java has a large built library of classes with lots of useful methods. Once you should become familiar with Objects like Random, String, Math, Integer, Character, Double, etc.

The String Class is a standard Java class – a whole host of behaviors via methods also special (because it used so much)

- `String()` literals exist (no other class has literals) `String name = "Mike D.";`
- String concatenation through the `+` operator

`String firstName = "Mike";`

`String lastName = "Scott";`

`String wholeName = firstName + lastName;`

`String wholeName = firstName + lastName;`

- Any primitive or object on other side of `+` operator from a String automatically converted to String

Standard Output

To print to standard output use

```
System.out.print( expression ); // no newline
```

```
System.out.println( expression ); // newline
```

```
System.out.println(); // just a newline
```

```
System.out.println( "x is: " + x + " \n y is: " + y ); //combo of text and variable with “\n” escape character
```

Table 3-2. *List of Character Escape Sequences*

Character Escape Sequence	Description
'\n'	A linefeed
'\r'	A carriage return
'\f'	A form feed
'\b'	A backspace
'\t'	A tab
'\\'	A backslash
'\"'	A double quote
'\''	A single quote

What Is an Operator?

An operator is a symbol that performs a specific kind of operation on one, two, or three operands, and produces a result. The type of the operator and its operands determines the kind of operation performed on the operands and the type of the result produced.

Operators in Java can be categorized based on two criteria:

- The number of operands they operate on
- The type of operation they perform on the operands

There are three types of operators based on the number of operands. An operator is called a unary, binary, or ternary operator based on the number of operands. If an operator takes one operand, it called a unary operator; if it takes two operands, it called a binary operator; if it takes three operands, it called a ternary operator.

Operators in Java

Operator	Name/Description	Example	Result
+	Addition	3+2	5
-	Subtraction	3-2	1
*	Multiplication	3*2	6
/	Division	10/5	2
%	Modulus	10%5	0
++	Increment and then return value	X=3; ++X	4
	Return value and then increment	X=3; X++	3
--	Decrement and then return value	X=3; --X	2
	Return value and then decrement	X=3; X--	3
&&	Logical “and” evaluates to true when both operands are true	3>2 && 5>3	False
	Logical “or” evaluates to true when either operand is true	3>1 2>5	True
!	Logical “not” evaluates to true if the operand is false	3!=2	True
==	Equal	5==9	False
!=	Not equal	6!=4	True
<	Less than	3<2	False
<=	Less than or equal	5<=2	False
>	Greater than	4>3	True
>=	Greater than or equal	4>=4	True
+	Concatenation(join two strings together)	“A”+”BC”	ABC

Below Table shows the more categories of Operators

Operator	Category	Precedence
Unary Operator	postfix	expression++ expression--
	prefix	++expression --expression +expression -expression ~!
Arithmetic Operator	multiplication	* / %
	addition	+ -
Shift Operator	shift	<< >> >>>
Relational Operator	comparison	< > <= >= instanceof
	equality	== !=
Bitwise Operator	bitwise AND	&
	bitwise exclusive OR	^
	bitwise inclusive OR	
Logical Operator	logical AND	&&
	logical OR	
Ternary Operator	ternary	? :
Assignment Operator	assignment	= += -= *= /= %= &= ^= = <<= >>= >>>=

Java User Input

The `Scanner` class is used to get user input, and it is found in the `java.util` package.

To use the `Scanner` class, create an object of the class and use any of the available methods found in the `Scanner` class documentation. In our example, we will use the `nextLine()` method, which is used to read Strings:

Input Types

In the example above, we used the `nextLine()` method, which is used to read Strings. To read other types, look at the table below:

Method	Description
<code>nextBoolean()</code>	Reads a <code>boolean</code> value from the user
<code>nextByte()</code>	Reads a <code>byte</code> value from the user
<code>nextDouble()</code>	Reads a <code>double</code> value from the user
<code>nextFloat()</code>	Reads a <code>float</code> value from the user
<code>nextInt()</code>	Reads a <code>int</code> value from the user
<code>nextLine()</code>	Reads a <code>String</code> value from the user
<code>nextLong()</code>	Reads a <code>long</code> value from the user
<code>nextShort()</code>	Reads a <code>short</code> value from the user

In the example below, we use different methods to read data of various types:

Example

```
import java.util.Scanner;

class MyClass {

    public static void main(String[] args) {

        Scanner myObj = new Scanner(System.in);

        System.out.println("Enter name, age and salary:");

        // String input

        String name = myObj.nextLine();

        // Numerical input

        int age = myObj.nextInt();

        double salary = myObj.nextDouble();

        // Output input by user
```

```
System.out.println("Name: " + name);  
  
System.out.println("Age: " + age);  
  
System.out.println("Salary: " + salary);  
  
}  
  
}
```

Note: If you enter wrong input (e.g. text in a numerical input), you will get an exception/error message (like "InputMismatchException").

Java If-else Statement

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

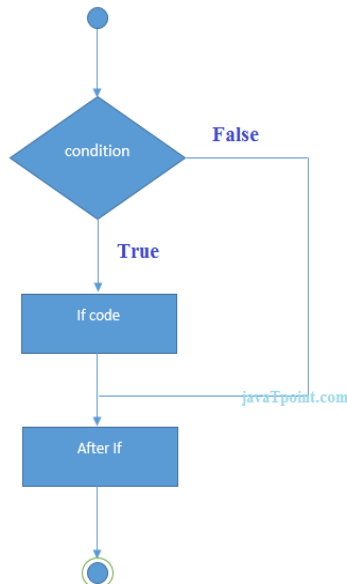
- if statement
- if-else statement
- if-else-if ladder
- nested if statement

Java if Statement

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

Syntax:

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



Example:

//Java Program to demonstrate the use of if statement.

```
public class IfExample {  
public static void main(String[] args) {  
    //defining an 'age' variable  
    int age=20;  
    //checking the age  
    if(age>18){  
        System.out.print("Age is greater than 18");  
    } } }
```

Java if-else Statement

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

Syntax:

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

```
//code if condition is false
```

```
}
```

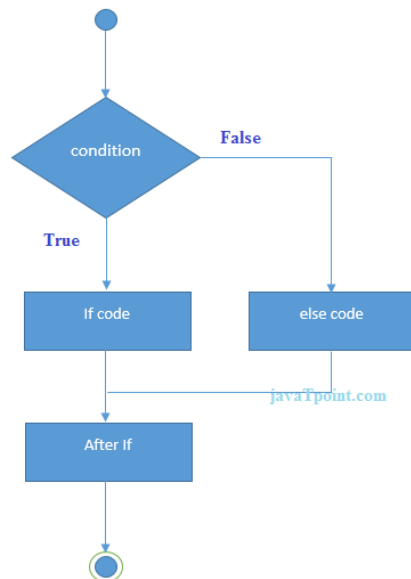
Example:

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

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

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

Flowchart



Leap Year Example:

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

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

Using Ternary Operator

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

Example:

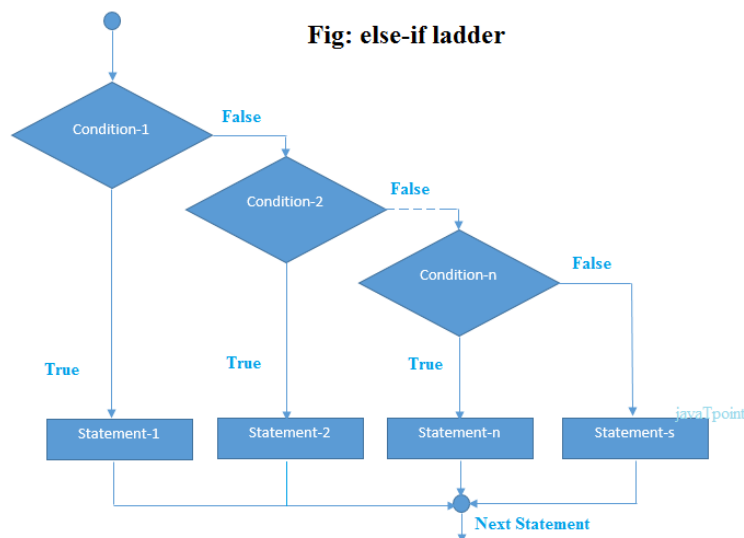
```
public class IfElseTernaryExample {  
    public static void main(String[] args) {  
        int number=13;  
        //Using ternary operator  
        String output=(number%2==0)?"even number":"odd number";  
        System.out.println(output);  
    }  
}
```

Java if-else-if ladder Statement

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

Syntax:

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



Example:

```
//Java Program to demonstrate the use of If else-if ladder.  
//It is a program of grading system for fail, D grade, C grade, B grade, A grade and A+.  
public class IfElseIfExample {
```



```

public static void main(String[] args) {
    int marks=65;
    if(marks<50){
        System.out.println("fail");
    }
    else if(marks>=50 && marks<60){
        System.out.println("D grade");
    }
    else if(marks>=60 && marks<70){
        System.out.println("C grade");
    }
    else if(marks>=70 && marks<80){
        System.out.println("B grade");
    }
    else if(marks>=80 && marks<90){
        System.out.println("A grade");
    } else if(marks>=90 && marks<100){
        System.out.println("A+ grade");
    } else{
        System.out.println("Invalid!");
    } } }

```

Program to check POSITIVE, NEGATIVE or ZERO:

```

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

```

Java Nested if statement

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

Syntax:

```

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

```

Example:

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

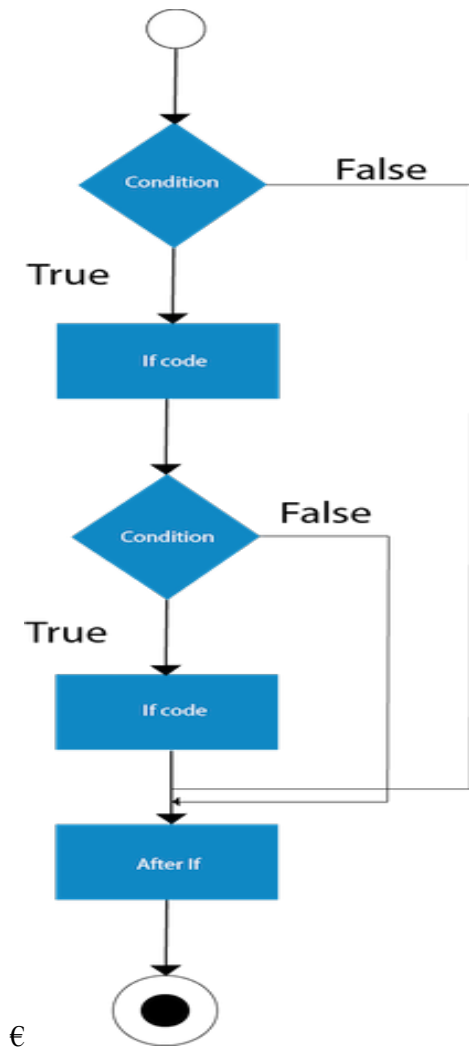
```
public class JavaNestedIfExample {  
public static void main(String[] args) {  
    //Creating two variables for age and weight  
    int age=20;  
    int weight=80;  
    //applying condition on age and weight  
    if(age>=18){  
        if(weight>50){  
            System.out.println("You are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

Example 2:

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

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

Flowchart



Java Switch Statement

The Java *switch statement* executes one statement from multiple conditions. It is like if-else-if ladder statement. The switch statement works with byte, short, int, long, enum types, String and some wrapper types like Byte, Short, Int, and Long. Since Java 7, you can use strings in the switch statement.

In other words, the switch statement tests the equality of a variable against multiple values.

Points to Remember

- There can be *one or N number of case values* for a switch expression.
- The case value must be of switch expression type only. The case value must be *literal or constant*. It doesn't allow variables.
- The case values must be *unique*. In case of duplicate value, it renders compile-time error.
- The Java switch expression must be of *byte, short, int, long (with its Wrapper type), enums and string*.
- Each case statement can have a *break statement* which is optional. When control reaches to the break statement, it jumps the control after the switch expression. If a break statement is not found, it executes the next case.
- The case value can have a *default label* which is optional.

Syntax:

```

switch(expression){
case value1:
//code to be executed;

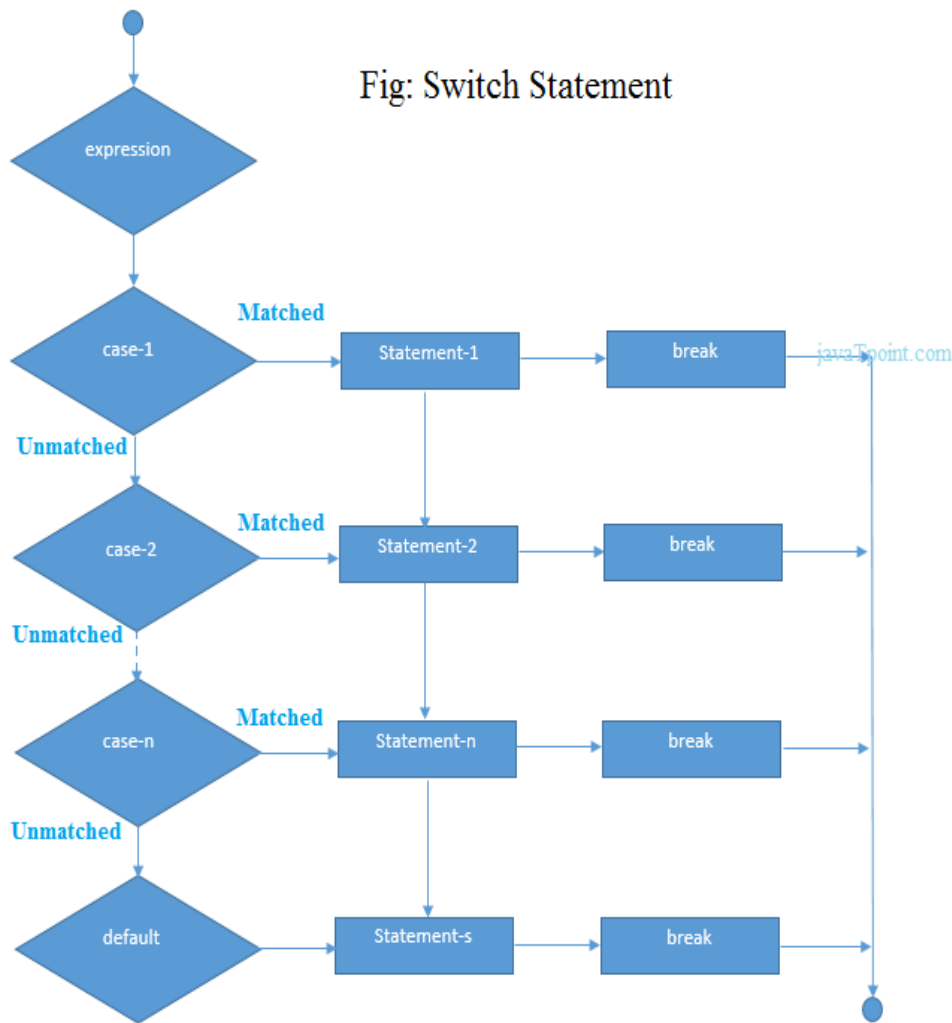
```

```

break; //optional
case value2:
    //code to be executed;
    break; //optional
.....
default:
    code to be executed if all cases are not matched; }
}

```

Flowchart



Example:

```

public class SwitchExample {
    public static void main(String[] args) {
        //Declaring a variable for switch expression
        int number=20;
        //Switch expression
        switch(number){
            //Case statements
            case 10: System.out.println("10");

```

```
break;
case 20: System.out.println("20");
break;
case 30: System.out.println("30");
break;
//Default case statement
default: System.out.println("Not in 10, 20 or 30");
} }
```

Program to check Vowel or Consonant:

If the character is A, E, I, O, or U, it is vowel otherwise consonant. It is not case-sensitive.

```
public class SwitchVowelExample {
public static void main(String[] args) {
    char ch='O';
    switch(ch)
    {
        case 'a':
            System.out.println("Vowel");
            break;
        case 'e':
            System.out.println("Vowel");
            break;
        case 'i':
            System.out.println("Vowel");
            break;
        case 'o':
            System.out.println("Vowel");
            break;
        case 'u':
            System.out.println("Vowel");
            break;
        case 'A':
            System.out.println("Vowel");
            break;
        case 'E':
            System.out.println("Vowel");
            break;
        case 'I':
            System.out.println("Vowel");
            break;
        case 'O':
            System.out.println("Vowel");
            break;
        case 'U':
            System.out.println("Vowel");
            break;
        default:
```

```
        System.out.println("Consonant");
    } } }
```

Java Switch Statement is fall-through

The Java switch statement is fall-through. It means it executes all statements after the first match if a break statement is not present.

Example:

```
//Java Switch Example where we are omitting the
//break statement
public class SwitchExample2 {
    public static void main(String[] args) {
        int number=20;
        //switch expression with int value
        switch(number){
            //switch cases without break statements
            case 10: System.out.println("10");
            case 20: System.out.println("20");
            case 30: System.out.println("30");
            default: System.out.println("Not in 10, 20 or 30");
        } }
```

Java Switch Statement with String

Java allows us to use strings in switch expression since Java SE 7. The case statement should be string literal.

Example:

```
//Java Program to demonstrate the use of Java Switch
//statement with String
public class SwitchStringExample {
    public static void main(String[] args) {
        //Declaring String variable
        String levelString="Expert";
        int level=0;
        //Using String in Switch expression
        switch(levelString){
            //Using String Literal in Switch case
            case "Beginner": level=1;
            break;
            case "Intermediate": level=2;
            break;
            case "Expert": level=3;
            break;
            default: level=0;
            break;
        }
    }
}
```

```

System.out.println("Your Level is: "+level);
} }

```

Loops in Java

In programming languages, loops are used to execute a set of instructions/functions repeatedly when some conditions become true. There are three types of loops in java.

- for loop
- while loop
- do-while loop

Java For Loop vs While Loop vs Do While Loop

Comparison	for loop	while loop	do while loop
Introduction	The Java for loop is a control flow statement that iterates a part of the programs multiple times.	The Java while loop is a control flow statement that executes a part of the programs repeatedly on the basis of given boolean condition.	The Java do while loop is a control flow statement that executes a part of the programs at least once and the further execution depends upon the given boolean condition.
When to use	If the number of iteration is fixed, it is recommended to use for loop.	If the number of iteration is not fixed, it is recommended to use while loop.	If the number of iteration is not fixed and you must have to execute the loop at least once, it is recommended to use the do-while loop.
Syntax	for(init;condition;incr/decr){ // code to be executed }	while(condition){ //code to be executed }	do { //code to be executed } while(condition);
Example	//for loop for(int i=1;i<=10;i++){ System.out.println(i); }	//while loop int i=1; while(i<=10){ System.out.println(i); i++; }	//do-while loop int i=1; do { System.out.println(i); i++; } while(i<=10);
Syntax for infinitive loop	for(;;){ //code to be executed }	while(true){ //code to be executed }	do { //code to be executed } while(true);

Java Nested For Loop

If we have a for loop inside the another loop, it is known as nested for loop. The inner loop executes completely whenever outer loop executes.

Example:

```

public class NestedForExample {
public static void main(String[] args) {
//loop of i
for(int i=1;i<=3;i++){
//loop of j
for(int j=1;j<=3;j++){
    System.out.println(i+" "+j);
} //end of i
} //end of j
}
}

```

Pyramid Example 1:

```

public class PyramidExample {
public static void main(String[] args) {
for(int i=1;i<=5;i++){
for(int j=1;j<=i;j++){
    System.out.print("* ");
}
System.out.println();//new line
}
} }

```

Java for-each Loop

The for-each loop is used to traverse array or collection in java. It is easier to use than simple for loop because we don't need to increment value and use subscript notation.

It works on elements basis not index. It returns element one by one in the defined variable.

Syntax:

```

for(Type var:array){
//code to be executed
}

```

Example:

```

//Java For-each loop example which prints the
//elements of the array
public class ForEachExample {
public static void main(String[] args) {
    //Declaring an array
    int arr[]={12,23,44,56,78};
    //Printing array using for-each loop
    for(int i:arr){
        System.out.println(i);
    } }
}

```

Java Break Statement

When a break statement is encountered inside a loop, the loop is immediately terminated and the program control resumes at the next statement following the loop.

The Java *break* is used to break loop or switch statement. It breaks the current flow of the program at specified condition. In case of inner loop, it breaks only inner loop.

We can use Java break statement in all types of loops such as for loop, while loop and do-while loop.

Syntax:

```

jump-statement;
break;

```

Java Break Statement with Loop

Example:

//Java Program to demonstrate the use of break statement

//inside the for loop.

```
public class BreakExample {  
public static void main(String[] args) {  
    //using for loop  
    for(int i=1;i<=10;i++){  
        if(i==5){  
            //breaking the loop  
            break;  
        }  
        System.out.println(i);  
    }  
}
```

Java Continue Statement

The continue statement is used in loop control structure when you need to jump to the next iteration of the loop immediately. It can be used with for loop or while loop.

The Java *continue statement* is used to continue the loop. It continues the current flow of the program and skips the remaining code at the specified condition. In case of an inner loop, it continues the inner loop only.

We can use Java continue statement in all types of loops such as for loop, while loop and do-while loop.

Syntax:

jump-statement;

continue;

Java Continue Statement Example

Example:

//Java Program to demonstrate the use of continue statement

//inside the for loop.

```
public class ContinueExample {  
public static void main(String[] args) {  
    //for loop  
    for(int i=1;i<=10;i++){  
        if(i==5){  
            //using continue statement  
            continue;//it will skip the rest statement  
        }  
        System.out.println(i);  
    }  
}  
}
```

Chapter Three: Object Oriented Programming

What is OOP?

Object Oriented programming is a programming style which is associated with the concepts like **class**, **object**, **Inheritance**, **Encapsulation**, **Abstraction**, **Polymorphism**. Most popular programming languages like Java, C++, C#, Ruby, etc. follow an object-oriented programming paradigm.

Classes and Objects in Java

Classes and Objects are basic concepts of Object Oriented Programming which revolve around the real life entities.

Class

A class represents the set of properties or methods that are common to all objects of one type.

```
class Animal {}  
class Dog extends Animal {}  
class Cat extends Animal {}  
public class Test
```

Object

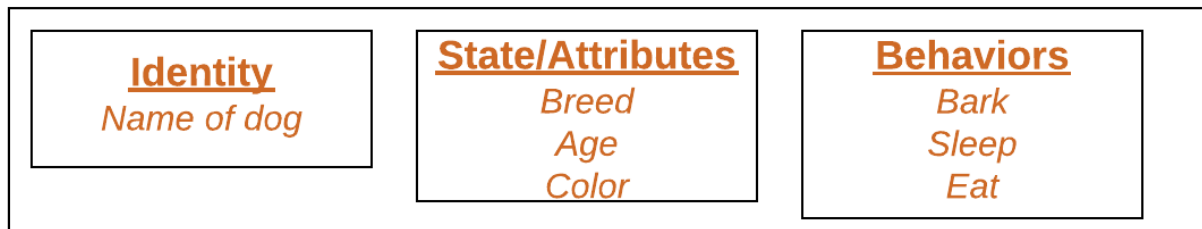
It is a basic unit of Object Oriented Programming and represents the real life entities. A typical Java program creates many objects, which as you know, interact by invoking methods. An object consists of :

State: It is represented by attributes of an object. It also reflects the properties of an object.

Behavior: It is represented by methods of an object. It also reflects the response of an object with other objects.

Identity: It gives a unique name to an object and enables one object to interact with other objects.

Example of an object : dog



Objects correspond to things found in the real world. For example, a graphics program may have objects such as “circle”, “square”, “menu”. An online shopping system might have objects such as “shopping cart”, “customer”, and “product”.

Declaring Objects (Also called instantiating a class)

As we declare variables like

```
Type Var_name;
```

This notifies the compiler that we will use name to refer to data whose type is “type”. So for reference variable, type must be strictly a concrete

Syntax:

`ClassName ObjectName=new ClassName/Constructor(parameters if any);`

Eg:

`Dog tuffy=new Dog();`

The “new” operator instantiates class by allocating memory for new object and also invokes the class constructor.

// Class Declaration

```
public class Dog
{
    // Instance Variables
    String name;
    String breed;
    int age;
    String color;
    // Constructor Declaration of Class
    // Default Constructor Declaration of Class
    public Dog(String name, String breed, int age, String color)
    {
        System.out.print("THIS IS default constructor");
    }
    //Parameterized Constructor Declaration of Class
    public Dog(String name, String breed, int age, String color)
    {
        this.name = name;
        this.breed = breed;
        this.age = age;
        this.color = color;
    }
    // User Defined method
    public String getName()
    {
        return name;
    }
    //Main method
    public static void main(String[] args)
    {
        Dog tuffy = new Dog("tuffy","papillon", 5, "white");
        tuffy.getName(); //Calling method getName()
    }
}
```

This class contains a main() method, user defined method, default constructor and a parameterized constructor. We can recognize a constructor because its declaration uses the same name as the class and it has no return type.

Object Oriented Programming: Abstraction

Abstraction basically deals with hiding the details and showing the essential things to the user. In reality, there is a lot of code that runs in the background. So you don't know the internal processing of how a call is generated, that's the beauty of abstraction. You can achieve abstraction in two ways:

- a) Abstract Class
- b) Interface

Abstract class: Abstract class in Java contains the ‘abstract’ keyword. If a class is declared abstract, it cannot be instantiated, which means you cannot create an object of an abstract class. Also, an abstract class can contain abstract as well as concrete methods.

Let's look at the syntax of an abstract class:

```
Abstract class Mobile { // abstract class mobile
Abstract void run();    // abstract method
```

Interface: Interface in Java is a blueprint of a class or you can say it is a collection of abstract methods and static constants. In an interface, each method is public and abstract but it does not contain any constructor. So an interface basically is a group of related methods with empty bodies. Let us understand interfaces better by taking an example of a ‘ParentCar’ interface with its related methods.

```
public interface ParentCar {
public void changeGear( int newValue);
public void speedUp(int increment);
public void applyBrakes(int decrement);
}
```

These methods need be present for every car, right? But their working is going to be different.

Object Oriented Programming: Encapsulation

Encapsulation is a mechanism where you bind your data and code together as a single unit. It also means to hide your data in order to make it safe from any modification. What does this mean? The best way to understand encapsulation is to look at the example of a medical capsule, where the drug is always safe inside the capsule. Similarly, through encapsulation the methods and variables of a class are well hidden and safe.

We can achieve encapsulation in Java by:

- Declaring the variables of a class as private.
- Providing public setter and getter methods to and view the variables values.



Let us look at the code below to get a better understanding of encapsulation:

```
public class Employee
{
    private String name;
    public String getName()
    {
        return name;
    }
    public void setName(String name)
    {
        this.name = name;
    }
    public static void main(String[] args)
    {
    }
}
```

Let us try to understand the above code. I have created a class **Employee** which has a private variable **name**. We have then created a getter and setter methods through which we can get and set the name of an employee. Through these methods, any class which wishes to access the name variable has to do it using these getter and setter methods.

Advantage of Encapsulation in Java

By providing only a setter or getter method, you can make the class **read-only** or **write-only**. In other words, you can skip the getter or setter methods.

It provides you the **control over the data**. Suppose you want to set the value of id which should be greater than 100 only, you can write the logic inside the setter method. You can write the logic not to store the negative numbers in the setter methods.

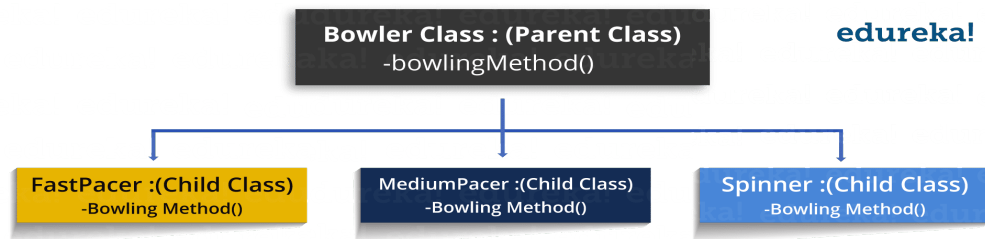
It is a way to achieve **data hiding** in Java because other class will not be able to access the data through the private data members.

The encapsulate class is **easy to test**. So, it is better for unit testing.

Object Oriented Programming: Polymorphism

Polymorphism means taking many forms, where ‘poly’ means many and ‘morph’ means forms. It is the ability of a variable, function or object to take on multiple forms. In other words, polymorphism allows you define one interface or method can have multiple implementations.

Let’s understand this by taking a real-life example and how this concept fits into Object oriented programming.



edureka!

Let's consider this real world scenario in cricket, we know that there are different types of bowlers i.e. Fast bowlers, Medium pace bowlers and spinners. As you can see in the above figure, there is a parent class- **BowlerClass** and it has three child classes: **FastPacer**, **MediumPacer** and **Spinner**. Bowler class has **bowlingMethod()** where all the child classes are inheriting this method.

Method OverLoading

Let us look at the following code to understand how the method overloading works:

```

class Adder {
    Static int add(int a, int b)
    {
        return a+b;
    }
    static double add( double a, double b)
    {
        return a+b;
    }
}

public static void main(String args[])
{
    System.out.println(add(11,11));
    System.out.println(add(12.3,12.6));
}
  
```

Method overloading vs overriding

Overriding

```

class Dog{
    public void bark(){
        System.out.println("woof ");
    }
}
class Hound extends Dog{
    public void sniff(){
        System.out.println("sniff ");
    }
    public void bark(){
        System.out.println("bowl");
    }
}
  
```

Same Method Name,
Same parameter

Overloading

```

class Dog{
    public void bark(){
        System.out.println("woof ");
    }
    //overloading method
    public void bark(int num){
        for(int i=0; i<num; i++)
            System.out.println("woof ");
    }
}
  
```

Same Method Name,
Different Parameter

“this” keyword in java

```

class Student{
int rollno;
String name;
float fee;
Student(int rollno,String name,float fee){
this.rollno=rollno;
this.name=name;
this.fee=fee;
}
void display(){System.out.println(rollno+" "+name+" "+fee);}
}
class TestThis2{
public static void main(String args[]){
Student s1=new Student(111,"ankit",5000f);
Student s2=new Student(112,"sumit",6000f);
s1.display();
s2.display();
}}

```

Java Constructors

A constructor in Java is a **special method** that is used to initialize objects. The constructor is called when an object of a class is created. It can be used to set initial values for object attributes:

1.Default Constructor

```

// Create a MyClass class
public class MyClass {
    int x; // Create a class attribute
    // Create a class constructor for the MyClass class
    public MyClass() {
        x = 5; // Set the initial value for the class attribute x
    }
    public static void main(String[] args) {
        MyClass myObj = new MyClass(); // Create an object of class MyClass (This will call the constructor)
        System.out.println(myObj.x); // Print the value of x
    }
}

```

Note that the constructor name must **match the class name**, and it cannot have a **return type** (like void).

2.Constructor Parameters

Constructors can also take parameters, which is used to initialize attributes.

The following example adds an `int y` parameter to the constructor. Inside the constructor we set `x` to `y` (`x=y`). When we call the constructor, we pass a parameter to the constructor (5), which will set the value of `x` to 5:

```

public class Car {
    int modelYear;
    String modelName;

    public Car(int year, String name) {
        modelYear = year;
        modelName = name;
    }
}

```

```

}
public static void main(String[] args) {
    Car myCar = new Car(1969, "Mustang");
    System.out.println(myCar.modelYear + " " + myCar.modelName);
}
}

```

Function Call in Java

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

Example of call by value in java

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

```

class Operation{
    int data=50;
    void change(int data){
        data=data+100;//changes will be in the local variable only
    }
    public static void main(String args[]){
        Operation op=new Operation();
        System.out.println("before change "+op.data);
        op.change(500);
        System.out.println("after change "+op.data);
    }
}

```

Call by reference

```

class Number {
    int x;
}
class CallByReference {
    public static void main ( String[] args ) {
        Number a = new Number();
        a.x=3;
        System.out.println("Value of a.x before calling increment() is "+a.x);
        increment(a);
        System.out.println("Value of a.x after calling increment() is "+a.x);
    }
    public static void increment(Number n) {
        System.out.println("Value of n.x before incrementing x is "+n.x);
        n.x=n.x+1;
        System.out.println("Value of n.x after incrementing x is "+n.x);
    }
}

```


Access Modifiers in Java

There are two types of modifiers in Java: **access modifiers**.

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

There are four types of Java access modifiers:

1. **Private:** The access level of a private modifier is only within the class. It cannot be accessed from outside the class.
2. **Default:** The access level of a default modifier is only within the package. It cannot be accessed from outside the package. If you do not specify any access level, it will be the default.
3. **Protected:** The access level of a protected modifier is within the package and outside the package through child class. If you do not make the child class, it cannot be accessed from outside the package.
4. **Public:** The access level of a public modifier is everywhere. It can be accessed from within the class, outside the class, within the package and outside the package.

“return” keyword in Java

return is a reserved keyword in Java i.e, we can't use it as an identifier. It is used to exit from a method, with or without a value.

return can be used with methods in two ways:

Methods returning a value : For methods that define a return type, return statement must be immediately followed by return value.

```
// Java program to illustrate usage
// of return keyword
class A {
    // Since return type of RR method is double
    // so this method should return double value
    double RR(double a, double b)
    {
        double sum = 0;
        sum = (a + b) / 2.0;
        // return statement below:
        return sum;
    }
    public static void main(String[] args)
    {
        System.out.println(new A().RR(5.5, 6.5));
    }
}
```

Nested Classes in Java

In java, it is possible to define a class within another class, such classes are known as *nested* classes.

```
class OuterClass
```

```
{
```

```
...
```

```
class NestedClass
```

```
{
```

```
...
```

```
}
```

```
}
```

Recursion in Java

Recursion in java is a process in which a method calls itself continuously. A method in java that calls itself is called recursive method.

It makes the code compact but complex to understand.

Syntax:

```
returntype methodname(){  
//code to be executed  
methodname();//calling same method  
}
```

Java Recursion Example 1: Infinite times

```
public class RecursionExample1 {  
static void p(){  
System.out.println("hello");  
p();  
}  
public static void main(String[] args) {  
p();  
}  
}
```

Chapter- 4[Inheritance, Packaging]

Inheritance in Java

Inheritance in Java is a mechanism in which one object acquires all the properties and behaviors of a parent object. It is an important part of OOPs (Object Oriented programming system).

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

Inheritance represents the **IS-A relationship** which is also known as a *parent-child* relationship.

Why use inheritance in java

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

Terms used in Inheritance

- **Class:** A class is a group of objects which have common properties. It is a template or blueprint from which objects are created.
- **Sub Class/Child Class:** Subclass is a class which inherits the other class. It is also called a derived class, extended class, or child class.
- **Super Class/Parent Class:** Superclass is the class from where a subclass inherits the features. It is also called a base class or a parent class.
- **Reusability:** As the name specifies, reusability is a mechanism which facilitates you to reuse the fields and methods of the existing class when you create a new class. You can use the same fields and methods already defined in the previous class.

The syntax of Java Inheritance

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

The **extends keyword** indicates that you are making a new class that derives from an existing class. The meaning of "extends" is to increase the functionality.

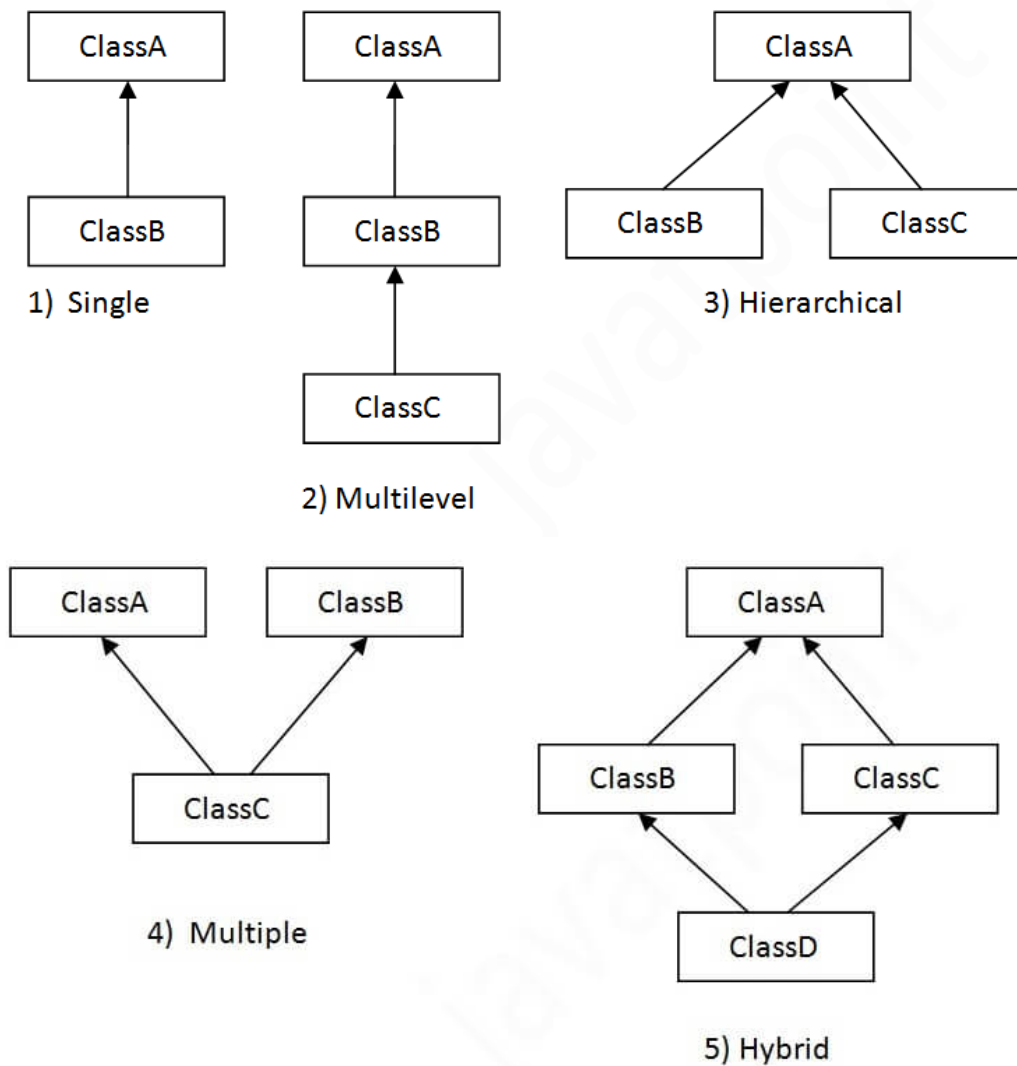
```
class Employee{
    float salary=40000;
}
class Programmer extends Employee{
    int bonus=10000;
    public static void main(String args[]){
        Programmer p=new Programmer();
        System.out.println("Programmer salary is:"+p.salary);
    }
}
```

```

    System.out.println("Bonus of Programmer is:"+p.bonus);
}
}

```

Types of Inheritance



Single Inheritance Example

```

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

```

Multilevel Inheritance Example

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

Hierarchical Inheritance Example

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

Q) Why multiple inheritance is not supported in java?

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

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

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

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

public static void main(String args[]){
    C obj=new C();
    obj.msg();//Now which msg() method would be invoked?
}
}
```

Object class in Java

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

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

getClass(), hashCode(), equals(Object obj), clone() throws CloneNotSupportedException, toString(), notify(), etc

Abstraction in Java

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

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

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

- It cannot be instantiated.
- It can have constructors and static methods also.
- It can have final methods which will force the subclass not to change the body of the method.

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

```

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

```

Final Classes and Methods

Inheritance is surely one of the highly useful features in Java. But at times, it may be desired that a class should not be extendable by other classes to prevent exploitation. For such purpose, we have the final keyword. We have already seen even what final variables are. Final classes and methods are also similar. A class declared as final cannot be extended while a method declared as final cannot be overridden in its subclasses. A method or a class is declared to be final using the final keyword. Though a final class cannot be extended, it can extend other classes. In simpler words, a final class can be a sub class but not a super class.

```

final public class A {
    //code
}

```

What is Package in Java?

A Package is a collection of related classes. It helps organize your classes into a folder structure and make it easy to locate and use them. More importantly, it helps improve re-usability.

Creating package

```
package nameOfPackage;
```

also

```
package p1.p2;
```

Importing package:

```
import packageName;
```

also

```
import p1.*;
```

Java Interface

Another way to achieve abstraction in Java, is with interfaces.

An interface is a completely "abstract class" that is used to group related methods with empty bodies:

Example

```
// interface
```

```
interface Animal {  
  
    public void animalSound(); // interface method (does not have a body)  
  
    public void run(); // interface method (does not have a body)  
  
}
```

```
// Interface
```

```
interface Animal {  
  
    public void animalSound(); // interface method (does not have a body)  
  
    public void sleep(); // interface method (does not have a body)  
  
}
```

```
// Pig "implements" the Animal interface
```

```
class Pig implements Animal {  
  
    public void animalSound() {  
  
        // The body of animalSound() is provided here  
  
        System.out.println("The pig says: wee wee");  
  
    }  
  
    public void sleep() {  
  
        // The body of sleep() is provided here  
  
        System.out.println("Zzz");  
  
    }  
  
}
```

```
class MyMainClass {
```



```
public static void main(String[] args) {  
    Pig myPig = new Pig(); // Create a Pig object  
    myPig.animalSound();  
    myPig.sleep();  
}  
}
```

Also

```
interface FirstInterface {  
    public void myMethod(); // interface method  
}  
  
interface SecondInterface {  
    public void myOtherMethod(); // interface method  
}  
  
class DemoClass implements FirstInterface, SecondInterface {  
    public void myMethod() {  
        System.out.println("Some text..");  
    }  
    public void myOtherMethod() {  
        System.out.println("Some other text...");  
    }  
}
```

Chapter- 5[Exception Handling]

Error : An Error “indicates serious problems that a reasonable application should not try to catch.”

Exceptions : An Exception “indicates conditions that a reasonable application might want to catch.”

Both Errors and Exceptions are the subclasses of java.lang.Throwable class. Errors are the conditions which cannot get recovered by any handling techniques. It surely cause termination of the program abnormally. Errors belong to unchecked type and mostly occur at runtime. Some of the examples of errors are Out of memory error or a System crash error.

Exceptions are the conditions that occur at runtime and may cause the termination of program. But they are recoverable using try, catch and throw keywords. Exceptions are divided into two categories : checked and unchecked exceptions. Checked exceptions like IOException known to the compiler at compile time while unchecked exceptions like ArrayIndexOutOfBoundsException known to the compiler at runtime. It is mostly caused by the program written by the programmer.

```
public class ExceptionEg {  
  
    public static void main(String[] args)  
    {  
        int a = 5, b = 0;  
  
        // Attempting to divide by zero  
        try {  
            int c = a / b;  
        }  
        catch (ArithmeticException e) {  
            e.printStackTrace();  
        }  
    }  
}
```

ERRORS

EXCEPTIONS

Recovering from Error is not possible.

We can recover from exceptions by either using try-catch block or throwing exceptions back to caller.

All errors in java are unchecked

Exceptions include both checked as

type.	well as unchecked type.
Errors are mostly caused by the environment in which program is running.	Program itself is responsible for causing exceptions.
Errors occur at runtime and not known to the compiler.	All exceptions occurs at runtime but checked exceptions are known to compiler while unchecked are not.
They are defined in java.lang.Error package.	They are defined in java.lang.Exception package
Examples :	Examples : Checked Exceptions : SQLException, IOException Unchecked Exceptions : ArrayIndexOutOfBoundsException, NullPointerException, ArithmeticException.
java.lang.StackOverflowError, java.lang.OutOfMemoryError	

throw

The throw keyword in Java is used to explicitly throw an exception from a method or any block of code. We can throw either checked or unchecked exception. The throw keyword is mainly used to throw custom exceptions.

```
class ThrowExcep
{
    static void fun()
    {
        try
        {
```

```

        throw new NullPointerException("demo");
    }
    catch(NullPointerException e)
    {
        System.out.println("Caught inside fun().");
        throw e; // rethrowing the exception
    }
}

```

throws

throws is a keyword in Java which is used in the signature of method to indicate that this method might throw one of the listed type exceptions. The caller to these methods has to handle the exception using a try-catch block.

```

class tst
{
    public static void main(String[] args)throws InterruptedException
    {
        Thread.sleep(10000);
        System.out.println("Hello Geeks");
    }
}

```

Java finally block is a block that is used *to execute important code* such as closing connection, stream etc.

Java finally block is always executed whether exception is handled or not.

Java finally block follows try or catch block.

```

class TestFinallyBlock{
    public static void main(String args[]){
    try{
        int data=25/5;
        System.out.println(data);
    }
    catch(NullPointerException e){System.out.println(e);}
    finally{System.out.println("finally block is always executed");}
    System.out.println("rest of the code...");
    }
}

```

Chapter 6[String Handling]

Java – String Class and its methods explained with examples

BY CHAITANYA SINGH | FILED UNDER: STRING HANDLING

String is a sequence of characters, for e.g. “Hello” is a string of 5 characters. In java, string is an immutable object which means it is constant and cannot be changed once it has been created. In this tutorial we will learn about String class and String methods in detail along with many other Java String tutorials.

Creating a String

There are two ways to create a String in Java

1. String literal
2. Using new keyword

String literal

In java, Strings can be created like this: Assigning a String literal to a String instance:

```
String str1 = "Welcome";  
String str2 = "Welcome";
```

The problem with this approach: As I stated in the beginning that String is an object in Java. However we have not created any string object using new keyword above. The compiler does that task for us it creates a string object having the string literal (that we have provided , in this case it is “Welcome”) and assigns it to the provided string instances.

But if the object already exist in the memory it does not create a new Object rather it assigns the same old object to the new instance, that means even though we have two string instances above(str1 and str2) compiler only created one string object (having the value “Welcome”) and assigned the same to both the instances. For example there are 10 string instances that have same value, it means that in memory there is only one object having the value and all the 10 string instances would be pointing to the same object.

What if we want to have two different object with the same string? For that we would need to create strings using **new keyword**.

Using New Keyword

As we saw above that when we tried to assign the same string object to two different literals, compiler only created one object and made both of the literals to point the same object. To overcome that approach we can create strings like this:

```
String str1 = new String("Welcome");  
String str2 = new String("Welcome");
```

In this case compiler would create two different object in memory having the same string.

A Simple Java String Example

```
public class Example{
    public static void main(String args[]){
        //creating a string by java string literal
        String str = "Beginnersbook";
        char arrch[]={'h','e','l','l','o'};
        //converting char array arrch[] to string str2
        String str2 = new String(arrch);

        //creating another java string str3 by using new keyword
        String str3 = new String("Java String Example");

        //Displaying all the three strings
        System.out.println(str);
        System.out.println(str2);
        System.out.println(str3);
    }
}
```

Output:

```
Beginnersbook
hello
Java String Example
```

Java String Methods

Here are the list of the methods available in the Java String class. These methods are explained in the separate tutorials with the help of examples. Links to the tutorials are provided below:

1. [char charAt\(int index\)](#): It returns the character at the specified index. Specified index value should be between 0 to length() -1 both inclusive. It throws `IndexOutOfBoundsException` if `index<0||>= length of String`.
2. [boolean equals\(Object obj\)](#): Compares the string with the specified string and returns true if both matches else false.
3. [boolean equalsIgnoreCase\(String string\)](#): It works same as equals method but it doesn't consider the case while comparing strings. It does a case insensitive comparison.
4. [int compareTo\(String string\)](#): This method compares the two strings based on the Unicode value of each character in the strings.
5. [int compareToIgnoreCase\(String string\)](#): Same as `CompareTo` method however it ignores the case during comparison.
6. [boolean startsWith\(String prefix, int offset\)](#): It checks whether the substring (starting from the specified offset index) is having the specified prefix or not.

7. boolean startsWith(String prefix): It tests whether the string is having specified prefix, if yes then it returns true else false.
8. boolean endsWith(String suffix): Checks whether the string ends with the specified suffix.
9. int hashCode(): It returns the hash code of the string.
10. int indexOf(int ch): Returns the index of first occurrence of the specified character ch in the string.
11. int indexOf(int ch, int fromIndex): Same as indexOf method however it starts searching in the string from the specified fromIndex.
12. int lastIndexOf(int ch): It returns the last occurrence of the character ch in the string.
13. int lastIndexOf(int ch, int fromIndex): Same as lastIndexOf(int ch) method, it starts search from fromIndex.
14. int indexOf(String str): This method returns the index of first occurrence of specified substring str.
15. int lastIndexOf(String str): Returns the index of last occurrence of string str.
16. String substring(int beginIndex): It returns the substring of the string. The substring starts with the character at the specified index.
17. String substring(int beginIndex, int endIndex): Returns the substring. The substring starts with character at beginIndex and ends with the character at endIndex.
18. String concat(String str): Concatenates the specified string “str” at the end of the string.
19. String replace(char oldChar, char newChar): It returns the new updated string after changing all the occurrences of oldChar with the newChar.
20. boolean contains(CharSequence s): It checks whether the string contains the specified sequence of char values. If yes then it returns true else false. It throws NullPointerException of ‘s’ is null.
21. String toUpperCase(Locale locale): Converts the string to upper case string using the rules defined by specified locale.
22. String toUpperCase(): Equivalent to toUpperCase(Locale.getDefault()).
23. public String intern(): This method searches the specified string in the memory pool and if it is found then it returns the reference of it, else it allocates the memory space to the specified string and assign the reference to it.
24. public boolean isEmpty(): This method returns true if the given string has 0 length. If the length of the specified Java String is non-zero then it returns false.
25. public static String join(): This method joins the given strings using the specified delimiter and returns the concatenated Java String
26. String replaceFirst(String regex, String replacement): It replaces the first occurrence of substring that fits the given regular expression “regex” with the specified replacement string.
27. String replaceAll(String regex, String replacement): It replaces all the occurrences of substrings that fits the regular expression regex with the replacement string.
28. String[] split(String regex, int limit): It splits the string and returns the array of substrings that matches the given regular expression. limit is a result threshold here.
29. String[] split(String regex): Same as split(String regex, int limit) method however it does not have any threshold limit.
30. String toLowerCase(Locale locale): It converts the string to lower case string using the rules defined by given locale.

31. public static String format(): This method returns a formatted java String
32. String toLowerCase(): Equivalent to toLowerCase(Locale. getDefault()).
33. String trim(): Returns the substring after omitting leading and trailing white spaces from the original string.
34. char[] toCharArray(): Converts the string to a character array.
35. static String copyValueOf(char[] data): It returns a string that contains the characters of the specified character array.
36. static String copyValueOf(char[] data, int offset, int count): Same as above method with two extra arguments – initial offset of subarray and length of subarray.
37. void getChars(int srcBegin, int srcEnd, char[] dest, int destBegin): It copies the characters of **src** array to the **dest** array. Only the specified range is being copied(srcBegin to srcEnd) to the dest subarray(starting from destBegin).
38. static String valueOf(): This method returns a string representation of passed arguments such as int, long, float, double, char and char array.
39. boolean contentEquals(StringBuffer sb): It compares the string to the specified string buffer.
40. boolean regionMatches(int srcoffset, String dest, int destoffset, int len): It compares the substring of input to the substring of specified string.
41. boolean regionMatches(boolean ignoreCase, int srcoffset, String dest, int destoffset, int len): Another variation of regionMatches method with the extra boolean argument to specify whether the comparison is case sensitive or case insensitive.
42. byte[] getBytes(String charsetName): It converts the String into sequence of bytes using the specified charset encoding and returns the array of resulted bytes.
43. byte[] getBytes(): This method is similar to the above method it just uses the default charset encoding for converting the string into sequence of bytes.
44. int length(): It returns the length of a String.
45. boolean matches(String regex): It checks whether the String is matching with the specified regular expression regex.
46. int codePointAt(int index): It is similar to the charAt method however it returns the Unicode code point value of specified index rather than the character itself.

StringBuffer is a peer class of **String** that provides much of the functionality of strings. String represents fixed-length, immutable character sequences while StringBuffer represents growable and writable character sequences.

StringBuffer may have characters and substrings inserted in the middle or appended to the end. It will automatically grow to make room for such additions and often has more characters preallocated than are actually needed, to allow room for growth.

```
import java.io.*;
```

```
class GFG {
```

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

```
    {
```

```
        StringBuffer s = new StringBuffer("GeeksGeeks");
```

```
        s.insert(5, "for");
```

```
        System.out.println(s); // returns GeeksforGeeks
```

```
        s.insert(0, 5);
```



```
System.out.println(s); // returns 5GeeksforGeeks
```

```
s.insert(3, true);
```

```
System.out.println(s); // returns 5GettrueeksforGeeks
```

```
s.insert(5, 41.35d);
```

```
System.out.println(s); // returns 5Getr41.35ueeksforGeeks
```

```
s.insert(8, 41.35f);
```

```
System.out.println(s); // returns 5Getr41.41.3535ueeksforGeeks
```

```
char geeks_arr[] = { 'p', 'a', 'w', 'a', 'n' };
```

```
// insert character array at offset 9
```

```
s.insert(2, geeks_arr);
```

```
System.out.println(s); // returns 5Gpawanetr41.41.3535ueeksforGeeks
```

```
}
```

```
}
```

Chapter -7[Threads]

Unlike many other computer languages, Java provides built-in support for multithreading. Multithreading in Java contains two or more parts that can run concurrently. A Java thread is actually a lightweight process.

This article will introduce you to all the Java Thread concepts many people find tricky or difficult to understand.

I'll be covering the following topics:

1. What is a Java Thread?
2. The Java Thread Model
3. Multithreading in Java
4. Main Java Thread
5. How to Create a Java Thread?

Before we proceed with the first topic, consider this example:

Imagine a stockbroker application with lots of complex capabilities, like

- Downloading the last stock prices
- Checking prices for warnings
- Analyzing historical data for a particular company

These are time-consuming functions. In a single-threaded runtime environment, these actions execute one after another. The next action can happen only when the previous one has finished.

If a historical analysis takes half an hour, and the user selects to perform a download and check afterward, the warning may come too late to buy or sell stock. This is the sort of application that cries out for multithreading. Ideally, the download should happen in the background (that is, in another thread). That way, other processes could happen at the same time so that, for example, a warning could be communicated instantly. All the while, the user is interacting with other parts of the application. The analysis, too, could happen in a separate thread, so the user can work with the rest of the application while the results are being calculated.

This is where a Java thread helps.

What Is a Java Thread?

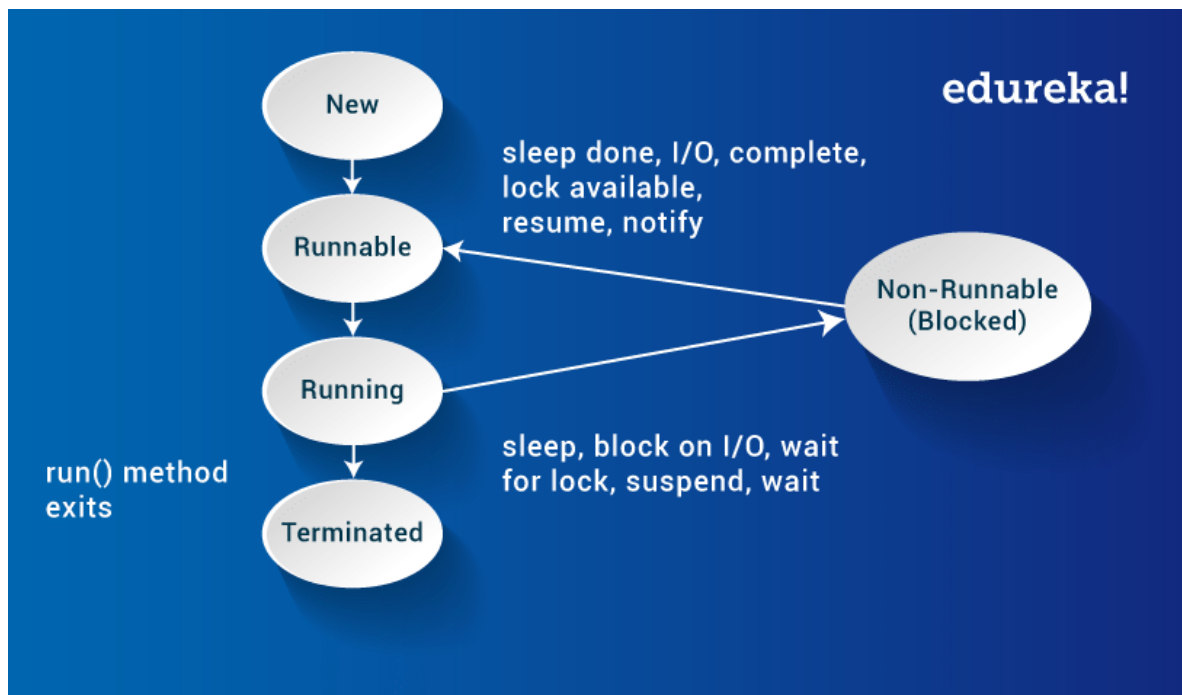
A thread is actually a lightweight process. Unlike many other computer languages, Java provides built-in support for multithreaded programming. A multithreaded program contains two or more parts that can run **concurrently**. Each part of such a program is called a thread and each thread defines a separate path of the execution. Thus, multithreading is a specialized form of multitasking.

The Java Thread Model

The Java run-time system depends on threads for many things. Threads reduce inefficiency by preventing the waste of CPU cycles.

Threads exist in several states:

- **New** - When we create an instance of Thread class, a thread is in a new state.
- **Running** - The Java thread is in running state.
- **Suspended** - A running thread can be **suspended**, which temporarily suspends its activity. A suspended thread can then be resumed, allowing it to pick up where it left off.
- **Blocked** - A Java thread can be blocked when waiting for a resource.
- **Terminated** - A thread can be terminated, which halts its execution immediately at any given time. Once a thread is terminated, it cannot be resumed.



Now let's jump to the most important topic of Java threads: thread class and runnable interface.

Multithreading in Java: Thread Class and Runnable Interface

Java's multithreading system is built upon the Thread class, its methods, and its companion interface, **Runnable**. To create a new thread, your program will either extend **Thread** or **implement** the **Runnable** interface.

The Thread class defines several methods that help manage threads:

Method	Meaning
getName	Obtain thread's name

getPriority	Obtain thread's priority
isAlive	Determine if a thread is still running
Join	Wait for a thread to terminate
Run	Entry point for the thread
Sleep	Suspend a thread for a period of time
Start	Start a thread by calling its run method

Now let's see how to use a Thread that begins with the **main java thread** that all Java programs have.

Main Java Thread

Here, I'll show you how to use Thread and Runnable interface to create and manage threads, beginning with the **main java thread**.

Why Is Main Thread So Important?

- Because it affects the other 'child' threads.
- Because it performs various shutdown actions.
- Because it's created automatically when your program is started.

How to Create a Java Thread

Java lets you create a thread one of two ways:

- By **implementing** the **Runnable** interface.
- By **extending** the **Thread**.

Let's look at how both ways help in implementing the Java thread.

Runnable Interface

The easiest way to create a thread is to create a class that implements the **Runnable** interface.

To implement Runnable interface, a class need only implement a single method called run(), which is declared like this:

```
public void run()
```

Inside run(), we will define the code that constitutes the new thread. Example:

```
public class MyClass implements Runnable {  
    public void run(){  
        System.out.println("MyClass running");  
    }  
}
```

To execute the run() method by a thread, pass an instance of MyClass to a Thread in its constructor (A **constructor in Java** is a block of code similar to a method that's called when an instance of an object is created). Here is how that is done:

```
Thread t1 = new Thread(new MyClass ());  
t1.start();
```

When the thread is started it will call the run() method of the MyClass instance instead of executing its own run() method. The above example would print out the text "**MyClass running**".

Extending Java Thread

The second way to create a thread is to create a new class that extends Thread, then override the run() method and then to create an instance of that class. The run() method is what is executed by the thread after you call start(). Here is an example of creating a Java Thread subclass:

```
public class MyClass extends Thread {  
    public void run(){  
        System.out.println("MyClass running");  
    }  
}
```

To create and start the above thread:

```
MyClass t1 = new MyClass ();  
T1.start();
```

When the run() method executes it will print out the text " **MyClass running**".

So far, we have been using only two threads: the **main** thread and one **child** thread. However, our program can affect as many threads as it needs. Let's see how we can create multiple threads.

Creating Multiple Threads

```
class MyThread implements Runnable {  
    String name;  
    Thread t;  
    MyThread(String thread){  
        name = threadname;  
        t = new Thread(this, name);  
        System.out.println("New thread: " + t);  
        t.start();  
    }  
}
```

```

public void run() {
    try {
        for(int i = 5; i > 0; i--) {
            System.out.println(name + ": " + i);
            Thread.sleep(1000);
        }
    } catch (InterruptedException e) {
        System.out.println(name + "Interrupted");
    }
    System.out.println(name + " exiting.");
}
}

class MultiThread {
    public static void main(String args[]) {
        new MyThread("One");
        new MyThread("Two");
        new NewThread("Three");
        try {
            Thread.sleep(10000);
        } catch (InterruptedException e) {
            System.out.println("Main thread Interrupted");
        }
        System.out.println("Main thread exiting.");
    }
}

```

Inter-thread communication in Java

Inter-thread communication or **Co-operation** is all about allowing synchronized threads to communicate with each other.

Cooperation (Inter-thread communication) is a mechanism in which a thread is paused running in its critical section and another thread is allowed to enter (or lock) in the same critical section to be executed. It is implemented by following methods of **Object class**:

- wait()
- notify()
- notifyAll()

Example of inter thread communication in java

Let's see the simple example of inter thread communication.

```

class Customer{
    int amount=10000;

    synchronized void withdraw(int amount){
        System.out.println("going to withdraw...");
    }
}

```

```

if(this.amount<amount){
System.out.println("Less balance; waiting for deposit...");
try{wait();}catch(Exception e){}
}
this.amount-=amount;
System.out.println("withdraw completed...");
}

```

```

synchronized void deposit(int amount){
System.out.println("going to deposit...");
this.amount+=amount;
System.out.println("deposit completed... ");
notify();
}
}

```

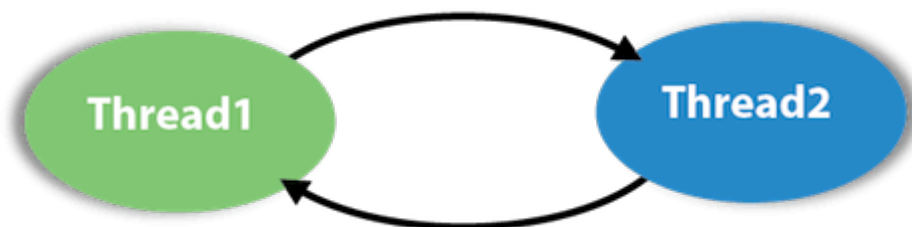
```

class Test{
public static void main(String args[]){
final Customer c=new Customer();
new Thread(){
public void run(){c.withdraw(15000);}
}.start();
new Thread(){
public void run(){c.deposit(10000);}
}.start();
}}

```

Deadlock in java

Deadlock in java is a part of multithreading. Deadlock can occur in a situation when a thread is waiting for an object lock, that is acquired by another thread and second thread is waiting for an object lock that is acquired by first thread. Since, both threads are waiting for each other to release the lock, the condition is called deadlock.



Example of Deadlock in java

```

public class TestDeadlockExample1 {
public static void main(String[] args) {
final String resource1 = "ratan jaiswal";
final String resource2 = "vimal jaiswal";

```

```

// t1 tries to lock resource1 then resource2
Thread t1 = new Thread() {
    public void run() {
        synchronized (resource1) {
            System.out.println("Thread 1: locked resource 1");

            try { Thread.sleep(100);} catch (Exception e) {}

            synchronized (resource2) {
                System.out.println("Thread 1: locked resource 2");
            }
        }
    }
};

// t2 tries to lock resource2 then resource1
Thread t2 = new Thread() {
    public void run() {
        synchronized (resource2) {
            System.out.println("Thread 2: locked resource 2");

            try { Thread.sleep(100);} catch (Exception e) {}

            synchronized (resource1) {
                System.out.println("Thread 2: locked resource 1");
            }
        }
    }
};

t1.start();
t2.start();
}
}

```


Java AWT Tutorial

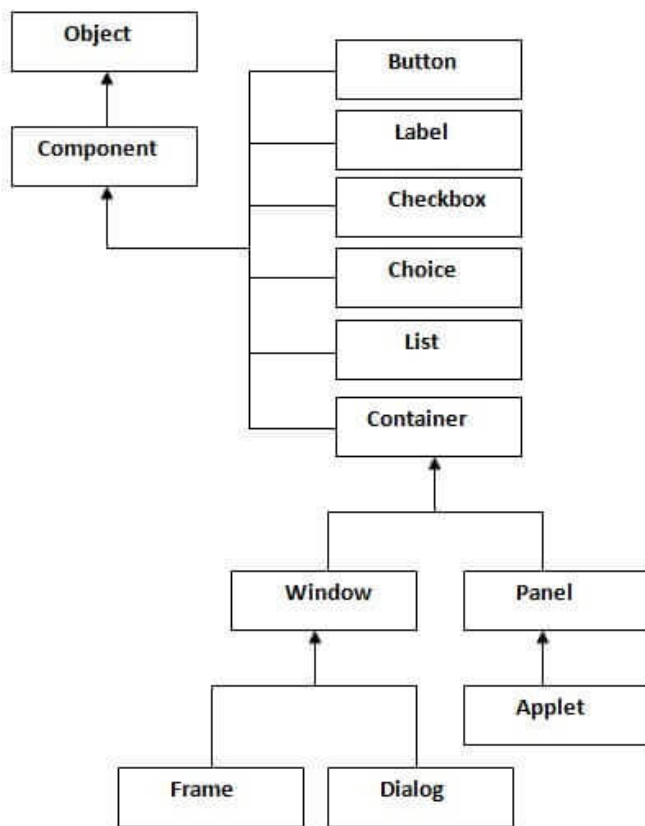
Java AWT (Abstract Window Toolkit) is *an API to develop GUI or window-based applications* in java.

Java AWT components are platform-dependent i.e. components are displayed according to the view of operating system. AWT is heavyweight i.e. its components are using the resources of OS.

The java.awt [package](#) provides [classes](#) for AWT api such as [TextField](#), [Label](#), [TextArea](#), [RadioButton](#), [CheckBox](#), [Choice](#), [List](#) etc.

Java AWT Hierarchy

The hierarchy of Java AWT classes are given below.



Container

The Container is a component in AWT that can contain another components like [buttons](#), textfields, labels etc. The classes that extends Container class are known as container such as Frame, Dialog and Panel.

Window

The window is the container that have no borders and menu bars. You must use frame, dialog or another window for creating a window.

Panel

The Panel is the container that doesn't contain title bar and menu bars. It can have other components like button, textfield etc.

Frame

The Frame is the container that contain title bar and can have menu bars. It can have other components like button, textfield etc.

Useful Methods of Component class

Method	Description
public void add(Component c)	inserts a component on this component.
public void setSize(int width,int height)	sets the size (width and height) of the component.
public void setLayout(LayoutManager m)	defines the layout manager for the component.
public void setVisible(boolean status)	changes the visibility of the component, by default false.

Java AWT Example

To create simple awt example, you need a frame. There are two ways to create a frame in AWT.

- By extending Frame class (inheritance)
- By creating the object of Frame class (association)

AWT Example by Inheritance

Let's see a simple example of AWT where we are inheriting Frame class. Here, we are showing Button component on the Frame.

```
import java.awt.*;
class First extends Frame{
    First(){
        Button b=new Button("click me");
        b.setBounds(30,100,80,30);// setting button position
        add(b);//adding button into frame
        setSize(300,300);//frame size 300 width and 300 height
        setLayout(null);//no layout manager
        setVisible(true);//now frame will be visible, by default not visible
    }
    public static void main(String args[]){
        First f=new First();
    }
}
```

The setBounds(int xaxis, int yaxis, int width, int height) method is used in the above example that sets the position of the awt button.

AWT Example by Association

Let's see a simple example of AWT where we are creating instance of Frame class. Here, we are showing Button component on the Frame.

```
import java.awt.*;
```

```

class First2{
First2(){
Frame f=new Frame();
Button b=new Button("click me");
b.setBounds(30,50,80,30);
f.add(b);
f.setSize(300,300);
f.setLayout(null);
f.setVisible(true);
}
public static void main(String args[]){
First2 f=new First2();
}}

```

Event and Listener (Java Event Handling)

Changing the state of an object is known as an event. For example, click on button, dragging mouse etc. The `java.awt.event` package provides many event classes and Listener interfaces for event handling.

Java Event classes and Listener interfaces

Event Classes	Listener Interfaces
ActionEvent	ActionListener
MouseEvent	MouseListener and MouseMotionListener
MouseWheelEvent	MouseWheelListener
KeyEvent	KeyListener
ItemEvent	ItemListener
TextEvent	TextListener
AdjustmentEvent	AdjustmentListener
WindowEvent	WindowListener
ComponentEvent	ComponentListener
ContainerEvent	ContainerListener
FocusEvent	FocusListener

Steps to perform Event Handling

Following steps are required to perform event handling:

- Register the component with the Listener
- Registration Methods
- For registering the component with the Listener, many classes provide the registration methods. For example:

Button

```
public void addActionListener(ActionListener a){}
```

MenuItem

```
public void addActionListener(ActionListener a){}
```

TextField

```
public void addActionListener(ActionListener a){}
```

```
public void addTextListener(TextListener a){}
```

TextArea

```
public void addTextListener(TextListener a){}
```

Checkbox

```
public void addItemListener(ItemListener a){}
```

Choice

```
public void addItemListener(ItemListener a){}
```

List

```
public void addActionListener(ActionListener a){}
```

```
public void addItemListener(ItemListener a){}
```

Java event handling by implementing ActionListener

```
import java.awt.*;
import java.awt.event.*;
class AEvent extends Frame implements ActionListener{
    TextField tf;
    AEvent(){

        //create components
        tf=new TextField();
        tf.setBounds(60,50,170,20);
        Button b=new Button("click me");
        b.setBounds(100,120,80,30);

        //register listener
        b.addActionListener(this);//passing current instance

        //add components and set size, layout and visibility
        add(b);add(tf);
        setSize(300,300);
        setLayout(null);
        setVisible(true);
    }
    public void actionPerformed(ActionEvent e){
```

```
tf.setText("Welcome");
}
public static void main(String args[]){
new AEvent();
}
}
```

public void setBounds(int xaxis, int yaxis, int width, int height); have been used in the above example that sets the position of the component it may be button, textfield etc.

Java AWT Button

The button class is used to create a labeled button that has platform independent implementation. The application result in some action when the button is pushed.

AWT Button Class declaration

public class Button **extends** Component **implements** Accessible
Example

```
import java.awt.*;
public class ButtonExample {
public static void main(String[] args) {
Frame f=new Frame("Button Example");
Button b=new Button("Click Here");
b.setBounds(50,100,80,30);
f.add(b);
f.setSize(400,400);
f.setLayout(null);
f.setVisible(true);
}
}
```

Java AWT Label

The object of Label class is a component for placing text in a container. It is used to display a single line of read only text. The text can be changed by an application but a user cannot edit it directly.

AWT Label Class Declaration

public class Label **extends** Component **implements** Accessible

Java Label Example

```
import java.awt.*;
class LabelExample{
public static void main(String args[]){
Frame f= new Frame("Label Example");
Label l1,l2;
l1=new Label("First Label.");
l1.setBounds(50,100, 100,30);
l2=new Label("Second Label.");
```



```

l2.setBounds(50,150, 100,30);
f.add(l1); f.add(l2);
f.setSize(400,400);
f.setLayout(null);
f.setVisible(true);
}
}

```

Java AWT TextField

The [object](#) of a TextField class is a text component that allows the editing of a single line text. It inherits TextComponent class.

AWT TextField Class Declaration

```
public class TextField extends TextComponent
```

Java AWT TextField Example

```

import java.awt.*;
class TextFieldExample{
public static void main(String args[]){
    Frame f= new Frame("TextField Example");
    TextField t1,t2;
    t1=new TextField("Welcome to Javatpoint.");
    t1.setBounds(50,100, 200,30);
    t2=new TextField("AWT Tutorial");
    t2.setBounds(50,150, 200,30);
    f.add(t1); f.add(t2);
    f.setSize(400,400);
    f.setLayout(null);
    f.setVisible(true);
}
}

```

Java AWT TextArea

The [object](#) of a TextArea class is a multi line region that displays text. It allows the editing of multiple line text. It inherits TextComponent class.

AWT TextArea Class Declaration

```
public class TextArea extends TextComponent
```

Java AWT TextArea Example

```

import java.awt.*;
public class TextAreaExample
{
    TextAreaExample(){
        Frame f= new Frame();
        TextArea area=new TextArea("Welcome to javatpoint");
    }
}

```

```

        area.setBounds(10,30, 300,300);
        f.add(area);
        f.setSize(400,400);
        f.setLayout(null);
        f.setVisible(true);
    }
    public static void main(String args[])
    {
        new TextAreaExample();
    }
}

```

Java AWT Checkbox

The Checkbox class is used to create a checkbox. It is used to turn an option on (true) or off (false). Clicking on a Checkbox changes its state from "on" to "off" or from "off" to "on".

AWT Checkbox Class Declaration

public class Checkbox **extends** Component **implements** ItemSelectable, Accessible

Java AWT Checkbox Example

```

import java.awt.*;
public class CheckboxExample
{
    CheckboxExample(){
        Frame f= new Frame("Checkbox Example");
        Checkbox checkbox1 = new Checkbox("C++");
        checkbox1.setBounds(100,100, 50,50);
        Checkbox checkbox2 = new Checkbox("Java", true);
        checkbox2.setBounds(100,150, 50,50);
        f.add(checkbox1);
        f.add(checkbox2);
        f.setSize(400,400);
        f.setLayout(null);
        f.setVisible(true);
    }
    public static void main(String args[])
    {
        new CheckboxExample();
    }
}

```

Java AWT CheckboxGroup

The object of CheckboxGroup class is used to group together a set of [Checkbox](#). At a time only one check box button is allowed to be in "on" state and remaining check box button in "off" state. It inherits the [object class](#).

Note: CheckboxGroup enables you to create radio buttons in AWT. There is no special control for creating radio buttons in AWT.

AWT CheckboxGroup Class Declaration

public class CheckboxGroup **extends** Object **implements** Serializable

Java AWT CheckboxGroup Example

```
import java.awt.*;
public class CheckboxGroupExample
{
    CheckboxGroupExample(){
        Frame f= new Frame("CheckboxGroup Example");
        CheckboxGroup cbg = new CheckboxGroup();
        Checkbox checkBox1 = new Checkbox("C++", cbg, false);
        checkBox1.setBounds(100,100, 50,50);
        Checkbox checkBox2 = new Checkbox("Java", cbg, true);
        checkBox2.setBounds(100,150, 50,50);
        f.add(checkBox1);
        f.add(checkBox2);
        f.setSize(400,400);
        f.setLayout(null);
        f.setVisible(true);
    }
    public static void main(String args[])
    {
        new CheckboxGroupExample();
    }
}
```

Java AWT Choice

The object of Choice class is used to show [popup menu](#) of choices. Choice selected by user is shown on the top of a menu. It inherits Component class.

AWT Choice Class Declaration

public class Choice **extends** Component **implements** ItemSelectable, Accessible

Java AWT Choice Example

```
import java.awt.*;
public class ChoiceExample
{
    ChoiceExample(){
        Frame f= new Frame();
        Choice c=new Choice();
        c.setBounds(100,100, 75,75);
    }
}
```

```

        c.add("Item 1");
        c.add("Item 2");
        c.add("Item 3");
        c.add("Item 4");
        c.add("Item 5");
        f.add(c);
        f.setSize(400,400);
        f.setLayout(null);
        f.setVisible(true);
    }
    public static void main(String args[])
    {
        new ChoiceExample();
    }
}

```

Java AWT List

The object of List class represents a list of text items. By the help of list, user can choose either one item or multiple items. It inherits Component class.

AWT List class Declaration

public class List **extends** Component **implements** ItemSelectable, Accessible

Java AWT List Example

```

import java.awt.*;
public class ListExample
{
    ListExample(){
        Frame f= new Frame();
        List l1=new List(5);
        l1.setBounds(100,100, 75,75);
        l1.add("Item 1");
        l1.add("Item 2");
        l1.add("Item 3");
        l1.add("Item 4");
        l1.add("Item 5");
        f.add(l1);
        f.setSize(400,400);
        f.setLayout(null);
        f.setVisible(true);
    }
    public static void main(String args[])
    {
        new ListExample();
    }
}

```

Java Layout Managers

Java BorderLayout

The BorderLayout is used to arrange the components in five regions: north, south, east, west and center. Each region (area) may contain one component only. It is the default layout of frame or window. The BorderLayout provides five constants for each region:

1. **public static final int NORTH**
2. **public static final int SOUTH**
3. **public static final int EAST**
4. **public static final int WEST**
5. **public static final int CENTER**

Constructors of BorderLayout class:

- **BorderLayout():** creates a border layout but with no gaps between the components.
- **BorderLayout(int hgap, int vgap):** creates a border layout with the given horizontal and vertical gaps between the components.

```
import java.awt.*;

import javax.swing.*;

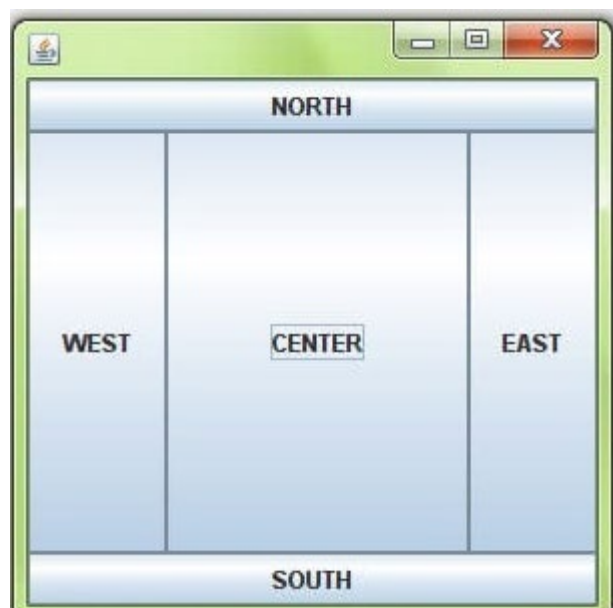
public class Border {
    JFrame f;
    Border() {
        f = new JFrame();

        JButton b1 = new JButton("NORTH");
        JButton b2 = new JButton("SOUTH");
        JButton b3 = new JButton("EAST");
        JButton b4 = new JButton("WEST");
        JButton b5 = new JButton("CENTER");

        f.add(b1, BorderLayout.NORTH);
        f.add(b2, BorderLayout.SOUTH);
        f.add(b3, BorderLayout.EAST);
        f.add(b4, BorderLayout.WEST);
        f.add(b5, BorderLayout.CENTER);

        f.setSize(300, 300);
        f.setVisible(true);
    }

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



Java GridLayout

The GridLayout is used to arrange the components in rectangular grid. One component is displayed in each rectangle.

Constructors of GridLayout class

1. **GridLayout():** creates a grid layout with one column per component in a row.
2. **GridLayout(int rows, int columns):** creates a grid layout with the given rows and columns but no gaps between the components.
3. **GridLayout(int rows, int columns, int hgap, int vgap):** creates a grid layout with the given rows and columns along with given horizontal and vertical gaps.

Example of GridLayout class

```
import java.awt.*;  
import javax.swing.*;  
public class MyGridLayout {  
    JFrame f;  
    MyGridLayout() {  
        f=new JFrame();  
        JButton b1=new JButton("1");  
        JButton b2=new JButton("2");  
        JButton b3=new JButton("3");  
        JButton b4=new JButton("4");  
        JButton b5=new JButton("5");  
        JButton b6=new JButton("6");  
        JButton b7=new JButton("7");  
        JButton b8=new JButton("8");  
        JButton b9=new JButton("9");  
        f.add(b1);f.add(b2);f.add(b3);f.add(b4);f.add(b5);  
        f.add(b6);f.add(b7);f.add(b8);f.add(b9);  
        f.setLayout(new GridLayout(3,3));  
        //setting grid layout of 3 rows and 3 columns  
        f.setSize(300,300);  
        f.setVisible(true);  
    }  
    public static void main(String[] args) {  
        new MyGridLayout();  
    }  
}
```



Java FlowLayout

The FlowLayout is used to arrange the components in a line, one after another (in a flow). It is the default layout of applet or panel.

Fields of FlowLayout class

1. **public static final int LEFT**
2. **public static final int RIGHT**
3. **public static final int CENTER**
4. **public static final int LEADING**
5. **public static final int TRAILING**

Constructors of FlowLayout class

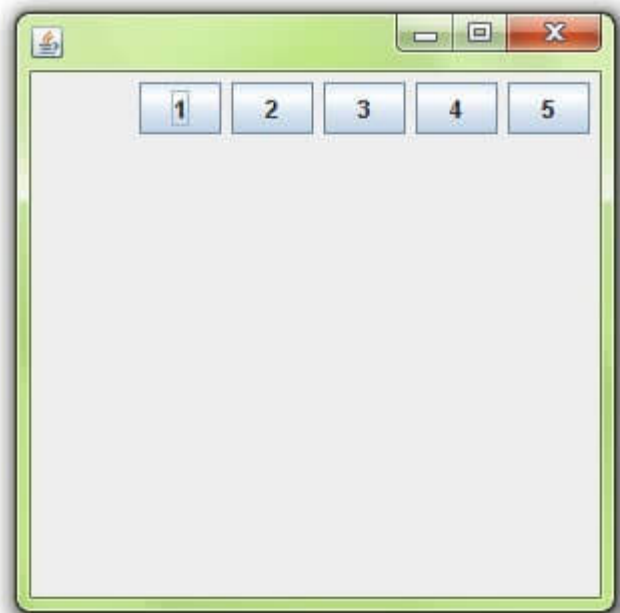
1. **FlowLayout():** creates a flow layout with centered alignment and a default 5 unit horizontal and vertical gap.
2. **FlowLayout(int align):** creates a flow layout with the given alignment and a default 5 unit horizontal and vertical gap.
3. **FlowLayout(int align, int hgap, int vgap):** creates a flow layout with the given alignment and the given horizontal and vertical gap.

Example of FlowLayout class

```
import java.awt.*;
import javax.swing.*;

public class MyFlowLayout {
    JFrame f;
    MyFlowLayout() {
        f=new JFrame();
        JButton b1=new JButton("1");
        JButton b2=new JButton("2");
        JButton b3=new JButton("3");
        JButton b4=new JButton("4");
        JButton b5=new JButton("5");

        f.add(b1);f.add(b2);f.add(b3);f.add(b4);f.add(b5);
        f.setLayout(new FlowLayout(FlowLayout.RIGHT));
        //setting flow layout of right alignment
        f.setSize(300,300);
        f.setVisible(true);
    }
    public static void main(String[] args) {
        new MyFlowLayout();
    } }
```



Java BorderLayout

The BorderLayout is used to arrange the components either vertically or horizontally. For this purpose, BorderLayout provides four constants. They are as follows:

Note: BorderLayout class is found in javax.swing package.

Fields of BorderLayout class

1. **public static final int X_AXIS**
2. **public static final int Y_AXIS**
3. **public static final int LINE_AXIS**
4. **public static final int PAGE_AXIS**

Constructor of BorderLayout class

1. **BorderLayout(Container c, int axis):** creates a box layout that arranges the components with the given axis.

Example of BorderLayout class with Y-AXIS:

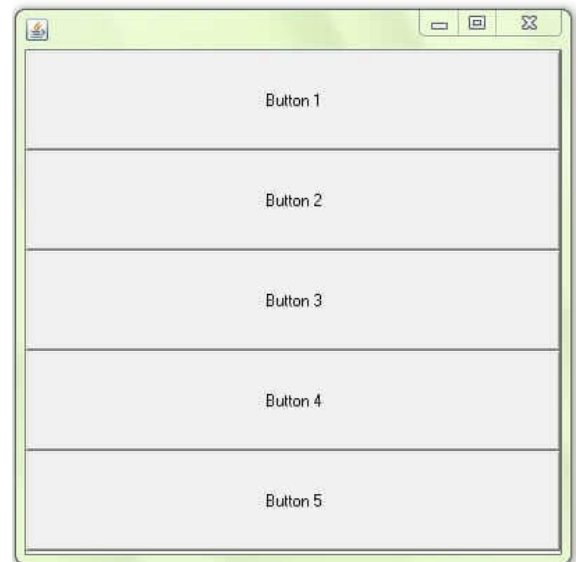
```
import java.awt.*;
import javax.swing.*;

public class BorderLayoutExample1 extends Frame {
    Button buttons[];

    public BorderLayoutExample1 () {
        buttons = new Button [5];

        for (int i = 0; i < 5; i++) {
            buttons[i] = new Button ("Button " + (i + 1));
            add (buttons[i]);
        }
        setLayout (new BorderLayout (this, BorderLayout.Y_AXIS));
        setSize(400,400);
        setVisible(true);
    }

    public static void main(String args[]){
        BorderLayoutExample1 b=new BorderLayoutExample1();
    }
}
```



Java CardLayout

The CardLayout class manages the components in such a manner that only one component is visible at a time. It treats each component as a card that is why it is known as CardLayout.

Constructors of CardLayout class

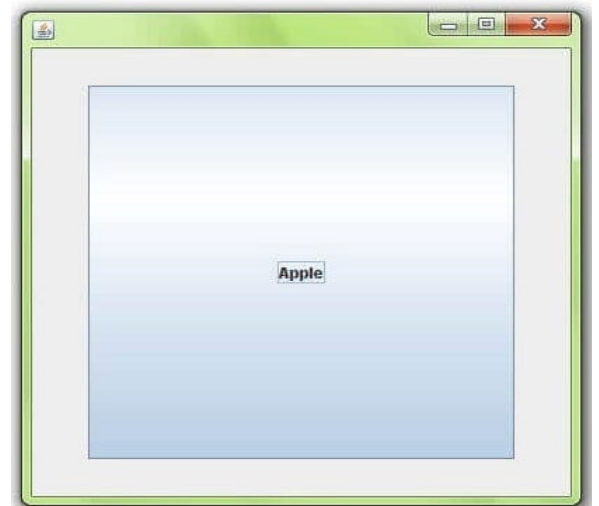
1. **CardLayout():** creates a card layout with zero horizontal and vertical gap.
2. **CardLayout(int hgap, int vgap):** creates a card layout with the given horizontal and vertical gap.

Commonly used methods of CardLayout class

- **public void next(Container parent):** is used to flip to the next card of the given container.
- **public void previous(Container parent):** is used to flip to the previous card of the given container.
- **public void first(Container parent):** is used to flip to the first card of the given container.
- **public void last(Container parent):** is used to flip to the last card of the given container.
- **public void show(Container parent, String name):** is used to flip to the specified card with the given name.

Example of CardLayout class

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
public class CardLayoutExample extends JFrame implements ActionListener{
    CardLayout card;
    JButton b1,b2,b3;
    Container c;
    CardLayoutExample(){
        c=getContentPane();
        card=new CardLayout(40,30);
        //create CardLayout object with 40 hor space and 30 ver space
        c.setLayout(card);
        b1=new JButton("Apple");
        b2=new JButton("Boy");
        b3=new JButton("Cat");
        b1.addActionListener(this);
        b2.addActionListener(this);
        b3.addActionListener(this);
        c.add("a",b1);c.add("b",b2);c.add("c",b3);
    }
    public void actionPerformed(ActionEvent e) {
        card.next(c);
    }
    public static void main(String[] args) {
        CardLayoutExample cl=new CardLayoutExample();
        cl.setSize(400,400);
        cl.setVisible(true);
        cl.setDefaultCloseOperation(EXIT_ON_CLOSE);
    } }
```



Java GridBagLayout

The Java GridBagLayout class is used to align components vertically, horizontally or along their baseline.

The components may not be of same size. Each GridBagLayout object maintains a dynamic, rectangular grid of cells. Each component occupies one or more cells known as its display area. Each component associates an instance of GridBagConstraints. With the help of constraints object we arrange component's display area on the grid. The GridBagLayout manages each component's minimum and preferred sizes in order to determine component's size.

Fields

Modifier and Type	Field	Description
double[]	columnWeights	It is used to hold the overrides to the column weights.
int[]	columnWidths	It is used to hold the overrides to the column minimum width.
protected Hashtable <Component,GridBagConstraints>	comptable	It is used to maintains the association between a component and its gridbag constraints.
protected GridBagConstraints	defaultConstraints	It is used to hold a gridbag constraints instance containing the default values.
protected GridBagLayoutInfo	layoutInfo	It is used to hold the layout information for the gridbag.
protected static int	MAXGRIDSIZE	No longer in use just for backward compatibility
protected static int	MINSIZE	It is smallest grid that can be laid out by the grid bag layout.
protected static int	PREFERRED_SIZE	It is preferred grid size that can be laid out by the grid bag layout.
int[]	rowHeights	It is used to hold the overrides to the row minimum heights.
double[]	rowWeights	It is used to hold the overrides to the row weights.

Useful Methods

Modifier and Type	Method	Description
void	addLayoutComponent(Component comp, Object constraints)	It adds specified component to the layout, using the specified constraints object.
void	addLayoutComponent(String name, Component comp)	It has no effect, since this layout manager does not use a per-component string.
protected void	adjustForGravity(GridBagConstraints constraints, Rectangle r)	It adjusts the x, y, width, and height fields to the correct values depending on the constraint geometry and pads.
protected void	AdjustForGravity(GridBagConstraints constraints, Rectangle r)	This method is for backwards compatibility only
protected void	arrangeGrid(Container parent)	Lays out the grid.
protected void	ArrangeGrid(Container parent)	This method is obsolete and supplied for backwards compatibility
GridBagConstraints	getConstraints(Component comp)	It is for getting the constraints for the specified component.
float	getLayoutAlignmentX(Container parent)	It returns the alignment along the x axis.
float	getLayoutAlignmentY(Container parent)	It returns the alignment along the y axis.
int[][]	getLayoutDimensions()	It determines column widths and row heights for the layout grid.
protected GridBagLayoutInfo	getLayoutInfo(Container parent, int sizeflag)	This method is obsolete and supplied for backwards compatibility.
protected GridBagLayoutInfo	GetLayoutInfo(Container parent, int sizeflag)	This method is obsolete and supplied for backwards compatibility.
Point	getLayoutOrigin()	It determines the origin of the layout area, in the graphics coordinate space of the target container.
double[][]	getLayoutWeights()	It determines the weights of the layout grid's columns and rows.
protected Dimension	getMinSize(Container parent, GridBagLayoutInfo info)	It figures out the minimum size of the master based on the information from getLayoutInfo.
protected Dimension	GetMinSize(Container parent, GridBagLayoutInfo info)	This method is obsolete and supplied for backwards compatibility only

Example

```
import java.awt.Button;
import java.awt.GridBagConstraints;
import java.awt.GridBagLayout;
import javax.swing.*;

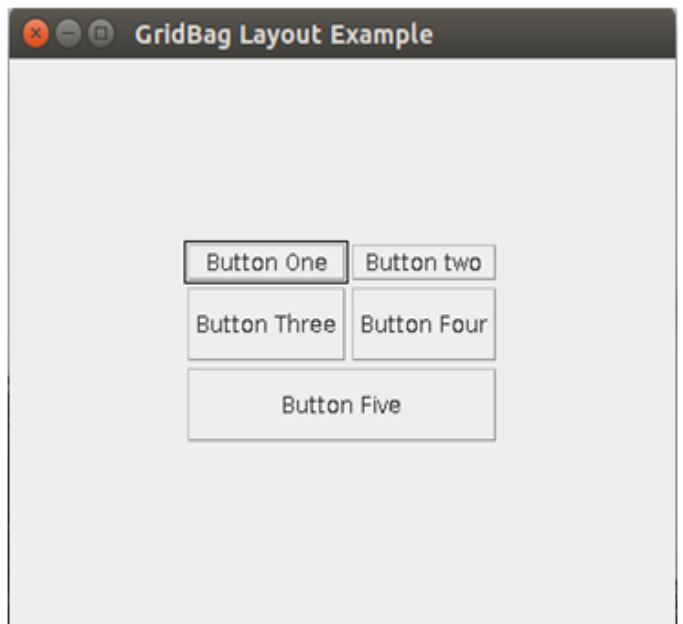
public class GridBagLayoutExample extends JFrame{
    public static void main(String[] args) {
        GridBagLayoutExample a = new GridBagLayoutExample();
    }

    public GridBagLayoutExample() {
        GridBagLayoutgrid = new GridBagLayout();
        GridBagConstraints gbc = new GridBagConstraints();
```

```

        setLayout(grid);
        setTitle("GridBag Layout Example");
        GridBagLayout layout = new GridBagLayout();
        this.setLayout(layout);
        gbc.fill = GridBagConstraints.HORIZONTAL;
        gbc.gridx = 0;
        gbc.gridy = 0;
        this.add(new Button("Button One"), gbc);
        gbc.gridx = 1;
        gbc.gridy = 0;
        this.add(new Button("Button two"), gbc);
        gbc.fill = GridBagConstraints.HORIZONTAL;
        gbc.ipady = 20;
        gbc.gridx = 0;
        gbc.gridy = 1;
        this.add(new Button("Button Three"), gbc);
        gbc.gridx = 1;
        gbc.gridy = 1;
        this.add(new Button("Button Four"), gbc);
        gbc.gridx = 0;
        gbc.gridy = 2;
        gbc.fill = GridBagConstraints.HORIZONTAL;
        gbc.gridwidth = 2;
        this.add(new Button("Button Five"), gbc);
        setSize(300, 300);
        setPreferredSize(getSize());
        setVisible(true);
        setDefaultCloseOperation(EXIT_ON_CLOSE);
    }
}

```



GroupLayout

GroupLayout groups its components and places them in a Container hierarchically. The grouping is done by instances of the Group class.

Group is an abstract class and two concrete classes which implement this Group class are SequentialGroup and ParallelGroup. SequentialGroup positions its child sequentially one after another where as ParallelGroup aligns its child on top of each other. The GroupLayout class provides methods such as createParallelGroup() and createSequentialGroup() to create groups.

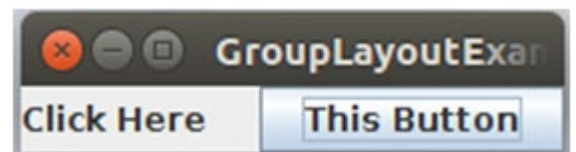
GroupLayout treats each axis independently. That is, there is a group representing the horizontal axis, and a group representing the vertical axis. Each component must exist in both a horizontal and vertical group, otherwise an IllegalStateException is thrown during layout, or when the minimum, preferred or maximum size is requested.

Useful Methods

Modifier and Type	Field	Description
void	addLayoutComponent(Component component, Object constraints)	It notify that a Component has been added to the parent container.
void	addLayoutComponent(String name, Component component)	It notify that a Component has been added to the parent container.
GroupLayout.ParallelGroup	createBaselineGroup(boolean resizable, boolean anchorBaselineToTop)	It creates and returns a ParallelGroup that aligns it's elements along the baseline.
GroupLayout.ParallelGroup	createParallelGroup()	It creates and returns a ParallelGroup with an alignment of Alignment.LEADING
GroupLayout.ParallelGroup	createParallelGroup(GroupLayout.Alignment alignment)	It creates and returns a ParallelGroup with the specified alignment.
GroupLayout.ParallelGroup	createParallelGroup(GroupLayout.Alignment alignment, boolean resizable)	It creates and returns a ParallelGroup with the specified alignment and resize behavior.
GroupLayout.SequentialGroup	createSequentialGroup()	It creates and returns a SequentialGroup.
boolean	getAutoCreateContainerGaps()	It returns true if gaps between the container and components that border the container are automatically created.
boolean	getAutoCreateGaps()	It returns true if gaps between components are automatically created.
boolean	getHonorsVisibility()	It returns whether component visibility is considered when sizing and positioning components.
float	getLayoutAlignmentX(Container parent)	It returns the alignment along the x axis.
float	getLayoutAlignmentY(Container parent)	It returns the alignment along the y axis.
Dimension	maximumLayoutSize(Container parent)	It returns the maximum size for the specified container.

Example

```
public class GroupExample {  
    public static void main(String[] args) {  
        JFrame frame = new JFrame("GroupLayoutExample");  
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);  
        Container contentPanel = frame.getContentPane();  
        GroupLayout groupLayout = new GroupLayout(contentPanel);  
        contentPanel.setLayout(groupLayout);  
        JLabel clickMe = new JLabel("Click Here");  
        JButton button = new JButton("This Button");  
        groupLayout.setHorizontalGroup(  
            groupLayout.createSequentialGroup()  
                .addComponent(clickMe)  
                .addGap(10, 20, 100)  
                .addComponent(button));  
        groupLayout.setVerticalGroup(  
            groupLayout.createParallelGroup(GroupLayout.Alignment.BASELINE)  
                .addComponent(clickMe)  
                .addComponent(button));  
        frame.pack();  
        frame.setVisible(true);  
    }  
}
```



Java SpringLayout

A **SpringLayout** arranges the children of its associated container according to a set of constraints. Constraints are nothing but horizontal and vertical distance between two component edges. Every constraints are represented by a SpringLayout.Constraint object.

Each child of a SpringLayout container, as well as the container itself, has exactly one set of constraints associated with them.

Each edge position is dependent on the position of the other edge. If a constraint is added to create new edge than the previous binding is discarded. SpringLayout doesn't automatically set the location of the components it manages.

Useful Methods

Modifier and Type	Method	Description
void	addLayoutComponent(Component component, Object constraints)	If constraints is an instance of SpringLayout.Constraint, associates the constraints with the specified component.
void	addLayoutComponent(String name, Component c)	Has no effect, since this layout manager does not use a per-component string.
Spring	getConstraint(String edgeName, Component c)	It returns the spring controlling the distance between the specified edge of the component and the top or left edge of its parent.
SpringLayout.Constraint	getConstraints(Component c)	It returns the constraints for the specified component.
float	getLayoutAlignmentX(Container p)	It returns 0.5f (centered).
float	getLayoutAlignmentY(Container p)	It returns 0.5f (centered).
void	invalidateLayout(Container p)	It Invalidates the layout, indicating that if the layout manager has cached information it should be discarded.
void	layoutContainer(Container parent)	It lays out the specified container.
Dimension	maximumLayoutSize(Container parent)	It is used to calculate the maximum size dimensions for the specified container, given the components it contains.
Dimension	minimumLayoutSize(Container parent)	It is used to calculate the minimum size dimensions for the specified container, given the components it contains.
Dimension	preferredLayoutSize(Container parent)	It is used to calculate the preferred size dimensions for the specified container, given the components it contains.

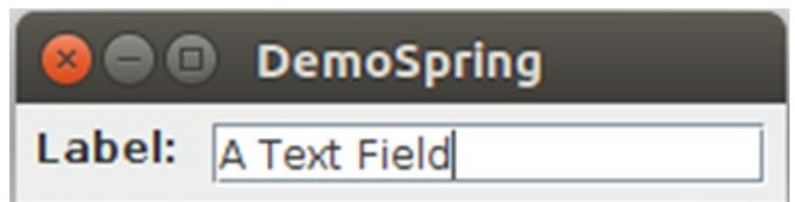
Example

```
import java.awt.Container;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JTextField;
import javax.swing.SpringLayout;
public class MySpringDemo {
    private static void createAndShowGUI() {
        JFrame frame = new JFrame("MySpringDemp");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        Container contentPane = frame.getContentPane();
        SpringLayout layout = new SpringLayout();
        contentPane.setLayout(layout);

        JLabel label = new JLabel("Label: ");
        JTextField textField = new JTextField("My Text Field", 15);
        contentPane.add(label);
        contentPane.add(textField);

        layout.putConstraint(SpringLayout.WEST, label, 6, SpringLayout.WEST, contentPane);
        layout.putConstraint(SpringLayout.NORTH, label, 6, SpringLayout.NORTH, contentPane);
        layout.putConstraint(SpringLayout.WEST, textField, 6, SpringLayout.EAST, label);
    }
}
```



```
layout.putConstraint(SpringLayout.NORTH, textField,6,SpringLayout.NORTH, contentPane);
layout.putConstraint(SpringLayout.EAST, contentPane,6,SpringLayout.EAST, textField);
layout.putConstraint(SpringLayout.SOUTH, contentPane,6,SpringLayout.SOUTH, textField);

frame.pack();
frame.setVisible(true);
}
public static void main(String[] args) {
    javax.swing.SwingUtilities.invokeLater(new Runnable() {
        public void run() {
            createAndShowGUI();
        }
    });
}
```

Java Swing

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.

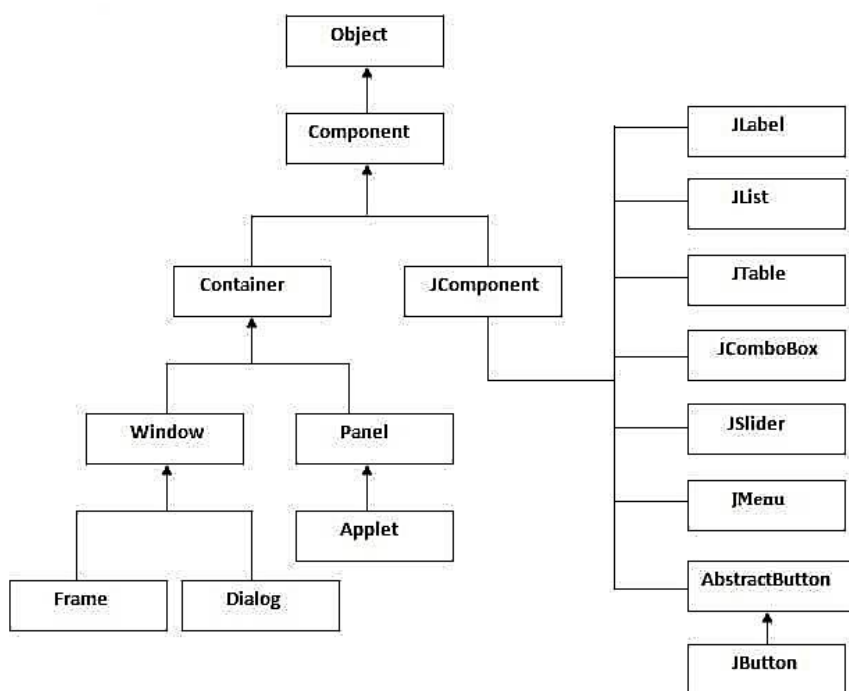
No.	Java AWT	Java Swing
1)	AWT components are platform-dependent .	Java swing components are platform-independent .
2)	AWT components are heavyweight .	Swing components are lightweight .
3)	AWT doesn't support pluggable look and feel .	Swing supports pluggable look and feel .
4)	AWT provides less components than Swing.	Swing provides more powerful components such as tables, lists, scrollpanes, colorchooser, tabbedpane etc.
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
public void add(Component c)	add a component on another component.
public void setSize(int width,int height)	sets size of the component.
public void setLayout(LayoutManager m)	sets the layout manager for the component.
public void setVisible(boolean b)	sets the visibility of the component. It is by default false.

Java 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.

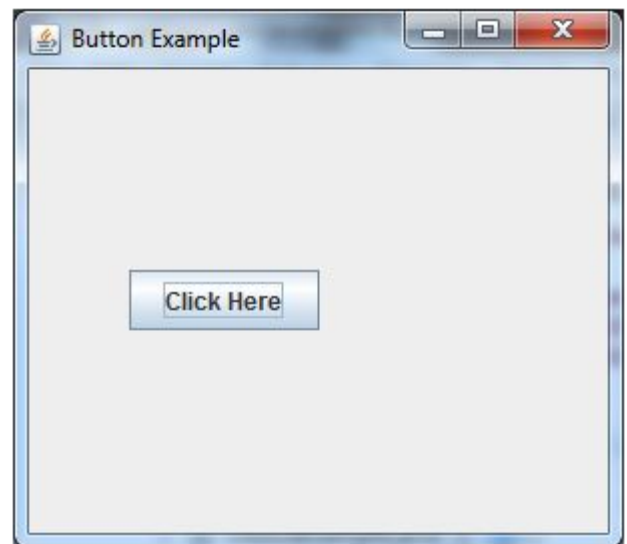
public class JButton **extends** AbstractButton **implements** Accessible

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 addActionListener(ActionListener a)	It is used to add the action listener to this object.

Java JButton Example

```
import javax.swing.*;
public class ButtonExample {
public static void main(String[] args) {
    JFrame f=new JFrame("Button Example");
    JButton b=new JButton("Click Here");
    b.setBounds(50,100,95,30);
    f.add(b);
    f.setSize(400,400);
    f.setLayout(null);
    f.setVisible(true);
}
}
```



Java JButton Example with ActionListener

```
import java.awt.event.*;
import javax.swing.*;
public class ButtonExample {
public static void main(String[] args) {
    JFrame f=new JFrame("Button Example");
    final JTextField tf=new JTextField();
    tf.setBounds(50,50, 150,20);
```



```

        JButton b=new JButton("Click Here");
        b.setBounds(50,100,95,30);
        b.addActionListener(new ActionListener(){
public void actionPerformed(ActionEvent e){
            tf.setText("Welcome to Javatpoint.");
        }
    });
    f.add(b);f.add(tf);
    f.setSize(400,400);
    f.setLayout(null);
    f.setVisible(true);
}
}

```

Example of displaying image on the button:

```

import javax.swing.*;
public class ButtonExample{
    ButtonExample(){
        JFrame f=new JFrame("Button Example");
        JButton b=new JButton(new ImageIcon("D:\\icon.png"));
        b.setBounds(100,100,100, 40);
        f.add(b);
        f.setSize(300,400);
        f.setLayout(null);
        f.setVisible(true);
        f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }
    public static void main(String[] args) {
        new ButtonExample();
    }
}

```

Java JLabel

The object of JLabel class is a component for placing text in a container. It is used to display a single line of read only text. The text can be changed by an application but a user cannot edit it directly. It inherits JComponent class.

JLabel class declaration

Let's see the declaration for javax.swing.JLabel class.

public class JLabel **extends** JComponent **implements** SwingConstants, Accessible

Commonly used Methods:

Methods	Description
String getText()	t returns the text string that a label displays.
void setText(String text)	It defines the single line of text this component will display.
void setHorizontalAlignment(int alignment)	It sets the alignment of the label's contents along the X axis.
Icon getIcon()	It returns the graphic image that the label displays.
int getHorizontalAlignment()	It returns the alignment of the label's contents along the X axis.

Java JLabel Example

```

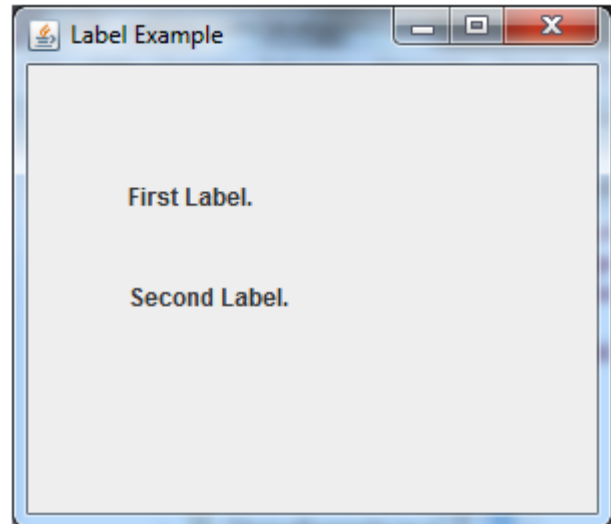
import javax.swing.*;
class LabelExample

```

```

{
public static void main(String args[])
{
    JFrame f= new JFrame("Label Example");
    JLabel l1,l2;
    l1=new JLabel("First Label.");
    l1.setBounds(50,50, 100,30);
    l2=new JLabel("Second Label.");
    l2.setBounds(50,100, 100,30);
    f.add(l1); f.add(l2);
    f.setSize(300,300);
    f.setLayout(null);
    f.setVisible(true);
}
}

```



Java JLabel Example with ActionListener

```

import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
public class LabelExample extends Frame implements ActionListener{
    JTextField tf; JLabel l; JButton b;
    LabelExample(){
        tf=new JTextField();
        tf.setBounds(50,50, 150,20);
        l=new JLabel();
        l.setBounds(50,100, 250,20);
        b=new JButton("Find IP");
        b.setBounds(50,150,95,30);
        b.addActionListener(this);
        add(b);add(tf);add(l);
        setSize(400,400);
        setLayout(null);
        setVisible(true);
    }
    public void actionPerformed(ActionEvent e) {
        try{
            String host=tf.getText();
            String ip=java.net.InetAddress.getByName(host).getHostAddress();
            l.setText("IP of "+host+" is: "+ip);
        }catch(Exception ex){System.out.println(ex);}
    }
    public static void main(String[] args) {
        new LabelExample();
    }
}

```

Java JTextField

The object of a JTextField class is a text component that allows the editing of a single line text. It inherits JTextComponent class.

JTextField class declaration

Let's see the declaration for javax.swing.JTextField class.

```

public class JTextField extends JTextComponent implements SwingConstants

```

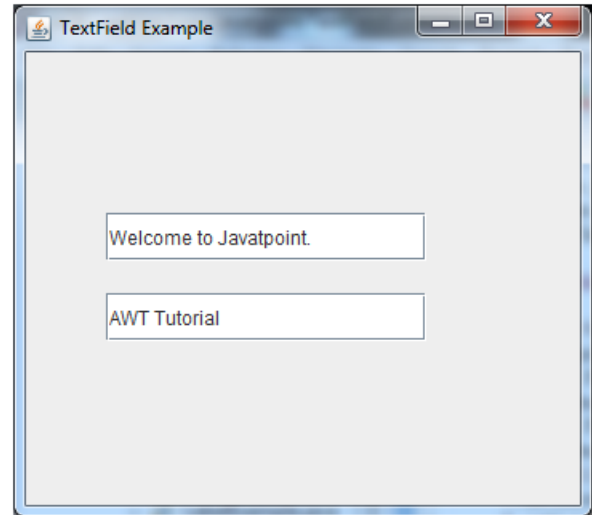
Commonly used Methods:

Methods	Description
---------	-------------

<code>void addActionListener(ActionListener l)</code>	It is used to add the specified action listener to receive action events from this textfield.
<code>Action getAction()</code>	It returns the currently set Action for this ActionEvent source, or null if no Action is set.
<code>void setFont(Font f)</code>	It is used to set the current font.
<code>void removeActionListener(ActionListener l)</code>	It is used to remove the specified action listener so that it no longer receives action events from this textfield.

Java JTextField Example

```
import javax.swing.*;
class TextFieldExample
{
public static void main(String args[])
{
JFrame f= new JFrame("TextField Example");
JTextField t1,t2;
t1=new JTextField("Welcome to Javatpoint.");
t1.setBounds(50,100, 200,30);
t2=new JTextField("AWT Tutorial");
t2.setBounds(50,150, 200,30);
f.add(t1); f.add(t2);
f.setSize(400,400);
f.setLayout(null);
f.setVisible(true);
}
}
```



Java JTextField Example with ActionListener

```
import javax.swing.*;
import java.awt.event.*;
public class TextFieldExample implements ActionListener{
JTextField tf1,tf2,tf3;
JButton b1,b2;
TextFieldExample(){
JFrame f= new JFrame();
tf1=new JTextField();
tf1.setBounds(50,50,150,20);
tf2=new JTextField();
tf2.setBounds(50,100,150,20);
tf3=new JTextField();
tf3.setBounds(50,150,150,20);
tf3.setEditable(false);
b1=new JButton("+");
b1.setBounds(50,200,50,50);
b2=new JButton("-");
b2.setBounds(120,200,50,50);
b1.addActionListener(this);
b2.addActionListener(this);
f.add(tf1);f.add(tf2);f.add(tf3);f.add(b1);f.add(b2);
f.setSize(300,300);
f.setLayout(null);
f.setVisible(true);
}
public void actionPerformed(ActionEvent e) {
```

```

String s1=tf1.getText();
String s2=tf2.getText();
int a=Integer.parseInt(s1);
int b=Integer.parseInt(s2);
int c=0;
if(e.getSource()==b1){
    c=a+b;
}else if(e.getSource()==b2){
    c=a-b;
}
String result=String.valueOf(c);
tf3.setText(result);
}
public static void main(String[] args) {
    new TextFieldExample();
} }

```

Java JTextArea

The object of a JTextArea class is a multi line region that displays text. It allows the editing of multiple line text. It inherits JTextComponent class

JTextArea class declaration

Let's see the declaration for javax.swing.JTextArea class.

```
public class JTextArea extends JTextComponent
```

Commonly used Methods:

Methods	Description
void setRows(int rows)	It is used to set specified number of rows.
void setColumns(int cols)	It is used to set specified number of columns.
void setFont(Font f)	It is used to set the specified font.
void insert(String s, int position)	It is used to insert the specified text on the specified position.
void append(String s)	It is used to append the given text to the end of the document.

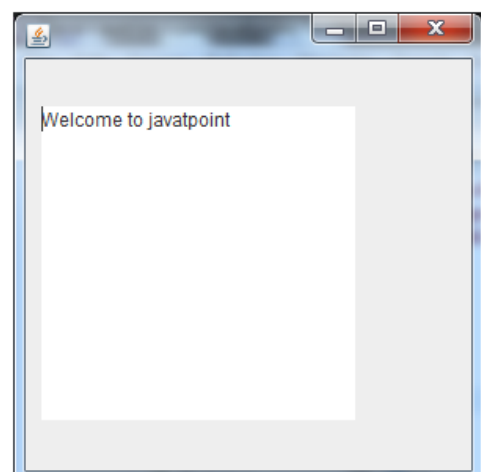
Java JTextArea Example

```

import javax.swing.*;
public class TextAreaExample
{
    TextAreaExample(){
        JFrame f=new JFrame();
        JTextArea area=new JTextArea("Welcome to javatpoint");
        area.setBounds(10,30, 200,200);
        f.add(area);
        f.setSize(300,300);
        f.setLayout(null);
        f.setVisible(true);
    }
    public static void main(String args[])
    {
        new TextAreaExample();
    }
}

```

Output:



Java JTextArea Example with ActionListener

```
import javax.swing.*;
import java.awt.event.*;
public class TextAreaExample implements ActionListener{
    JLabel l1,l2;
    JTextArea area;
    JButton b;
    TextAreaExample() {
        JFrame f= new JFrame();
        l1=new JLabel();
        l1.setBounds(50,25,100,30);
        l2=new JLabel();
        l2.setBounds(160,25,100,30);
        area=new JTextArea();
        area.setBounds(20,75,250,200);
        b=new JButton("Count Words");
        b.setBounds(100,300,120,30);
        b.addActionListener(this);
        f.add(l1);f.add(l2);f.add(area);f.add(b);
        f.setSize(450,450);
        f.setLayout(null);
        f.setVisible(true);
    }
    public void actionPerformed(ActionEvent e){
        String text=area.getText();
        String words[]=text.split("\\s");
        l1.setText("Words: "+words.length);
        l2.setText("Characters: "+text.length());
    }
    public static void main(String[] args) {
        new TextAreaExample();
    }
}
```

Java JPasswordField

The object of a JPasswordField class is a text component specialized for password entry. It allows the editing of a single line of text. It inherits JTextField class.

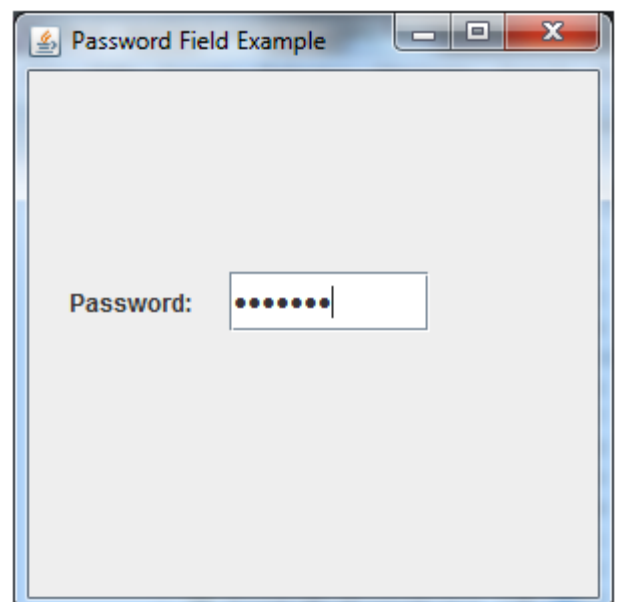
JPasswordField class declaration

Let's see the declaration for javax.swing.JPasswordField class.

```
public class JPasswordField extends JTextField
```

Java JPasswordField Example

```
import javax.swing.*;
public class PasswordFieldExample {
    public static void main(String[] args) {
        JFrame f=new JFrame("Password Field Example");
        JPasswordField value = new JPasswordField();
        JLabel l1=new JLabel("Password:");
        l1.setBounds(20,100, 80,30);
        value.setBounds(100,100,100,30);
```



```

        f.add(value); f.add(l1);
        f.setSize(300,300);
        f.setLayout(null);
        f.setVisible(true);
    }
}

```

Java JPasswordField Example with ActionListener

```

import javax.swing.*;
import java.awt.event.*;
public class PasswordFieldExample {
    public static void main(String[] args) {
        JFrame f=new JFrame("Password Field Example");
        final JLabel label = new JLabel();
        label.setBounds(20,150, 200,50);
        final JPasswordField value = new JPasswordField();
        value.setBounds(100,75,100,30);
        JLabel l1=new JLabel("Username:");
        l1.setBounds(20,20, 80,30);
        JLabel l2=new JLabel("Password:");
        l2.setBounds(20,75, 80,30);
        JButton b = new JButton("Login");
        b.setBounds(100,120, 80,30);
        final JTextField text = new JTextField();
        text.setBounds(100,20, 100,30);
        f.add(value); f.add(l1); f.add(label); f.add(l2); f.add(b); f.add(text);
        f.setSize(300,300);
        f.setLayout(null);
        f.setVisible(true);
        b.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                String data = "Username " + text.getText();
                data += ", Password: "
                + new String(value.getPassword());
                label.setText(data);
            }
        });
    }
}

```

Java JCheckBox

The JCheckBox class is used to create a checkbox. It is used to turn an option on (true) or off (false). Clicking on a CheckBox changes its state from "on" to "off" or from "off" to "on ".It inherits [JToggleButton](#) class.

JCheckBox class declaration

Let's see the declaration for javax.swing.JCheckBox class.

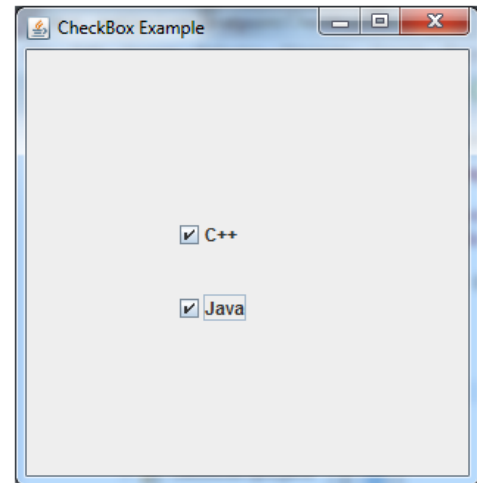
```
public class JCheckBox extends JToggleButton implements Accessible
```

Commonly used Methods:

Methods	Description
AccessibleContext getAccessibleContext()	It is used to get the AccessibleContext associated with this JCheckBox.

Java JCheckBox Example

```
import javax.swing.*;
public class CheckBoxExample
{
    CheckBoxExample(){
        JFrame f= new JFrame("CheckBox Example");
        JCheckBox checkBox1 = new JCheckBox("C++");
        checkBox1.setBounds(100,100, 50,50);
        JCheckBox checkBox2 = new JCheckBox("Java", true);
        checkBox2.setBounds(100,150, 50,50);
        f.add(checkBox1);
        f.add(checkBox2);
        f.setSize(400,400);
        f.setLayout(null);
        f.setVisible(true);
    }
    public static void main(String args[])
    {
        new CheckBoxExample();
    }
}
```



Java JCheckBox Example with ItemListener

```
import javax.swing.*;
import java.awt.event.*;
public class CheckBoxExample
{
    CheckBoxExample(){
        JFrame f= new JFrame("CheckBox Example");
        final JLabel label = new JLabel();
        label.setHorizontalAlignment(JLabel.CENTER);
        label.setSize(400,100);
        JCheckBox checkbox1 = new JCheckBox("C++");
        checkbox1.setBounds(150,100, 50,50);
        JCheckBox checkbox2 = new JCheckBox("Java");
        checkbox2.setBounds(150,150, 50,50);
        f.add(checkbox1); f.add(checkbox2); f.add(label);
        checkbox1.addItemListener(new ItemListener() {
            public void itemStateChanged(ItemEvent e) {
                label.setText("C++ Checkbox: "
                    + (e.getStateChange() == 1 ? "checked" : "unchecked"));
            }
        });
        checkbox2.addItemListener(new ItemListener() {
            public void itemStateChanged(ItemEvent e) {
                label.setText("Java Checkbox: "
                    + (e.getStateChange() == 1 ? "checked" : "unchecked"));
            }
        });
        f.setSize(400,400);
        f.setLayout(null);
        f.setVisible(true);
    }
    public static void main(String args[])
    {
        new CheckBoxExample();
    }
}
```

```
}  
}  
1.
```

Java JCheckBox Example: Food Order

```
import javax.swing.*;  
import java.awt.event.*;  
public class CheckBoxExample extends JFrame implements ActionListener{  
    JLabel l;  
    JCheckBox cb1,cb2,cb3;  
    JButton b;  
    CheckBoxExample(){  
        l=new JLabel("Food Ordering System");  
        l.setBounds(50,50,300,20);  
        cb1=new JCheckBox("Pizza @ 100");  
        cb1.setBounds(100,100,150,20);  
        cb2=new JCheckBox("Burger @ 30");  
        cb2.setBounds(100,150,150,20);  
        cb3=new JCheckBox("Tea @ 10");  
        cb3.setBounds(100,200,150,20);  
        b=new JButton("Order");  
        b.setBounds(100,250,80,30);  
        b.addActionListener(this);  
        add(l);add(cb1);add(cb2);add(cb3);add(b);  
        setSize(400,400);  
        setLayout(null);  
        setVisible(true);  
        setDefaultCloseOperation(EXIT_ON_CLOSE);  
    }  
    public void actionPerformed(ActionEvent e){  
        float amount=0;  
        String msg="";  
        if(cb1.isSelected()){  
            amount+=100;  
            msg="Pizza: 100\n";  
        }  
        if(cb2.isSelected()){  
            amount+=30;  
            msg+="Burger: 30\n";  
        }  
        if(cb3.isSelected()){  
            amount+=10;  
            msg+="Tea: 10\n";  
        }  
        msg+="-----\n";  
        JOptionPane.showMessageDialog(this,msg+"Total: "+amount);  
    }  
    public static void main(String[] args) {  
        new CheckBoxExample();  
    }  
}
```

Java JRadioButton

The JRadioButton class is used to create a radio button. It is used to choose one option from multiple options. It is widely used in exam systems or quiz.

It should be added in ButtonGroup to select one radio button only.

JRadioButton class declaration

Let's see the declaration for javax.swing.JRadioButton class.

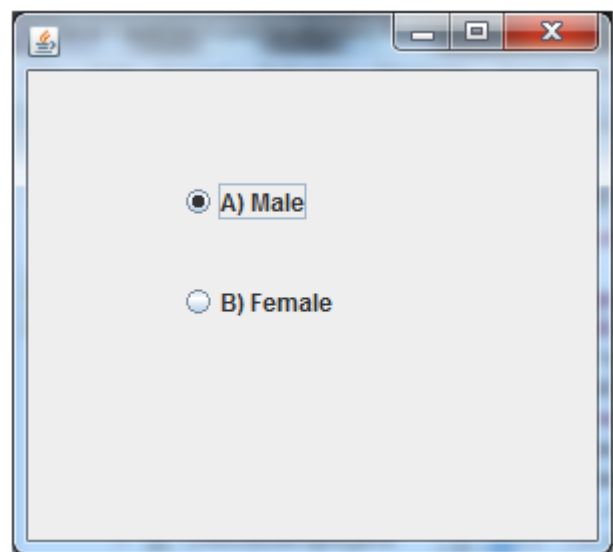
public class JRadioButton **extends** JToggleButton **implements** Accessible

Commonly used Methods:

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 addActionListener(ActionListener a)	It is used to add the action listener to this object.

Java JRadioButton Example

```
import javax.swing.*;
public class RadioButtonExample {
    JFrame f;
    RadioButtonExample(){
        f=new JFrame();
        JRadioButton r1=new JRadioButton("A) Male");
        JRadioButton r2=new JRadioButton("B) Female");
        r1.setBounds(75,50,100,30);
        r2.setBounds(75,100,100,30);
        ButtonGroup bg=new ButtonGroup();
        bg.add(r1);bg.add(r2);
        f.add(r1);f.add(r2);
        f.setSize(300,300);
        f.setLayout(null);
        f.setVisible(true);
    }
    public static void main(String[] args) {
        new RadioButtonExample();
    }
}
```



Java JRadioButton Example with ActionListener

```
import javax.swing.*;
import java.awt.event.*;
class RadioButtonExample extends JFrame implements ActionListener{
    JRadioButton rb1,rb2;
    JButton b;
    RadioButtonExample(){
        rb1=new JRadioButton("Male");
        rb1.setBounds(100,50,100,30);
        rb2=new JRadioButton("Female");
        rb2.setBounds(100,100,100,30);
        ButtonGroup bg=new ButtonGroup();
        bg.add(rb1);bg.add(rb2);
        b=new JButton("click");
        b.setBounds(100,150,80,30);
        b.addActionListener(this);
        add(rb1);add(rb2);add(b);
    }
}
```

```
setSize(300,300);
setLayout(null);
setVisible(true);
}
public void actionPerformed(ActionEvent e){
    if(rb1.isSelected()){
        JOptionPane.showMessageDialog(this,"You are Male.");
    }
    if(rb2.isSelected()){
        JOptionPane.showMessageDialog(this,"You are Female.");
    }
}
public static void main(String args[]){
    new RadioButtonExample();
}}
```

Java JComboBox

The object of Choice class is used to show popup menu of choices. Choice selected by user is shown on the top of a [menu](#). It inherits [JComponent](#) class.

JComboBox class declaration

Let's see the declaration for javax.swing.JComboBox class.

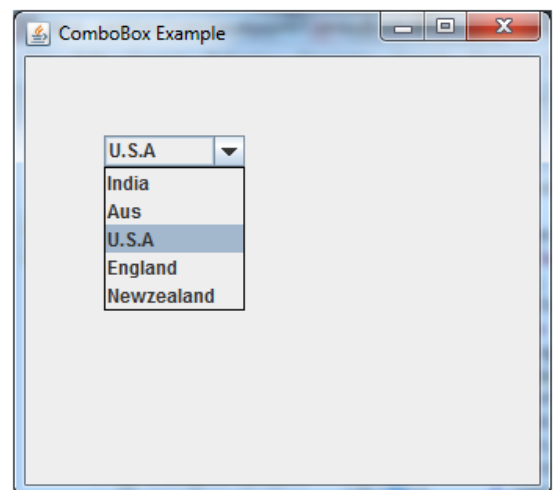
public class JComboBox **extends** JComponent **implements** ItemSelectable, ListDataListener, ActionListener, Accessible

Commonly used Methods:

Methods	Description
void addItem(Object anObject)	It is used to add an item to the item list.
void removeItem(Object anObject)	It is used to delete an item to the item list.
void removeAllItems()	It is used to remove all the items from the list.
void setEditable(boolean b)	It is used to determine whether the JComboBox is editable.
void addActionListener(ActionListener a)	It is used to add the ActionListener .
void addItemListener(ItemListener i)	It is used to add the ItemListener .

Java JComboBox Example

```
import javax.swing.*;
public class ComboBoxExample {
    JFrame f;
    ComboBoxExample(){
        f=new JFrame("ComboBox Example");
        String country[]={"India","Aus","U.S.A","England","Newzealand"};
        JComboBox cb=new JComboBox(country);
        cb.setBounds(50, 50,90,20);
        f.add(cb);
        f.setLayout(null);
        f.setSize(400,500);
        f.setVisible(true);
    }
    public static void main(String[] args) {
        new ComboBoxExample();
    }
}
```



Java JComboBox Example with ActionListener

```
import javax.swing.*;
import java.awt.event.*;
public class ComboBoxExample {
    JFrame f;
    ComboBoxExample(){
        f=new JFrame("ComboBox Example");
        final JLabel label = new JLabel();
        label.setHorizontalAlignment(JLabel.CENTER);
        label.setSize(400,100);
        JButton b=new JButton("Show");
        b.setBounds(200,100,75,20);
        String languages[]={ "C", "C++", "C#", "Java", "PHP" };
    }
}
```

```

final JComboBox cb=new JComboBox(languages);
cb.setBounds(50, 100,90,20);
f.add(cb); f.add(label); f.add(b);
f.setLayout(null);
f.setSize(350,350);
f.setVisible(true);
b.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
String data = "Programming language Selected: "
+ cb.getItemAt(cb.getSelectedIndex());
label.setText(data);
}
});
}
public static void main(String[] args) {
    new ComboBoxExample();
}
}

```

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:

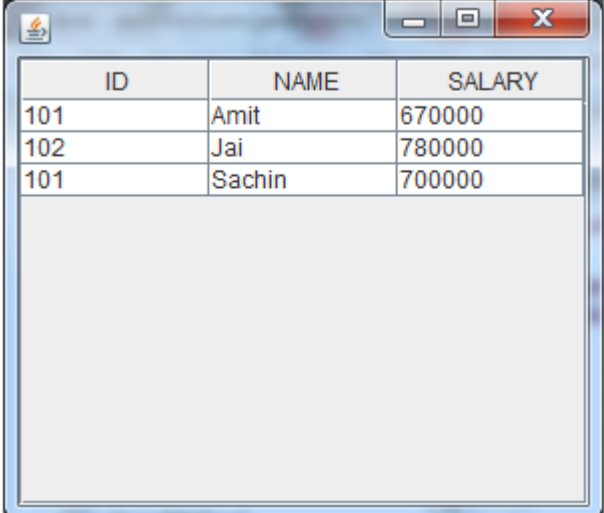
Constructor	Description
JTable()	Creates a table with empty cells.
JTable(Object[][] rows, Object[] columns)	Creates a table with the specified data.

Java JTable Example

```

import javax.swing.*.*;
public class TableExample {
    JFrame f;
    TableExample(){
        f=new JFrame();
        String data[][]={ {"101","Amit","670000"},
                           {"102","Jai","780000"},
                           {"101","Sachin","700000"} };
        String column[]={"ID","NAME","SALARY"};
        JTable jt=new JTable(data,column);
        jt.setBounds(30,40,200,300);
        JScrollPane sp=new JScrollPane(jt);
        f.add(sp);
        f.setSize(300,400);
        f.setVisible(true);
    }
    public static void main(String[] args) {
        new TableExample();
    }
}

```



ID	NAME	SALARY
101	Amit	670000
102	Jai	780000
101	Sachin	700000

Java JTable Example with ListSelectionListener

```

import javax.swing.*.*;
import javax.swing.event.*;
public class TableExample {
    public static void main(String[] a) {
        JFrame f = new JFrame("Table Example");
        String data[][]={ {"101","Amit","670000"},
                           {"102","Jai","780000"},
                           {"101","Sachin","700000"} };
        String column[]={ "ID","NAME","SALARY"};
        final JTable jt=new JTable(data,column);
        jt.setCellSelectionEnabled(true);
        ListSelectionModel select= jt.getSelectionModel();
        select.setSelectionMode(ListSelectionModel.SINGLE_SELECTION);
        select.addListSelectionListener(new ListSelectionListener() {
            public void valueChanged(ListSelectionEvent e) {
                String Data = null;
                int[] row = jt.getSelectedRows();
                int[] columns = jt.getSelectedColumns();
                for (int i = 0; i < row.length; i++) {
                    for (int j = 0; j < columns.length; j++) {
                        Data = (String) jt.getValueAt(row[i], columns[j]);
                    }
                }
                System.out.println("Table element selected is: " + Data);
            }
        });
        JScrollPane sp=new JScrollPane(jt);
        f.add(sp);
        f.setSize(300, 200);
        f.setVisible(true);
    }
}

```

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.

```
public class JList extends JComponent implements Scrollable, Accessible
```

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

```

import javax.swing.*.*;
public class ListExample

```

```

{
    ListExample(){
        JFrame f= new JFrame();
        DefaultListModel<String> l1 = new DefaultListModel<>();
        l1.addElement("Item1");
        l1.addElement("Item2");
        l1.addElement("Item3");
        l1.addElement("Item4");
        JList<String> list = new JList<>(l1);
        list.setBounds(100,100, 75,75);
        f.add(list);
        f.setSize(400,400);
        f.setLayout(null);
        f.setVisible(true);
    }
    public static void main(String args[])
    {
        new ListExample();
    }
}

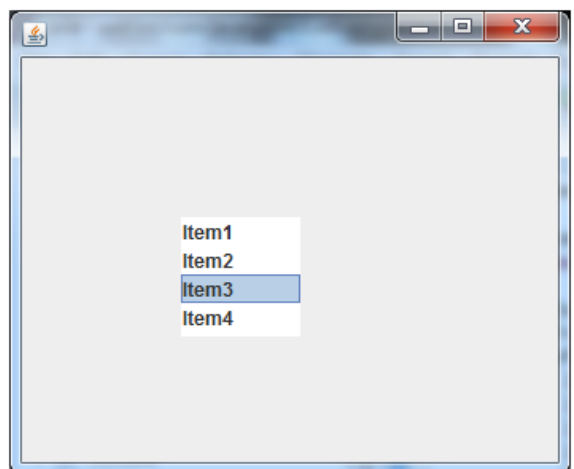
```

Java JList Example with ActionListener

```

import javax.swing.*;
import java.awt.event.*;
public class ListExample
{
    ListExample(){
        JFrame f= new JFrame();
        final JLabel label = new JLabel();
        label.setSize(500,100);
        JButton b=new JButton("Show");
        b.setBounds(200,150,80,30);
        final DefaultListModel<String> l1 = new DefaultListModel<>();
        l1.addElement("C");
        l1.addElement("C++");
        l1.addElement("Java");
        l1.addElement("PHP");
        final JList<String> list1 = new JList<>(l1);
        list1.setBounds(100,100, 75,75);
        DefaultListModel<String> l2 = new DefaultListModel<>();
        l2.addElement("Turbo C++");
        l2.addElement("Struts");
        l2.addElement("Spring");
        l2.addElement("YII");
        final JList<String> list2 = new JList<>(l2);
        list2.setBounds(100,200, 75,75);
        f.add(list1); f.add(list2); f.add(b); f.add(label);
        f.setSize(450,450);
        f.setLayout(null);
        f.setVisible(true);
        b.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                String data = "";
                if (list1.getSelectedIndex() != -1) {
                    data = "Programming language Selected: " + list1.getSelectedValue();
                    label.setText(data);
                }
                if (list2.getSelectedIndex() != -1){
                    data += ", Framework Selected: ";

```



```

        for(Object frame :list2.getSelectedValues()){
            data += frame + " ";
        }
    }
    label.setText(data);
}
});
}
}
public static void main(String args[])
{
    new ListExample();
}}

```

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

public class JOptionPane **extends** JComponent **implements** Accessible

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.

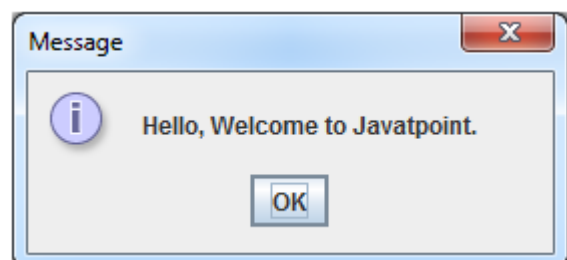
Java JOptionPane Example: showMessageDialog()

```

import javax.swing.*;
public class OptionPaneExample {
    JFrame f;
    OptionPaneExample(){
        f=new JFrame();
        JOptionPane.showMessageDialog(f,"Hello, Welcome to Javatpoint.");
    }
    public static void main(String[] args) {
        new OptionPaneExample();
    }
}

```

Output:



Java JOptionPane Example: showMessageDialog()

```

import javax.swing.*;
public class OptionPaneExample {
    JFrame f;
    OptionPaneExample(){
        f=new JFrame();
        JOptionPane.showMessageDialog(f,"Successfully Updated.", "Alert",JOptionPane.WARNING_MESSAGE);
    }
    public static void main(String[] args) {
        new OptionPaneExample();
    }
}

```

Java JOptionPane Example: showInputDialog()

```

import javax.swing.*;
public class OptionPaneExample {
    JFrame f;
    OptionPaneExample(){
        f=new JFrame();
        String name=JOptionPane.showInputDialog(f,"Enter Name");
    }
    public static void main(String[] args) {
        new OptionPaneExample();
    }
}

```

Java JOptionPane Example: showConfirmDialog()

```

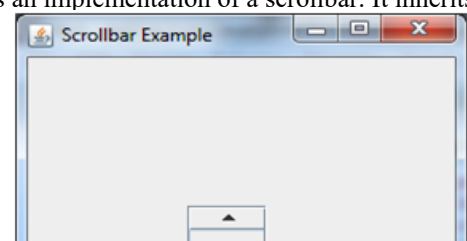
import javax.swing.*;
import java.awt.event.*;
public class OptionPaneExample extends WindowAdapter{
    JFrame f;
    OptionPaneExample(){
        f=new JFrame();
        f.addWindowListener(this);
        f.setSize(300, 300);
        f.setLayout(null);
        f.setDefaultCloseOperation(JFrame.DO_NOTHING_ON_CLOSE);
        f.setVisible(true);
    }
    public void windowClosing(WindowEvent e) {
        int a=JOptionPane.showConfirmDialog(f,"Are you sure?");
        if(a==JOptionPane.YES_OPTION){
            f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        }
    }
    public static void main(String[] args) {
        new OptionPaneExample();
    }
}

```

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.

public class JScrollBar **extends** JComponent **implements** Adjustable, Accessible

Java JScrollBar Example

```
import javax.swing.*;
class ScrollBarExample
{
    ScrollBarExample(){
        JFrame f= new JFrame("Scrollbar Example");
        JScrollBar s=new JScrollBar();
        s.setBounds(100,100, 50,100);
        f.add(s);
        f.setSize(400,400);
        f.setLayout(null);
        f.setVisible(true);
    }
    public static void main(String args[])
    {
        new ScrollBarExample();
    }
}
```

Java JScrollBar Example with AdjustmentListener

```
import javax.swing.*;
import java.awt.event.*;
class ScrollBarExample
{
    ScrollBarExample(){
        JFrame f= new JFrame("Scrollbar Example");
        final JLabel label = new JLabel();
        label.setHorizontalAlignment(JLabel.CENTER);
        label.setSize(400,100);
        final JScrollBar s=new JScrollBar();
        s.setBounds(100,100, 50,100);
        f.add(s); f.add(label);
        f.setSize(400,400);
        f.setLayout(null);
        f.setVisible(true);
        s.addAdjustmentListener(new AdjustmentListener() {
            public void adjustmentValueChanged(AdjustmentEvent e) {
                label.setText("Vertical Scrollbar value is:"+ s.getValue());
            }
        });
    }
    public static void main(String args[])
    {
        new ScrollBarExample();
    }
}
```

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

public class JMenuBar **extends** JComponent **implements** MenuElement, Accessible

JMenu class declaration

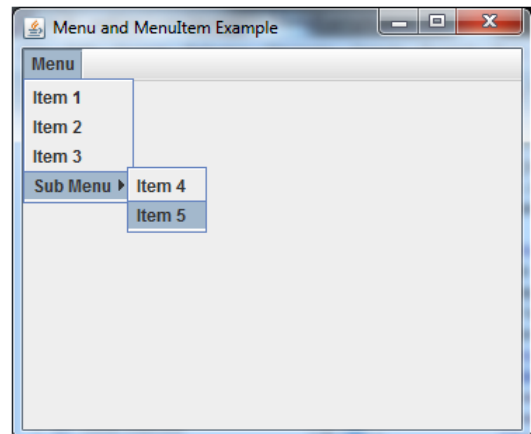
public class JMenu **extends** JMenuItem **implements** MenuElement, Accessible

JMenuItem class declaration

public class JMenuItem **extends** AbstractButton **implements** Accessible, MenuElement

Java JMenuItem and JMenu Example

```
import javax.swing.*;
class MenuExample
{
    JMenu menu, submenu;
    JMenuItem i1, i2, i3, i4, i5;
    MenuExample(){
        JFrame f= new JFrame("Menu and MenuItem Example");
        JMenuBar mb=new JMenuBar();
        menu=new JMenu("Menu");
        submenu=new JMenu("Sub Menu");
        i1=new JMenuItem("Item 1");
        i2=new JMenuItem("Item 2");
        i3=new JMenuItem("Item 3");
        i4=new JMenuItem("Item 4");
        i5=new JMenuItem("Item 5");
        menu.add(i1); menu.add(i2); menu.add(i3);
        submenu.add(i4); submenu.add(i5);
        menu.add(submenu);
        mb.add(menu);
        f.setJMenuBar(mb);
        f.setSize(400,400);
        f.setLayout(null);
        f.setVisible(true);
    }
    public static void main(String args[])
    {
        new MenuExample();
    }
}
```



Example of creating Edit menu for Notepad:

```
import javax.swing.*;
import java.awt.event.*;
public class MenuExample implements ActionListener{
    JFrame f;
    JMenuBar mb;
    JMenu file,edit,help;
    JMenuItem cut,copy,paste,selectAll;
    JTextArea ta;
    MenuExample(){
        f=new JFrame();
        cut=new JMenuItem("cut");
        copy=new JMenuItem("copy");
        paste=new JMenuItem("paste");
        selectAll=new JMenuItem("selectAll");
```

```

cut.addActionListener(this);
copy.addActionListener(this);
paste.addActionListener(this);
selectAll.addActionListener(this);
mb=new JMenuBar();
file=new JMenu("File");
edit=new JMenu("Edit");
help=new JMenu("Help");
edit.add(cut);edit.add(copy);edit.add(paste);edit.add(selectAll);
mb.add(file);mb.add(edit);mb.add(help);
ta=new JTextArea();
ta.setBounds(5,5,360,320);
f.add(mb);f.add(ta);
f.setJMenuBar(mb);
f.setLayout(null);
f.setSize(400,400);
f.setVisible(true);
}
public void actionPerformed(ActionEvent e) {
if(e.getSource()==cut)
ta.cut();
if(e.getSource()==paste)
ta.paste();
if(e.getSource()==copy)
ta.copy();
if(e.getSource()==selectAll)
ta.selectAll();
}
public static void main(String[] args) {
    new MenuExample();
}
}

```

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.

public class JPopupMenu **extends** JComponent **implements** Accessible, MenuElement

Commonly used Constructors:

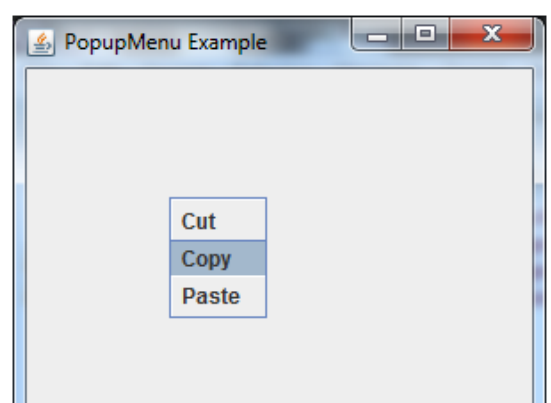
Constructor	Description
JPopupMenu()	Constructs a JPopupMenu without an "invoker".
JPopupMenu(String label)	Constructs a JPopupMenu with the specified title.

Java JPopupMenu Example

```

import javax.swing.*;
import java.awt.event.*;
class PopupMenuExample
{
    PopupMenuExample(){
        final JFrame f= new JFrame("PopupMenu Example");
        final JPopupMenu popupmenu = new JPopupMenu("Edit");
    }
}

```



```

JMenuItem cut = new JMenuItem("Cut");
JMenuItem copy = new JMenuItem("Copy");
JMenuItem paste = new JMenuItem("Paste");
popupmenu.add(cut); popupmenu.add(copy); popupmenu.add(paste);
f.addMouseListener(new MouseAdapter() {
    public void mouseClicked(MouseEvent e) {
        popupmenu.show(f, e.getX(), e.getY());
    }
});
f.add(popupmenu);
f.setSize(300,300);
f.setLayout(null);
f.setVisible(true);
}
public static void main(String args[])
{
    new PopupMenuExample();
}
}

```

Java JPopupMenu Example with MouseListener and ActionListener

```

import javax.swing.*;
import java.awt.event.*;
class PopupMenuExample
{
    PopupMenuExample(){
        final JFrame f= new JFrame("PopupMenu Example");
        final JLabel label = new JLabel();
        label.setHorizontalAlignment(JLabel.CENTER);
        label.setSize(400,100);
        final JPopupMenu popupmenu = new JPopupMenu("Edit");
        JMenuItem cut = new JMenuItem("Cut");
        JMenuItem copy = new JMenuItem("Copy");
        JMenuItem paste = new JMenuItem("Paste");
        popupmenu.add(cut); popupmenu.add(copy); popupmenu.add(paste);
        f.addMouseListener(new MouseAdapter() {
            public void mouseClicked(MouseEvent e) {
                popupmenu.show(f, e.getX(), e.getY());
            }
        });
        cut.addActionListener(new ActionListener(){
            public void actionPerformed(ActionEvent e) {
                label.setText("cut MenuItem clicked.");
            }
        });
        copy.addActionListener(new ActionListener(){
            public void actionPerformed(ActionEvent e) {
                label.setText("copy MenuItem clicked.");
            }
        });
        paste.addActionListener(new ActionListener(){
            public void actionPerformed(ActionEvent e) {
                label.setText("paste MenuItem clicked.");
            }
        });
        f.add(label); f.add(popupmenu);
        f.setSize(400,400);
        f.setLayout(null);
    }
}

```

```

        f.setVisible(true);
    }
    public static void main(String args[])
    {
        new PopupMenuExample();
    }
}

```

Java JCheckBoxMenuItem

JCheckBoxMenuItem class represents [checkbox](#) which can be included on a [menu](#) . A JCheckBoxMenuItem can have text or a graphic icon or both, associated with it. [MenuItem](#) can be selected or deselected. MenuItems can be configured and controlled by actions.

Nested class

Modifier and Type	Class	Description
protected class	JCheckBoxMenuItem.AccessibleJCheckBoxMenuItem	This class implements accessibility support for the JCheckBoxMenuItem class.

Constructor

Constructor	Description
JCheckBoxMenuItem()	It creates an initially unselected check box menu item with no set text or icon.
JCheckBoxMenuItem(Action a)	It creates a menu item whose properties are taken from the Action supplied.
JCheckBoxMenuItem(Icon icon)	It creates an initially unselected check box menu item with an icon.
JCheckBoxMenuItem(String text)	It creates an initially unselected check box menu item with text.
JCheckBoxMenuItem(String text, boolean b)	It creates a check box menu item with the specified text and selection state.
JCheckBoxMenuItem(String text, Icon icon)	It creates an initially unselected check box menu item with the specified text and icon.
JCheckBoxMenuItem(String text, Icon icon, boolean b)	It creates a check box menu item with the specified text, icon, and selection state.

Methods

Modifier	Method	Description
AccessibleContext	getAccessibleContext()	It gets the AccessibleContext associated with this JCheckBoxMenuItem.
Object[]	getSelectedObjects()	It returns an array (length 1) containing the check box menu item label or null if the check box is not selected.
boolean	getState()	It returns the selected-state of the item.
String	getUIClassID()	It returns the name of the L&F class that renders this component.
protected String	paramString()	It returns a string representation of this JCheckBoxMenuItem.
void	setState(boolean b)	It sets the selected-state of the item.

Java JCheckBoxMenuItem Example

```

import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.KeyEvent;
import javax.swing.AbstractButton;

```

```

import javax.swing.Icon;
import javax.swing.JCheckBoxMenuItem;
import javax.swing.JFrame;
import javax.swing.JMenu;
import javax.swing.JMenuBar;
import javax.swing.JMenuItem;

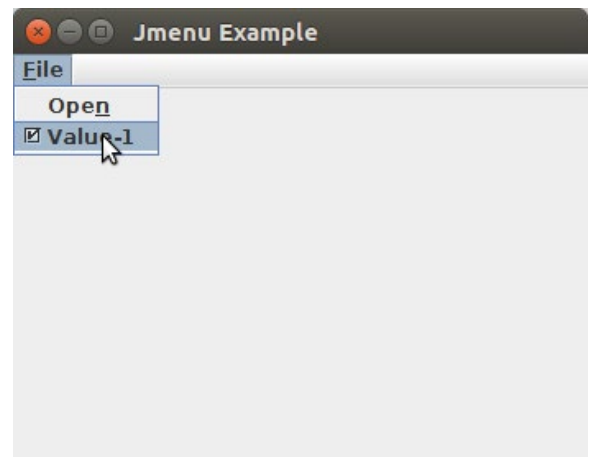
public class JavaCheckBoxMenuItem {
    public static void main(final String args[]) {
        JFrame frame = new JFrame("Jmenu Example");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        JMenuBar menuBar = new JMenuBar();
        // File Menu, F - Mnemonic
        JMenu fileMenu = new JMenu("File");
        fileMenu.setMnemonic(KeyEvent.VK_F);
        menuBar.add(fileMenu);
        // File->New, N - Mnemonic
        JMenuItem menuItem1 = new JMenuItem("Open", KeyEvent.VK_N);
        fileMenu.add(menuItem1);

        JCheckBoxMenuItem caseMenuItem = new JCheckBoxMenuItem("Option_1");
        caseMenuItem.setMnemonic(KeyEvent.VK_C);
        fileMenu.add(caseMenuItem);

        ActionListener aListener = new ActionListener() {
            public void actionPerformed(ActionEvent event) {
                AbstractButton aButton = (AbstractButton) event.getSource();
                boolean selected = aButton.getModel().isSelected();
                String newLabel;
                Icon newIcon;
                if (selected) {
                    newLabel = "Value-1";
                } else {
                    newLabel = "Value-2";
                }
                aButton.setText(newLabel);
            }
        };

        caseMenuItem.addActionListener(aListener);
        frame.setJMenuBar(menuBar);
        frame.setSize(350, 250);
        frame.setVisible(true);
    }
}

```



Java JDialog

The JDialog control represents a top level window with a border and a title used to take some form of input from the user. It inherits the Dialog class.

Unlike JFrame, it doesn't have maximize and minimize buttons.

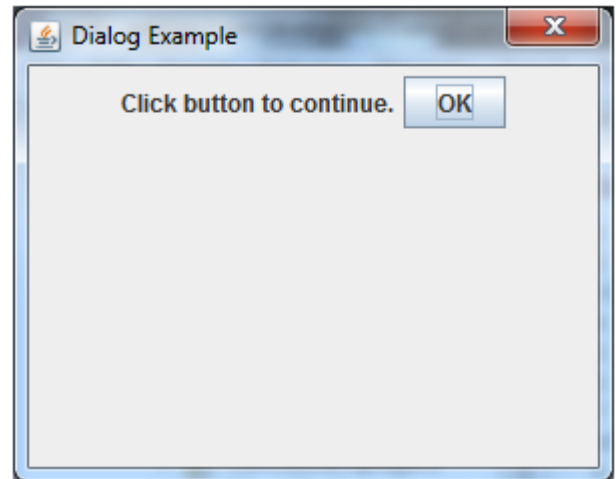
JDialog class declaration

Let's see the declaration for javax.swing.JDialog class.

public class JDialog **extends** Dialog **implements** WindowConstants, Accessible, RootPaneContainer

Java JDialog Example

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
public class DialogExample {
    private static JDialog d;
    DialogExample() {
        JFrame f= new JFrame();
        d = new JDialog(f, "Dialog Example", true);
        d.setLayout( new FlowLayout() );
        JButton b = new JButton ("OK");
        b.addActionListener ( new ActionListener()
        {
            public void actionPerformed( ActionEvent e )
            {
                DialogExample.d.setVisible(false);
            }
        });
        d.add( new JLabel ("Click button to continue."));
        d.add(b);
        d.setSize(300,300);
        d.setVisible(true);
    }
    public static void main(String args[])
    {
        new DialogExample();
    }
}
```



Java JPanel

The JPanel is a simplest container class. It provides space in which an application can attach any other component. It inherits the JComponents class.

It doesn't have title bar.

JPanel class declaration

public class JPanel **extends** JComponent **implements** Accessible

Java JPanel Example

```
import java.awt.*;
import javax.swing.*;
public class PanelExample {
    PanelExample()
    {
        JFrame f= new JFrame("Panel Example");
        JPanel panel=new JPanel();
        panel.setBounds(40,80,200,200);
        panel.setBackground(Color.gray);
        JButton b1=new JButton("Button 1");
        b1.setBounds(50,100,80,30);
        b1.setBackground(Color.yellow);
```



```
JButton b2=new JButton("Button 2");
b2.setBounds(100,100,80,30);
b2.setBackground(Color.green);
panel.add(b1); panel.add(b2);
f.add(panel);
    f.setSize(400,400);
    f.setLayout(null);
    f.setVisible(true);
}
public static void main(String args[])
{
    new PanelExample();
}
}
```


Java JFileChooser

The object of JFileChooser class represents a dialog window from which the user can select file. It inherits JComponent class.

JFileChooser class declaration

Let's see the declaration for javax.swing.JFileChooser class.

public class JFileChooser **extends** JComponent **implements** Accessible

Java JFileChooser Example

```
import javax.swing.*;
import java.awt.event.*;
import java.io.*;
public class FileChooserExample extends JFrame implements ActionListener{
    JMenuBar mb;
    JMenu file;
    JMenuItem open;
    JTextArea ta;
    FileChooserExample(){
        open=new JMenuItem("Open File");
        open.addActionListener(this);
        file=new JMenu("File");
        file.add(open);
        mb=new JMenuBar();
        mb.setBounds(0,0,800,20);
        mb.add(file);
        ta=new JTextArea(800,800);
        ta.setBounds(0,20,800,800);
        add(mb);
        add(ta);
    }
    public void actionPerformed(ActionEvent e) {
        if(e.getSource()==open){
            JFileChooser fc=new JFileChooser();
            int i=fc.showOpenDialog(this);
            if(i==JFileChooser.APPROVE_OPTION){
                File f=fc.getSelectedFile();
                String filepath=f.getPath();
                try{
                    BufferedReader br=new BufferedReader(new FileReader(filepath));
                    String s1="",s2="";
                    while((s1=br.readLine())!=null){
                        s2+=s1+"\n";
                    }
                    ta.setText(s2);
                    br.close();
                }catch (Exception ex) {ex.printStackTrace(); }
            }
        }
    }
    public static void main(String[] args) {
        FileChooserExample om=new FileChooserExample();
        om.setSize(500,500);
        om.setLayout(null);
        om.setVisible(true);
        om.setDefaultCloseOperation(EXIT_ON_CLOSE);
    }
}
```

}

Java JFrame

The `javax.swing.JFrame` class is a type of container which inherits the `java.awt.Frame` class. `JFrame` works like the main window where components like labels, buttons, textfields are added to create a GUI.

Unlike `Frame`, `JFrame` has the option to hide or close the window with the help of `setDefaultCloseOperation(int)` method.

Nested Class

Modifier and Type	Class	Description
protected class	<code>JFrame.AccessibleJFrame</code>	This class implements accessibility support for the <code>JFrame</code> class.

Fields

Modifier and Type	Field	Description
protected <code>AccessibleContext</code>	<code>accessibleContext</code>	The accessible context property.
static int	<code>EXIT_ON_CLOSE</code>	The exit application default window close operation.
protected <code>JRootPane</code>	<code>rootPane</code>	The <code>JRootPane</code> instance that manages the <code>contentPane</code> and optional <code>menuBar</code> for this frame, as well as the <code>glassPane</code> .
protected boolean	<code>rootPaneCheckingEnabled</code>	If true then calls to <code>add</code> and <code>setLayout</code> will be forwarded to the <code>contentPane</code> .

Constructors

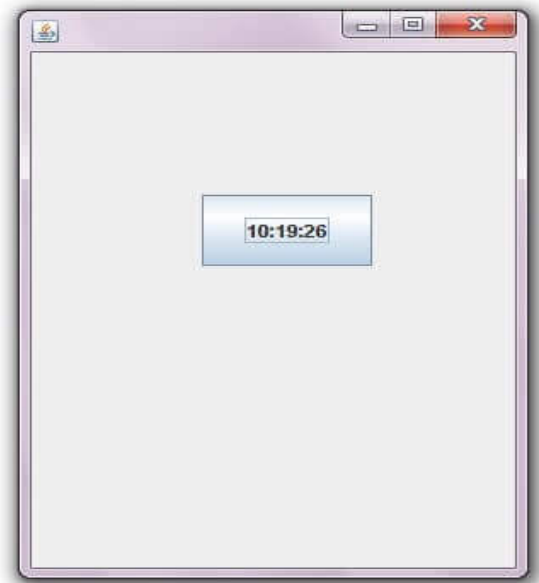
Constructor	Description
<code>JFrame()</code>	It constructs a new frame that is initially invisible.
<code>JFrame(GraphicsConfiguration gc)</code>	It creates a <code>Frame</code> in the specified <code>GraphicsConfiguration</code> of a screen device and a blank title.
<code>JFrame(String title)</code>	It creates a new, initially invisible <code>Frame</code> with the specified title.
<code>JFrame(String title, GraphicsConfiguration gc)</code>	It creates a <code>JFrame</code> with the specified title and the specified <code>GraphicsConfiguration</code> of a screen device.

Useful Methods

Modifier and Type	Method	Description
protected void	<code>addImpl(Component comp, Object constraints, int index)</code>	Adds the specified child <code>Component</code> .
protected <code>JRootPane</code>	<code>createRootPane()</code>	Called by the constructor methods to create the default <code>rootPane</code> .
protected void	<code>frameInit()</code>	Called by the constructors to init the <code>JFrame</code> properly.
void	<code>setContentPane(Container contentPane)</code>	It sets the <code>contentPane</code> property
static void	<code>setDefaultLookAndFeelDecorated(boolean defaultLookAndFeelDecorated)</code>	Provides a hint as to whether or not newly created <code>JFrames</code> should have their <code>Window</code> decorations (such as borders, widgets to close the window, title...) provided by the current look and feel.
void	<code>setIconImage(Image image)</code>	It sets the image to be displayed as the icon for this window.
void	<code>setJMenuBar(JMenuBar menubar)</code>	It sets the <code>menubar</code> for this frame.
void	<code>setLayeredPane(JLayeredPane layeredPane)</code>	It sets the <code>layeredPane</code> property.
<code>JRootPane</code>	<code>getRootPane()</code>	It returns the <code>rootPane</code> object for this frame.
<code>TransferHandler</code>	<code>getTransferHandler()</code>	It gets the <code>transferHandler</code> property.

JFrame Example

```
import java.awt.FlowLayout;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JPanel;
public class JFrameExample {
    public static void main(String s[]) {
        JFrame frame = new JFrame("JFrame Example");
        JPanel panel = new JPanel();
        panel.setLayout(new FlowLayout());
        JLabel label = new JLabel("JFrame By Example");
        JButton button = new JButton();
        button.setText("Button");
        panel.add(label);
        panel.add(button);
        frame.add(panel);
        frame.setSize(200, 300);
        frame.setLocationRelativeTo(null);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setVisible(true);
    }
}
```



Example of digital clock in swing:

```
import javax.swing.*;
import java.awt.*;
import java.text.*;
import java.util.*;
public class DigitalWatch implements Runnable{
    JFrame f;
    Thread t=null;
    int hours=0, minutes=0, seconds=0;
    String timeString = "";
    JButton b;

    DigitalWatch(){
        f=new JFrame();

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

        b=new JButton();
        b.setBounds(100,100,100,50);

        f.add(b);
        f.setSize(300,400);
        f.setLayout(null);
        f.setVisible(true);
    }

    public void run() {
        try {
            while (true) {

                Calendar cal = Calendar.getInstance();
```

```

hours = cal.get( Calendar.HOUR_OF_DAY );
if ( hours > 12 ) hours -= 12;
minutes = cal.get( Calendar.MINUTE );
seconds = cal.get( Calendar.SECOND );

SimpleDateFormat formatter = new SimpleDateFormat("hh:mm:ss");
Date date = cal.getTime();
timeString = formatter.format( date );

printTime();

t.sleep( 1000 ); // interval given in milliseconds
}
}
catch (Exception e) { }
}

public void printTime(){
b.setText(timeString);
}

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

```

Displaying image in swing:

For displaying image, we can use the method `drawImage()` of `Graphics` class.

Syntax of `drawImage()` method:

`public abstract boolean drawImage(Image img, int x, int y, ImageObserver observer);` is used draw the specified image.

Example of displaying image in swing:

```

import java.awt.*;

import javax.swing.JFrame;

public class MyCanvas extends Canvas{

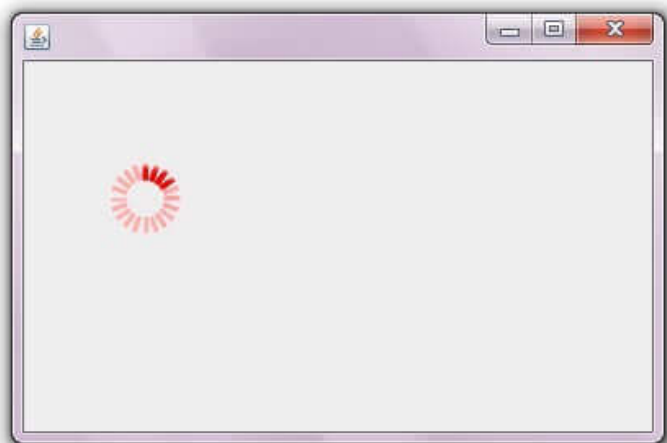
    public void paint(Graphics g) {

        Toolkit t=Toolkit.getDefaultToolkit();
        Image i=t.getImage("p3.gif");
        g.drawImage(i, 120,100,this);

    }

    public static void main(String[] args) {
        MyCanvas m=new MyCanvas();
        JFrame f=new JFrame();
        f.add(m);
        f.setSize(400,400);
        f.setVisible(true);
    }
}

```



Ch-12 Java - Applet Basics

An applet is a Java program that runs in a Web browser. An applet can be a fully functional Java application because it has the entire Java API at its disposal.

There are some important differences between an applet and a standalone Java application, including the following –

- An applet is a Java class that extends the `java.applet.Applet` class.
- A `main()` method is not invoked on an applet, and an applet class will not define `main()`.
- Applets are designed to be embedded within an HTML page.
- When a user views an HTML page that contains an applet, the code for the applet is downloaded to the

user's machine.

- A JVM is required to view an applet. The JVM can be either a plug-in of the Web browser or a separate

runtime environment.

- The JVM on the user's machine creates an instance of the applet class and invokes various methods

during the applet's lifetime.

- Applets have strict security rules that are enforced by the Web browser. The security of an applet is

often referred to as sandbox security, comparing the applet to a child playing in a sandbox with various

rules that must be followed.

- Other classes that the applet needs can be downloaded in a single Java Archive (JAR) file.

Life Cycle of an Applet

Four methods in the Applet class gives you the framework on which you build any serious applet –

- `init` – This method is intended for whatever

initialization is needed for your applet. It is

called after the `param` tags inside the applet

tag have been processed.

- `start` – This method is automatically called

after the browser calls the `init` method. It is

also called whenever the user returns to the

page containing the applet after having gone

off to other pages.

- `stop` – This method is automatically called

when the user moves off the page on which

the applet sits. It can, therefore, be called

repeatedly in the same applet.

- `destroy` – This method is only called when

the browser shuts down normally. Because applets are meant to live on an HTML page, you should not

normally leave resources behind after a user leaves the page that contains the applet. `paint` – Invoked immediately after the `start()` method, and also any time the applet needs to repaint itself

in the browser. The `paint()` method is actually inherited from the `java.awt`.

A "Hello, World" Applet

Following is a simple applet named `HelloWorldApplet.java` –

```
import java.applet.*;

import java.awt.*;

public class HelloWorldApplet extends Applet {

    public void paint (Graphics g) {

        g.drawString ("Hello World", 25, 50);

    }

}
```

Invoking an Applet

An applet may be invoked by embedding directives in an HTML file and viewing the file through an applet viewer or Java-enabled browser.

The `<applet>` tag is the basis for embedding an applet in an HTML file. Following is an example that invokes the "Hello, World" applet –

```
<html>

<title>The Hello, World Applet</title>

<hr>

<applet code = "HelloWorldApplet.class" width = "320" height = "120">
```

If your browser was Java-enabled, a "Hello, World" message would appear here.

```
</applet>

<hr>

</html>
```

Note – You can refer to [HTML Applet Tag](#) to understand more about calling applet from HTML.

Getting Applet Parameters

The following example demonstrates how to make an applet respond to setup parameters specified in the document. This applet displays a checkerboard pattern of black and a second color.

The second color and the size of each square may be specified as parameters to the applet within the document.

CheckerApplet gets its parameters in the init() method. It may also get its parameters in the paint() method.

However, getting the values and saving the settings once at the start of the applet, instead of at every refresh, is convenient and efficient.

The applet viewer or browser calls the init() method of each applet it runs. The viewer calls init() once, immediately after loading the applet. (Applet.init() is implemented to do nothing.) Override the default implementation to insert custom initialization code.

The Applet.getParameter() method fetches a parameter given the parameter's name (the value of a parameter is always a string). If the value is numeric or other non-character data, the string must be parsed.

The following is a skeleton of CheckerApplet.java –

```
import java.applet.*;
import java.awt.*;

public class CheckerApplet extends Applet {
    int squareSize = 50; // initialized to default size

    public void init() {}

    private void parseSquareSize (String param) {}

    private Color parseColor (String param) {}

    public void paint (Graphics g) {}
}
```

Here are CheckerApplet's init() and private parseSquareSize() methods –

```
public void init () {
    String squareSizeParam = getParameter ("squareSize");
    parseSquareSize (squareSizeParam);

    String colorParam = getParameter ("color");
    Color fg = parseColor (colorParam);

    setBackground (Color.black);
    setForeground (fg);
}

private void parseSquareSize (String param) {
```

```

if (param == null) return;

try {

squareSize = Integer.parseInt (param);

} catch (Exception e) {

// Let default value remain

}

}

```

The applet calls `parseSquareSize()` to parse the `squareSize` parameter. `parseSquareSize()` calls the library method `Integer.parseInt()`, which parses a string and returns an integer. `Integer.parseInt()` throws an exception whenever its argument is invalid.

Therefore, `parseSquareSize()` catches exceptions, rather than allowing the applet to fail on bad input.

The applet calls `parseColor()` to parse the color parameter into a `Color` value. `parseColor()` does a series of string comparisons to match the parameter value to the name of a predefined color. You need to implement these methods to make this applet work.

Specifying Applet Parameters

The following is an example of an HTML file with a `CheckerApplet` embedded in it. The HTML file specifies both parameters to the applet by means of the `<param>` tag.

```

<html>

<title>Checkerboard Applet</title>

<hr>

<applet code = "CheckerApplet.class" width = "480" height = "320">

<param name = "color" value = "blue">

<param name = "squaresize" value = "30">

</applet>

<hr>

</html>

```

Note – Parameter names are not case sensitive.

CH-13 JDBC Tutorial

JDBC API is a Java API that can access any kind of tabular data, especially data stored in a Relational Database. JDBC works with Java on a variety of platforms, such as Windows, Mac OS, and the various versions of UNIX.

Why to Learn JDBC?

JDBC stands for Java Database Connectivity, which is a standard Java API for database-independent connectivity between

the Java programming language and a wide range of databases.

The JDBC library includes APIs for each of the tasks mentioned below that are commonly associated with database usage.

Import JDBC Packages

```
import java.sql.* ; // for standard JDBC programs
```

```
import java.math.* ; // for BigDecimal and BigInteger support
```

Register JDBC driver

```
Method1 Class.forName("com.mysql.jdbc.Driver");
```

```
Method2 Driver myDriver = new oracle.jdbc.driver.OracleDriver();
```

```
DriverManager.registerDriver( myDriver );
```

Making a connection to a database.

RDBMS JDBC driver name URL format

MySQL `com.mysql.jdbc.Driver jdbc:mysql://hostname/ databaseName`

ORACLE `oracle.jdbc.driver.OracleDriver jdbc:oracle:thin:@hostname:port`

`Number:databaseName`

DB2 `COM.ibm.db2.jdbc.net.DB2Driver jdbc:db2:hostname:port Number/databaseName`

Sybase `com.sybase.jdbc.SybDriver jdbc:sybase:Tds:hostname: port`

`Number/databaseName`

for mysql Database.

```
static final String DB_URL = "jdbc:mysql://localhost/";
```

```
static final String USER = "username";
```

```
static final String PASS = "password";
```

```
Connection conn = DriverManager.getConnection(DB_URL, USER, PASS);
```

OR for Access Database.

```
String databaseURL = "jdbc:ucanaccess://e://Java//JavaSE//MsAccess//Contacts.accdb";
```

Connection connection = DriverManager.getConnection(databaseURL)

Creating SQL or MySQL statements.

Fetching data from database

"SELECT Lname FROM Customers WHERE Snum = 2001"

Inserting data to database

"INSERT INTO EMPLOYEE (ID,FIRST_NAME,LAST_NAME,STAT_CD) VALUES (1,'Lokesh','Gupta',5)"

Updating data values in database

"update users set num_points = 6000 where id = 2"

Deleting data from database

"delete from employee where emp_id=?";

Executing SQL or MySQL queries in the database.

Statement stmt;

stmt= conn.createStatement();

ResultSet rs = stmt.executeQuery(sql); //for select command

stmt.executeUpdate(sql);

Viewing & Modifying the resulting records.

JDBC - Insert Records Example

```
import java.sql.*;
```

```
public class JDBCExample {
```

```
    static final String JDBC_DRIVER = "com.mysql.jdbc.Driver";
```

```
    static final String DB_URL = "jdbc:mysql://localhost/STUDENTS";
```

```
    static final String USER = "username";
```

```
    static final String PASS = "password";
```

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

```
        Connection conn = null;
```

```
        Statement stmt = null;
```

```
        try{
```

```
            Class.forName("com.mysql.jdbc.Driver");
```

```
            System.out.println("Connecting to a selected database...");
```

```
            conn = DriverManager.getConnection(DB_URL, USER, PASS);
```

```
System.out.println("Connected database successfully...");

System.out.println("Inserting records into the table...");

stmt = conn.createStatement();

String sql = "INSERT INTO Registration " + "VALUES (100, 'Zara', 'Ali', 18)";

stmt.executeUpdate(sql);

System.out.println("Inserted records into the table...");

}

catch(SQLException se){ //Handle errors for JDBC

se.printStackTrace();

}

}

}

//end main

}

//end JDBCExample
```

```
ResultSet rs = stmt.executeQuery(sql);

while(rs.next()){

int id = rs.getInt("id");

int age = rs.getInt("age");

String first = rs.getString("first");

String last = rs.getString("last");

System.out.print("ID: " + id);

System.out.print(", Age: " + age);

System.out.print(", First: " + first);

System.out.println(", Last: " + last);

}

rs.close();

}catch(SQLException se){

//Handle errors for JDBC

se.printStackTrace();

}

System.out.println("Goodbye!");

} //end main

} //end JDBCExample

Closing JDBC Connections

conn.close();
```