CS-19 Programming with JAVA

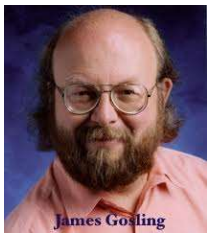
BCA Semester – 4

BCA Department

Unit -1

History, Introduction and Language Basics, Classes and Objects

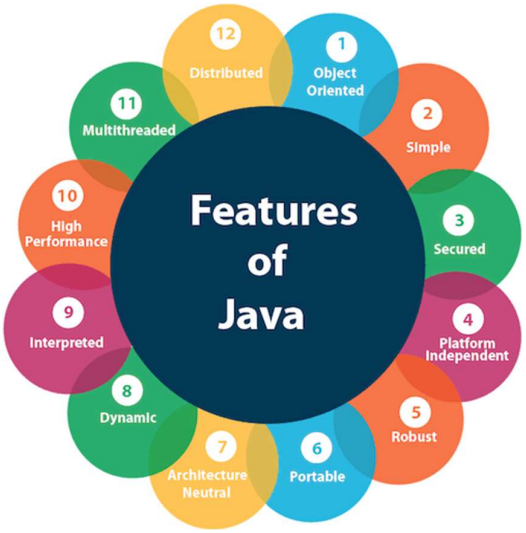
History and Features of Java



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

The principles for creating Java programming were "Simple, Robust, Portable, Platform-independent, Secured, High Performance, Multithreaded, Architecture Neutral, Object-Oriented, Interpreted, and Dynamic". Java was developed by James Gosling, who is known as the father of Java, in 1995. James Gosling and his team members started the project in the early '90s. James Gosling - founder of java Currently, Java is used in internet programming, mobile devices, games, e-business solutions, etc. There are given significant points that describe the history of Java.

Features of JAVA



Simple: Java is very easy to learn, and its syntax is simple, clean and easy to understand. According to Sun, Java language is a simple programming language because: Java syntax is based on C++ (so easier for programmers to learn it after C++). Java has removed many complicated and rarely-used features, for example, explicit pointers, operator overloading, etc. There is no need to remove unreferenced objects because there is an Automatic Garbage Collection in Java.

Object-Oriented: Java is an object-oriented programming language. Everything in Java is an object. Object-oriented means we organize our software as a combination of different types of objects that incorporates both data and behavior. Object-oriented programming (OOPs) is a methodology that simplifies software development and maintenance by providing some rules.

Portable: 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.

Platform Independent: Java is platform independent because it is different from other languages like C, C++, etc. which are compiled into platform specific machines while Java is a write once, run anywhere language. A platform is the hardware or software environment in which a program runs. There are two types of platforms software-based and hardware-based. Java provides a software-based platform. The Java platform differs from most other platforms in the sense that it is a software-based platform that runs on the top of other hardware-based platforms. It has two components:

Secured: Java is best known for its security. With Java, we can develop virus-free systems. Java is secured because: No explicit pointer, and Java Programs run inside a virtual machine sandbox. Java language provides these securities by default. Some security can also be provided by an application developer explicitly through SSL, JAAS, Cryptography, etc.

- **Bytecode Verifier:** It checks the code fragments for illegal code that can violate access right to objects.
- **Security Manager:** It determines what resources a class can access such as reading and writing to the local disk.

Robust: Robust simply means strong. Java is robust because It uses strong memory management, There is a lack of pointers that avoids security problems, and There is automatic garbage collection in java which runs on the Java Virtual Machine to get rid of objects which are not being used by a Java application anymore. There are exception handling and the type checking mechanism in Java. All these points make Java robust.

Architecture-neutral: Java is architecture neutral because there are no implementation dependent features, for example, the size of primitive types is fixed. In C programming, int data type occupies 2 bytes of memory for 32-bit architecture and 4 bytes of memory for 64-bit architecture. However, it occupies 4 bytes of memory for both 32 and 64-bit architectures in Java.

High-performance: Java is faster than other traditional interpreted programming languages because Java bytecode is "close" to native code. It is still a little bit slower than a compiled language (e.g., C++). Java is an interpreted language that is why it is slower than compiled languages, e.g., C, C++, etc.

Multi-threaded: A thread is like a separate program, executing concurrently. We can write Java programs that deal with many tasks at once by defining multiple threads. The main advantage of multi-threading is that it doesn't occupy memory for each thread. It shares a common memory area. Threads are important for multi-media, Web applications, etc.

Distributed: Java is distributed because it facilitates users to create distributed applications in Java. RMI and EJB are used for creating distributed applications. This feature of Java makes us able to access files by calling the methods from any machine on the internet.

Dynamic: Java is a dynamic language. It supports dynamic loading of classes. It means classes are loaded on demand. It also supports functions from its native languages, i.e., C and C++.Java supports dynamic compilation and automatic memory management (garbage collection).

JDK, JVM and JRE

JVM: JVM (Java Virtual Machine) is an abstract machine. It is called a virtual machine because it doesn't physically exist. It is a specification that provides a runtime environment in which Java bytecode can be executed. It can also run those programs which are written in other languages and compiled to Java bytecode. JVMs are available for many hardware and software platforms. JVM, JRE, and JDK are platform dependent because the configuration of each OS is different from each other. However, Java is platform independent. There are three notions of the JVM: specification, implementation, and instance. The JVM performs Loads code, Verifies code, Executes code, and Provides runtime environment.

JRE: JRE is an acronym for Java Runtime Environment. It is also written as Java RTE. The Java Runtime Environment is a set of software tools which are used for developing Java applications. It is used to provide the runtime environment. It is the implementation of JVM. It physically exists. It contains a set of libraries + other files that JVM uses at runtime. The implementation of JVM is also actively released by other companies besides Sun Micro Systems.

JDK: JDK is an acronym for Java Development Kit. The Java Development Kit (JDK) is a software development environment which is used to develop Java applications and applets. It physically exists. It contains JRE + development tools.JDK is an implementation of any one of the below given Java Platforms released by Oracle Corporation: Standard Edition Java Platform, Enterprise Edition Java Platform, and Micro Edition Java Platform.

JDK Tools

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

Data Type

Integer: The int data type is a 32-bit signed two's complement integer. Its value-range lies between -2,147,483,648 (-2^31) to 2,147,483,647 (2^31 -1) (inclusive). Its minimum value is - 2,147,483,648and maximum value is 2,147,483,647. Its default value is 0.The int data type is generally used as a default data type for integral values unless if there is no problem about memory.
Example: int a = 100000, int b = -200000.

Float: The float data type is a single-precision 32-bit IEEE 754 floating point.Its value range is unlimited. It is recommended to use a float (instead of double) if you need to save memory in large arrays of floating point numbers. The float data type should never be used for precise values, such as currency. Its default value is 0.0F.
Example: float f1 = 234.5f

Character: The char data type is a single 16-bit Unicode character. Its value-range lies between '\u0000' (or 0) to '\uffff' (or 65,535 inclusive).The char data type is used to store characters.
Example: char letterA = 'A'

Boolean: The Boolean data type is used to store only two possible values: true and false. This data type is used for simple flags that track true/false conditions. The Boolean data type specifies one bit of information, but its "size" can't be defined precisely.
Example: Boolean one = false

Java Tokens

Keywords: Java keywords are also known as reserved words. Keywords are particular words which acts as a key to a code. These are predefined words by Java so it cannot be used as a variable or object name. For Example .. abstract, byte, break, Boolean etc.

Binary Literal: Java added a new feature Binary Literal in Java 7. I allow you to express integral types (byte, short, int, and long) in binary number system. To specify a binary literal, add the prefix 0b or 0B to the integral value.

Identifier: A valid identifier in java…

- Must begin with a letter (A to Z or a to z), currency character ($) or an underscore (_).
- Can have any combination of characters after the first character.
- Cannot be a keyword.

Comments: The Java comments are the statements that are not executed by the compiler and interpreter. The comments can be used to provide information or explanation about the variable, method, class or any statement. It can also be used to hide program code. Types of Java Comments There are three types of comments in Java.

1)Single Line Comment (//)
2)Multi Line Comment (/*….*/)
3)Documentation Comment (/**….*/)

Operators

Operator in Java is a symbol that is used to perform operations on variables and values. For example: +, -, *, / etc. There are many types of operators in Java which are given below:

Arithmetic Operator:

Operator	Description	Example
+ (Addition)	Adds values on either side of the operator.	A + B will give 30
- (Subtraction)	Subtracts right-hand operand from left-hand operand.	A - B will give -10
* (Multiplication)	Multiplies values on either side of the operator.	A * B will give 200
/ (Division)	Divides left-hand operand by right-hand operand.	B / A will give 2
% (Modulus)	Divides left-hand operand by right-hand operand and returns remainder.	B % A will give 0
++ (Increment)	Increases the value of operand by 1.	B++ gives 21
-- (Decrement)	Decreases the value of operand by 1.	B-- gives 19

Relational Operator:

Operator	Description	Example
== (equal to)	Checks if the values of two operands are equal or not, if yes then condition becomes true.	(A == B) is not true.
!= (not equal to)	Checks if the values of two operands are equal or not, if values are not equal then condition becomes true.	(A != B) is true.
> (greater than)	Checks if the value of left operand is greater than the value of right operand, if yes then condition becomes true.	(A > B) is not true.
< (less than)	Checks if the value of left operand is less than the value of right operand, if yes then condition becomes true.	(A < B) is true.
>= (greater than or equal to)	Checks if the value of left operand is greater than or equal to the value of right operand, if yes then condition becomes true.	(A >= B) is not true.
<= (less than or equal to)	Checks if the value of left operand is less than or equal to the value of right operand, if yes then condition becomes true.	(A <= B) is true.

Boolean Operator:

Operator	Description	Example
== (equal to)	Checks if the values of two operands are equal or not, if yes then condition becomes true.	(A == B) is not true.
!= (not equal to)	Checks if the values of two operands are equal or not, if values are not equal then condition becomes true.	(A != B) is true.
> (greater than)	Checks if the value of left operand is greater than the value of right operand, if yes then condition becomes true.	(A > B) is not true.
< (less than)	Checks if the value of left operand is less than the value of right operand, if yes then condition becomes true.	(A < B) is true.
>= (greater than or equal to)	Checks if the value of left operand is greater than or equal to the value of right operand, if yes then condition becomes true.	(A >= B) is not true.
<= (less than or equal to)	Checks if the value of left operand is less than or equal to the value of right operand, if yes then condition becomes true.	(A <= B) is true.
&& (logical and)	Called Logical AND operator. If both the operands are non-zero, then the condition becomes true.	(A && B) is false
|| (logical or)	Called Logical OR Operator. If any of the two operands are non-zero, then the condition becomes true.	(A || B) is true
! (logical not)	Called Logical NOT Operator. Use to reverses the logical state of its operand. If a condition is true then Logical NOT operator will make false.	!(A && B) is true

Logical Operator:

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

Bitwise Operator:

Operator	Description	Example
& (bitwise and)	Binary AND Operator copies a bit to the result if it exists in both operands.	(A & B) will give 12 which is 0000 1100
| (bitwise or)	Binary OR Operator copies a bit if it exists in either operand.	(A | B) will give 61 which is 0011 1101
^ (bitwise XOR)	Binary XOR Operator copies the bit if it is set in one operand but not both.	(A ^ B) will give 49 which is 0011 0001
~ (bitwise compliment)	Binary Ones Complement Operator is unary and has the effect of 'flipping' bits.	(~A) will give -61 which is 1100 0011 in 2's complement form due to a signed binary number.

<< (left shift)	Binary Left Shift Operator. The left operands value is moved left by the number of bits specified by the right operand.	A << 2 will give 240 which is 1111 0000
>> (right shift)	Binary Right Shift Operator. The left operands value is moved right by the number of bits specified by the right operand.	A >> 2 will give 15 which is 1111
>>> (zero fill right shift)	Shift right zero fill operator. The left operands value is moved right by the number of bits specified by the right operand and shifted values are filled up with zeros.	A >>>2 will give 15 which is 0000 1111

Assignment Operator:

Operator	Description	Example
=	Simple assignment operator. Assigns values from right side operands to left side operand.	C = A + B will assign value of A + B into C
+=	Add AND assignment operator. It adds right operand to the left operand and assign the result to left operand.	C += A is equivalent to C = C + A
-=	Subtract AND assignment operator. It subtracts right operand from the left operand and assign the result to left operand.	C -= A is equivalent to C = C – A
*=	Multiply AND assignment operator. It multiplies right operand with the left operand and assign the result to left operand.	C *= A is equivalent to C = C * A
/=	Divide AND assignment operator. It divides left operand with the right operand and assign the result to left operand.	C /= A is equivalent to C = C / A
%=	Modulus AND assignment operator. It takes modulus using two operands and assign the result to left operand.	C %= A is equivalent to C = C % A
<<=	Left shift AND assignment operator.	C <<= 2 is same as C = C << 2
>>=	Right shift AND assignment operator.	C >>= 2 is same as C = C >> 2
&=	Bitwise AND assignment operator.	C &= 2 is same as C = C & 2
^=	bitwise exclusive OR and assignment operator.	C ^= 2 is same as C = C ^ 2
|=	bitwise inclusive OR and assignment operator.	C |= 2 is same as C = C | 2

Unary Operator:

Operator	Description	Example
~ (bitwise compliment)	Binary One's Complement Operator is unary and has the effect of 'flipping' bits.	(~A) will give -61 which is 1100 0011 in 2's complement form due to a signed binary number.
++ (Increment)	Increases the value of operand by 1.	B++ gives 21
-- (Decrement)	Decreases the value of operand by 1.	B-- gives 19

Type Casting

A Type casting in Java is used to convert objects or variables of one type into another. When we are converting or assigning one data type to another they might not compatible. If it is suitable then it will do smoothly otherwise chances of data loss. Java Type Casting is classified into two types. Widening Casting (Implicit) – Automatic Type Conversion, and Narrowing Casting (Explicit) – Need Explicit Conversion.

Decision Statements

IF 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 like simple if, if…else, if…else if.. (Leader iif), and Nest if. The following is the simple syntax:

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

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. The following is 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;
}
```

Looping Statements

For Loop: The Java for loop is used to iterate a part of the program several times. If the number of iterations is fixed, it is recommended to use for loop. The following is the syntax:

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

While Loop: The Java while loop is used to iterate a part of the program several times. If the number of iterations is not fixed, it is recommended to use while loop. The following is the syntax:

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

Do…While Loop: The Java do-while loop is used to iterate a part of the program several times. If the number of iterations is not fixed and you must have to execute the loop at least once, it is recommended to use do-while loop. The Java do-while loop is executed at least once because condition is checked after loop body. The following is the syntax:

```
do {
```

```
        //code to be executed
} while(condition);
```

Jumping Statements

Break: 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 statement 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. The Syntax:

```
break;
```

Continue: 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. The Syntax:

```
continue;
```

Return: Java return keyword is used to complete the execution of a method. The return followed by the appropriate value that is returned to the caller. This value depends on the method return type like int method always return an integer value. It is used to exit from the method. It is not allowed to use return keyword in void method. The value passed with return keyword must match with return type of the method.

Array

Array is a collection of elements which share same name and same data type. Array element can be accessed by index number. Index number starts from 0 to length – one. Array can be of different types, which describes as follows.

One Dimensioal Array: A one-dimensional array is, essentially, a list of like-typed variables. To create an array, you first must create an array variable of the desired type. The general form of a one-dimensional array declaration is:

```
type var-name[ ];
```

Multidimensional Arrays: In Java, multidimensional arrays are actually array of arrays. These, as you might expect, look and act like regular multidimensional arrays. However, as you will see, there are a couple of subtle differences. To declare a multidimensional array variable, specify each additional index using another set of square brackets. For example, the following declares a two-dimensional array variable called twoD.

```
int twoD[][] = new int[4][5];
```

Jagged Array: Jagged array is a multidimensional array where member arrays are of different size. For example, we can create a 2D array where first array is of 3 elements, and is of 4 elements. Following is the example demonstrating the concept of jagged array.

```
public class Tester {
   public static void main(String[] args){
      int[][] twoDimenArray = new int[2][];
```

```
            //first row has 3 columns
            twoDimenArray[0] = new int[3];

            //second row has 4 columns
            twoDimenArray[1] = new int[4];

            int counter = 0;
            //initializing array
            for(int row=0; row < twoDimenArray.length; row++){
               for(int col=0; col < twoDimenArray[row].length; col++){
                  twoDimenArray[row][col] = counter++;
               }
            }

            //printing array
            for(int row=0; row < twoDimenArray.length; row++){
               System.out.println();
               for(int col=0; col < twoDimenArray[row].length; col++){
                  System.out.print(twoDimenArray[row][col] + " ");
               }
            }
         }
      }
```

Output:

```
0 1 2
3 4 5 6
```

Command Line Argument

The java command-line argument is an argument i.e. passed at the time of running the java program. The arguments passed from the console can be received in the java program and it can be used as an input. So, it provides a convenient way to check the behavior of the program for the different values. You can pass N (1,2,3 and so on) numbers of arguments from the command prompt.

OOPs Concept

Class: Collection of objects is called class. It is a logical entity. A class can also be defined as a blueprint from which you can create an individual object. Class doesn't consume any space.

Object: Any entity that has state and behavior is known as an object. For example, a chair, pen, table, keyboard, bike, etc. It can be physical or logical. An Object can be defined as an instance of a class. An object contains an address and takes up some space in memory. Objects can communicate without knowing the details of each other's data or code. The only necessary thing is the type of message accepted and the type of response returned by the objects.

Encapsulation: Binding (or wrapping) code and data together into a single unit are known as encapsulation. For example, a capsule, it is wrapped with different medicines. A java class is the example of encapsulation. Java bean is the fully encapsulated class because all the data members are private here.

Inheritance: When one object acquires all the properties and behaviors of a parent object, it is known as inheritance. It provides code reusability. It is used to achieve runtime polymorphism.

Polymorphism: If one task is performed in different ways, it is known as polymorphism. For example: to convince the customer differently, to draw something, for example, shape, triangle, rectangle, etc. In Java, we use method overloading and method overriding to achieve polymorphism. Another example can be to speak something; for example, a cat speaks meow, dog barks woof, etc.

Constructor: In Java, a constructor is a block of codes similar to the method. It is called when an instance of the class is created. At the time of calling constructor, memory for the object is allocated in the memory. It is a special type of method which is used to initialize the object. Every time an object is created using the new() keyword, at least one constructor is called. It calls a default constructor if there is no constructor available in the class. In such case, Java compiler provides a default constructor by default. There are two types of constructors in Java: no-arg constructor, and parameterized constructor. There are two types of constructors in Java Default constructor (no-arg constructor), and Parameterized constructor.

Static and Non-Static Members

Static Member: In Java, static members are those which belongs to the class and you can access these members without instantiating the class. The static keyword can be used with methods, fields, classes (inner/nested), blocks.

- **Static Methods:** You can create a static method by using the keyword static. Static methods can access only static fields, methods. To access static methods there is no need to instantiate the class, you can do it just using the class name as.
- **Static Variable:** If you declare any variable as static, it is known as a static variable. The static variable can be used to refer to the common property of all objects (which is not unique for each object), for example, the company name of employees, college name of students, etc. The static variable gets memory only once in the class area at the time of class loading.

Non-Static Members: Any variable of a class which is not static is called non-static variable/method or an instance variable.

Difference Between static and Non-Static Members

Key	Static	Non-Static
Access	A static variable can be accessed by static members as well as non-static member functions.	A non-static variable can not be accessed by static member functions.
Sharing	A static variable acts as a global variable and is shared among all the objects of the class.	A non-static variables are specific to instance object in which they are created.
Memory allocation	Static variables occupies less space and memory allocation happens once.	A non-static variable may occupy more space. Memory allocation may happen at run time.
Keyword	A static variable is declared using static keyword.	A normal variable is not required to have any special keyword.

Overloading

If a class has multiple methods having same name but different in parameters, it is known as Method Overloading. If we have to perform only one operation, having same name of the methods increases the readability of the program. Suppose you have to perform addition of the given numbers but there can be any

number of arguments, if you write the method such as a(int,int) for two parameters, and b(int,int,int) for three parameters then it may be difficult for you as well as other programmers to understand the behavior of the method because its name differs. So, we perform method overloading to figure out the program quickly. Advantage of method overloading is, it increases the readability of the program. There are two ways to overload the method in java by changing number of arguments, and by changing the data type.

Varargs

The varrags allows the method to accept zero or multiple arguments. Before varargs either we use overloaded method or take an array as the method parameter but it was not considered good because it leads to the maintenance problem. If we don't know how many arguments we will have to pass in the method, varargs is the better approach. Advantage of Varargs is that we don't have to provide overloaded methods so less code.

Syntax of varargs:

```
public static void fun(int ... a)
{
   // method body
}
```

This syntax tells the compiler that fun( ) can be called with zero or more arguments. As a result, here, a is implicitly declared as an array of type int[]. Below is a code snippet for illustrating the above concept :

```
// Java program to demonstrate varargs
class Test1 {
      static void fun(int... a)
      {
            System.out.println("Number of arguments: "+ a.length);
            for (int i : a)
                  System.out.print(i + " ");
                        System.out.println();
                  }
      public static void main(String args[])

            // one parameter
            fun(100);
            // four parameters
            fun(1, 2, 3, 4);
            // no parameter
            fun();
      }
}
```

Output:

```
Number of arguments: 1
100
Number of arguments: 4
1 2 3 4
Number of arguments: 0
```

Unit – 2

Inheritance, Java Packages

Universal Class (Object Class)

A class can be defined as a template/blueprint that describes the behavior/state that the object of its type support. A class is a blueprint from which individual objects are created. Object class is present in java.lang package. Every class in Java is directly or indirectly derived from the Object class. If a Class does not extend any other class then it is direct child class of Object and if extends other class then it is an indirectly derived. Therefore, the Object class methods are available to all Java classes. Hence Object class acts as a root of inheritance hierarchy in any Java Program. Object Class Some Methods for Example…

- public final Class getClass()
- public int hashCode()
- public boolean equals(Object obj)
- public String toString()
- public final void notify()

Access Specifiers

- **Default Access Modifier** - No Keyword: Default access modifier means we do not explicitly declare an access modifier for a class, field, method, etc. A variable or method declared without any access control modifier is available to any other class in the same package. The fields in an interface are implicitly public static final and the methods in an interface are by default public.
- **Private Access Modifier – Private:** Methods, variables, and constructors that are declared private can only be accessed within the declared class itself. Private access modifier is the most restrictive access level. Class and interfaces cannot be private. Variables that are declared private can be accessed outside the class, if public getter methods are present in the class. Using the private modifier is the main way that an object encapsulates itself and hides data from the outside world.
- **Public Access Modifier – Public:** A class, method, constructor, interface, etc. declared public can be accessed from any other class. Therefore, fields, methods, blocks declared inside a public class can be accessed from any class belonging to the Java Universe. However, if the public class we are trying to access is in a different package, then the public class still needs to be imported. Because of class inheritance, all public methods and variables of a class are inherited by its subclasses.
- **Protected Access Modifier – Protected:** Variables, methods, and constructors, which are declared protected in a superclass can be accessed only by the subclasses in other package or any class within the package of the protected members' class. The protected access modifier cannot be applied to class and interfaces. Methods, fields can be declared protected, however methods and fields in a interface cannot be declared protected. Protected access gives the subclass a chance to use the helper method or variable, while preventing a nonrelated class from trying to use it.

Inheritance

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?

1)For Method Overriding (so runtime polymorphism can be achieved).
2)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
}
```

Types of Inheritance:

- **Single Inheritance:** When a class inherits another class, it is known as a single inheritance. In the example given below, Dog class inherits the Animal class, so there is the single inheritance.
- **Multilevel Inheritance:** When there is a chain of inheritance, it is known as multilevel inheritance. As you can see in the example given below, BabyDog class inherits the Dog class which again inherits the Animal class, so there is a multilevel inheritance.
- **Hierarchical Inheritance:** When two or more classes inherits a single class, it is known as hierarchical inheritance. In the example given below, Dog and Cat classes inherits the Animal class, so there is hierarchical inheritance.

Note: Java does not support Multiple Inheritance.

Method Overriding: If subclass (child class) has the same method as declared in the parent class, it is known as method overriding in Java. In other words, If a subclass provides the specific implementation of the method that has been declared by one of its parent class, it is known as method overriding. The following are usage of Java Method Overriding:

- Method overriding is used to provide the specific implementation of a method which is already provided by its superclass.
- Method overriding is used for runtime polymorphism

Rules for Java Method Overriding:

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

Interface

An interface in Java is a blueprint of a class. It has static constants and abstract methods. The interface in Java is a mechanism to achieve abstraction. There can be only abstract methods in the Java interface, not method body. It is used to achieve abstraction and multiple inheritance in Java. In other words, you can say that interfaces can have abstract methods and variables. It cannot have a method body.

Java Interface also represents the IS-A relationship. It cannot be instantiated just like the abstract class. Since Java 8, we can have default and static methods in an interface. Since Java 9, we can have private methods in an interface. Why use Java interface? There are mainly three reasons to use interface. They are given below:

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

Syntax:

```
interface <interface_name> {
    // declare constant fields
    // declare methods that abstract by default.
}
```

Nested and Inner Class

Java Inner Classes: Java inner class or nested class is a class which is declared inside the class or interface. We use inner classes to logically group classes and interfaces in one place so that it can be more readable and

maintainable. Additionally, it can access all the members of outer class including private data members and methods.

Advantage of java inner classes

- Nested classes represent a special type of relationship that is it can access all the members (data members and methods) of outer class including private.
- Nested classes are used to develop more readable and maintainable code because it logically group classes and interfaces in one place only.
- Code Optimization: It requires less code to write.

Syntax of Inner class:

```
class Java_Outer_class{
      //code
      class Java_Inner_class{
           //code
      }
}
```

Nested Class: There are two types of nested classes non-static and static nested classes. The non-static nested classes are also known as inner classes. Non-static nested class (inner class):

1)Member inner class
2)Anonymous inner class
3)Local inner class

Static nested class

Type	Description
Member Inner Class	A class created within class and outside method.
Anonymous Inner Class	A class created for implementing interface or extending class. Its name is decided by the java compiler.
Local Inner Class	A class created within method.
Static Nested Class	A static class created within class.
Nested Interface	An interface created within class or interface.

Math Class

Java Math class provides several methods to work on math calculations like min(), max(), avg(), sin(), cos(), tan(), round(), ceil(), floor(), abs() etc. Unlike some of the StrictMath class numeric methods, all implementations of the equivalent function of Math class can't define to return the bit-for-bit same results. This relaxation permits implementation with better-performance where strict reproducibility is not required. If the size is int or long and the results overflow the range of value, the methods addExact(), subtractExact(), multiplyExact(), and toIntExact() throw an ArithmeticException. For other arithmetic operations like increment, decrement, divide, absolute value, and negation overflow occur only with a specific minimum or maximum value. It should be checked against the maximum and minimum value as appropriate.

Wrapper classes

The wrapper class in Java provides the mechanism to convert primitive into object and object into primitive. Since J2SE 5.0, autoboxing and unboxing feature convert primitives into objects and objects into primitives automatically. The automatic conversion of primitive into an object is known as autoboxing and vice-versa unboxing.

Java is an object-oriented programming language, so we need to deal with objects many times like in Collections, Serialization, Synchronization, etc. Let us see the different scenarios, where we need to use the wrapper classes.

- **Change the value in Method:** Java supports only call by value. So, if we pass a primitive value, it will not change the original value. But, if we convert the primitive value in an object, it will change the original value. Serialization: We need to convert the objects into streams to perform the serialization. If we have a primitive value, we can convert it in objects through the wrapper classes.
- **Synchronization:** Java synchronization works with objects in Multithreading.
- **java.util package:** The java.util package provides the utility classes to deal with objects.
- **Collection Framework:** Java collection framework works with objects only. All classes of the collection framework (ArrayList, LinkedList, Vector, HashSet, LinkedHashSet, TreeSet, PriorityQueue, ArrayDeque, etc.) deal with objects only.

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

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

Unit – 3

Exception Handling, Threading and Streams (Input and Output)

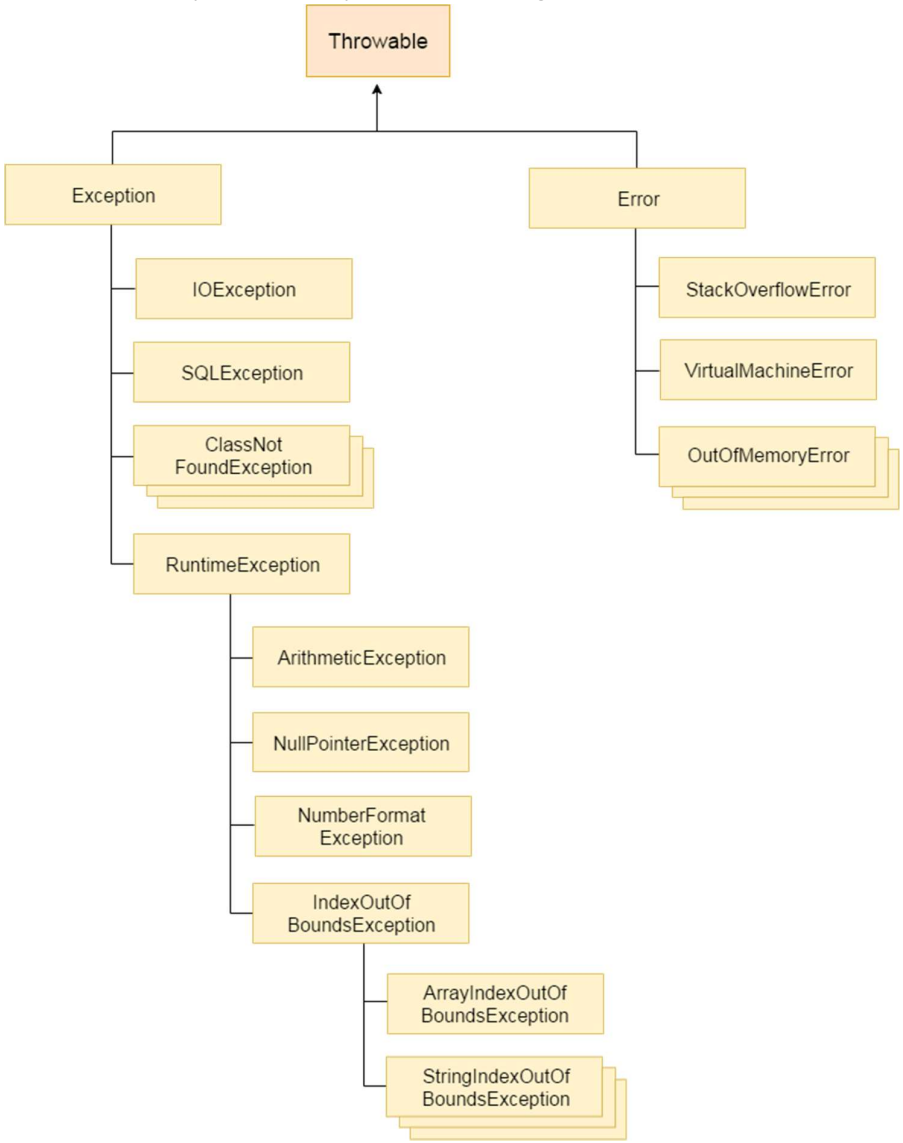
Introduction to Exception Handling

The Exception Handling in Java is one of the powerful mechanisms to handle the runtime errors so that normal flow of the application can be maintained. In this page, we will learn about Java exceptions, its type and the difference between checked and unchecked exceptions. What is Exception Handling? Exception Handling is a mechanism to handle runtime errors such as ClassNotFoundException, IOException, SQLException, RemoteException, etc.

What is Exception in Java? Dictionary Meaning: Exception is an abnormal condition. In Java, an exception is an event that disrupts the normal flow of the program. It is an object which is thrown at runtime. The Advantage of Exception Handling is to maintain the normal flow of the application. An exception normally disrupts the normal flow of the application that is why we use exception handling.

Hierarchy of Java Exception classes

The java.lang.Throwable class is the root class of Java Exception hierarchy which is inherited by two subclasses: Exception and Error. A hierarchy of Java Exception classes are given below:



Types of Java Exceptions

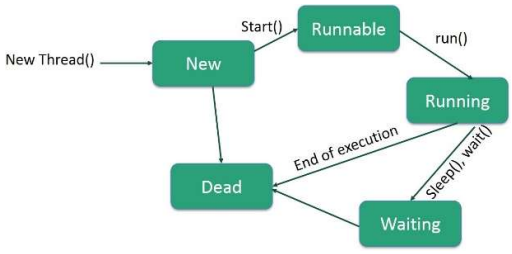
There are mainly two types of exceptions: checked and unchecked. Here, an error is considered as the unchecked exception. According to Oracle, there are three types of exceptions:

- **Checked Exception:** The classes which directly inherit Throwable class except RuntimeException and Error are known as checked exceptions e.g. IOException, SQLException etc. Checked exceptions are checked at compile-time.
- **Unchecked Exception:** The classes which inherit RuntimeException are known as unchecked exceptions e.g. ArithmeticException, NullPointerException, ArrayIndexOutOfBoundsException etc. Unchecked exceptions are not checked at compile-time, but they are checked at runtime.
- **Error:** Error is irrecoverable e.g. OutOfMemoryError, VirtualMachineError, AssertionError etc.

Try, Catch, Finally, Throw, Throws: There are 5 keywords which are used in handling exceptions in Java.

Keyword	Description
try	The "try" keyword is used to specify a block where we should place exception code. The try block must be followed by either catch or finally. It means, we can't use try block alone.
catch	The "catch" block is used to handle the exception. It must be preceded by try block which means we can't use catch block alone. It can be followed by finally block later.
finally	The "finally" block is used to execute the important code of the program. It is executed whether an exception is handled or not.
throw	The "throw" keyword is used to throw an exception.
throws	The "throws" keyword is used to declare exceptions. It doesn't throw an exception. It specifies that there may occur an exception in the method. It is always used with method signature.

Thread Life Cycle: A thread in Java at any point of time exists in any one of the following states. A thread lies only in one of the shown states at any instant:



- **New** − A new thread begins its life cycle in the new state. It remains in this state until the program starts the thread. It is also referred to as a born thread.
- **Runnable** − After a newly born thread is started, the thread becomes runnable. A thread in this state is considered to be executing its task.
- **Waiting** − Sometimes, a thread transitions to the waiting state while the thread waits for another thread to perform a task. A thread transitions back to the runnable state only when another thread signals the waiting thread to continue executing.
- **Timed Waiting** − A runnable thread can enter the timed waiting state for a specified interval of time. A thread in this state transitions back to the runnable state when that time interval expires or when the event it is waiting for occurs.
- **Terminated (Dead)** − A runnable thread enters the terminated state when it completes its task or otherwise terminates.

Thread Class and its methods

The second way to create a thread is to create a new class that extends Thread class using the following two simple steps. This approach provides more flexibility in handling multiple threads created using available methods in Thread class.

- You will need to override run( ) method available in Thread class. This method provides an entry point for the thread and you will put your complete business logic inside this method Syntax:

```
public void run( )
```

- Once Thread object is created, you can start it by calling start() method, which executes a call to run() method. Syantax

```
void start( );
```

Thread Methods

Sr.	Method & Description
1	**public void start()** Starts the thread in a separate path of execution, then invokes the run() method on this Thread object.
2	**public void run()** If this Thread object was instantiated using a separate Runnable target, the run() method is invoked on that Runnable object.
3	**public final void setName(String name)** Changes the name of the Thread object. There is also a getName() method for retrieving the name.
4	**public final void setPriority(int priority)** Sets the priority of this Thread object. The possible values are between 1 and 10.
5	**public final void setDaemon(boolean on)** A parameter of true denotes this Thread as a daemon thread.
6	**public final void join(long millisec)** The current thread invokes this method on a second thread, causing the current thread to block until the second thread terminates or the specified number of milliseconds passes.
7	**public void interrupt()** Interrupts this thread, causing it to continue execution if it was blocked for any reason.
8	**public final boolean isAlive()** Returns true if the thread is alive, which is any time after the thread has been started but before it runs to completion.

Thread Synchronization

When we start two or more threads within a program, there may be a situation when multiple threads try to access the same resource and finally they can produce unforeseen result due to concurrency issues. For example, if multiple threads try to write within a same file then they may corrupt the data because one of the threads can override data or while one thread is opening the same file at the same time another thread might be closing the same file. So, there is a need to synchronize the action of multiple threads and make sure that only one thread can access the resource at a given point in time. This is implemented using a concept called monitors. Each object in Java is associated with a monitor, which a thread can lock or unlock. Only one thread at a time may hold a lock on a monitor. Java programming language provides a very handy way of creating threads and synchronizing their task by using synchronized blocks. You keep shared resources within this block. Following is the general form of the synchronized statement −

```
synchronized(objectidentifier) {
```

```
      // Access shared variables and other shared resources
   }
```

Multithreading Example with Synchronization

```
   class PrintDemo {
      public void printCount() {
         try {
            for(int i = 5; i > 0; i--) {
               System.out.println("Counter   ---   "  + i );
            }
         } catch (Exception e) {
            System.out.println("Thread  interrupted.");
         }
      }
   }
   class ThreadDemo extends Thread {
      private Thread t;
      private String threadName;
      PrintDemo  PD;
      ThreadDemo( String name,  PrintDemo pd) {
         threadName = name;
         PD = pd;
      }
      public void run() {
         synchronized(PD) {
            PD.printCount();
         }
         System.out.println("Thread " +  threadName + " exiting.");
      }
      public void start () {
         System.out.println("Starting " +  threadName );
         if (t == null) {
            t = new Thread (this, threadName);
            t.start ();
         }
      }
   }
   public class TestThread {
      public static void main(String args[]) {
         PrintDemo PD = new PrintDemo();
         ThreadDemo T1 = new ThreadDemo( "Thread - 1 ", PD );
         ThreadDemo T2 = new ThreadDemo( "Thread - 2 ", PD );
         T1.start();
         T2.start();
       // wait for threads to end
         try {
            T1.join();
            T2.join();
         } catch ( Exception e) {
            System.out.println("Interrupted");
         }
      }
   }
```

Deamon Thread, Non-Deamon Thread

Deamon Thread: A Daemon thread is a background service thread which runs as a low priority thread and performs background operations like garbage collection. JVM exits if only daemon threads are remaining. The setDaemon() method of the Thread class is used to mark/set a particular thread as either a daemon thread or a user thread. The Java Virtual Machine exits when the only threads running are all daemon threads. This method must be called before the thread is started. Example:

```
class adminThread extends Thread {
   adminThread() {
      setDaemon(false);
   }
   public void run() {
      boolean d = isDaemon();
      System.out.println("daemon = " + d);
   }
}
public class Tester {
   public static void main(String[] args) throws Exception {
      Thread thread = new adminThread();
      System.out.println("thread = " + thread.currentThread());
      thread.setDaemon(false);
      thread.start();
   }
}
```

Output

```
thread = Thread[main,5,main]
daemon = false
```

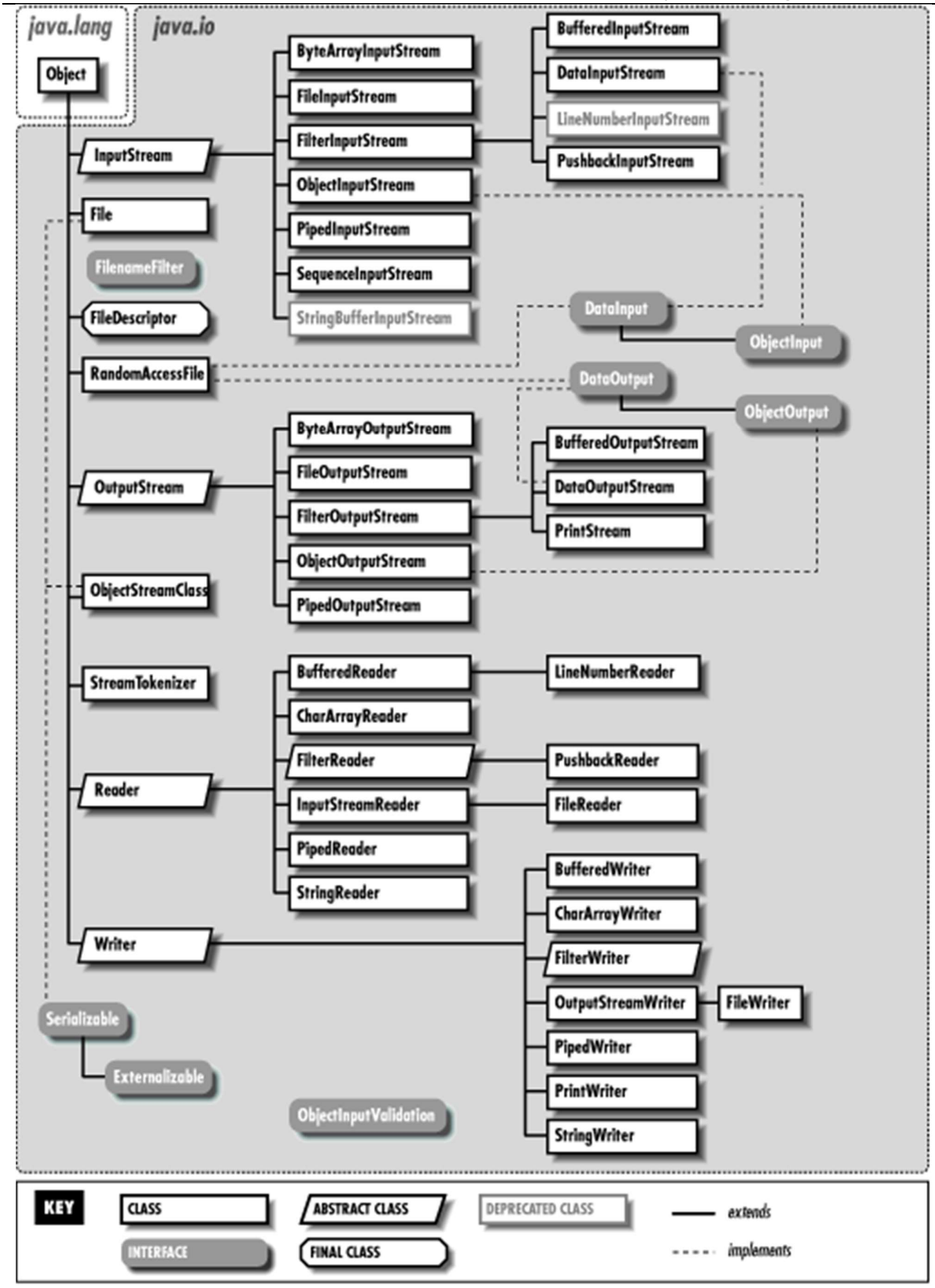
Non - Deamon Thread: The daemon threads are typically used to perform services for user threads. The main() method of the application thread is a user thread (non-daemon thread). The JVM doesn't terminate unless all the user thread (non-daemon) terminates. We can explicitly specify a thread created by a user thread to be a daemon thread by calling setDaemon(true). To determine if a thread is a daemon thread by using the method isDaemon().

Stream

A stream can be defined as a sequence of data. There are two kinds of Streams:

- **InPutStream** − The InputStream is used to read data from a source.
- **OutPutStream** − The OutputStream is used for writing data to a destination.

Java provides strong but flexible support for I/O related to files and networks with **java.io** package. The java.io package containing classes and interfaces related with I/O process. Basically there are two category of classes and interfaces: one is Byte Stream and another is Character Stream the following diagram describe hierarchy of available classes and interface in java.io pckage:



Character Streams: Java Byte streams are used to perform input and output of 8-bit bytes, whereas Java Character streams are used to perform input and output for 16-bit unicode. Though there are many classes related to character streams but the most frequently used classes are, FileReader and FileWriter. Though

internally FileReader uses FileInputStream and FileWriter uses FileOutputStream but here the major difference is that FileReader reads two bytes at a time and FileWriter writes two bytes at a time.

Byte Streams: Java byte streams are used to perform input and output of 8-bit bytes. Though there are many classes related to byte streams but the most frequently used classes are, FileInputStream and FileOutputStream.

RandomAccessFile Class: The Java.io.RandomAccessFile class file behaves like a large array of bytes stored in the file system.Instances of this class support both reading and writing to a random access file. Following is the declaration for Java.io.RandomAccessFile class −

```
public class RandomAccessFile
   extends Object
implements DataOutput, DataInput, Closeable
```

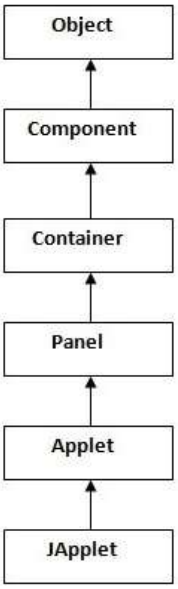
Stream Tokenizer Class: The Java.io.StreamTokenizer class takes an input stream and parses it into "tokens", allowing the tokens to be read one at a time. The stream tokenizer can recognize identifiers, numbers, quoted strings, and various comment styles. Following is the declaration for Java.io.StreamTokenizer class −

```
public class StreamTokenizer
   extends Object
```

Unit 4

Applets and Layout Managers

Java Applet



Applet is a special type of program that is embedded in the webpage to generate the dynamic content. It runs inside the browser and works at client side. There are many advantages of applet. They are as follows:

- It works at client side so less response time.
- Secured
- It can be executed by browsers running under many plateforms, including Linux, Windows, Mac Os etc.

Drawback of Applet is Plugin is required at client browser to execute applet. As displayed in the diagram, Applet class extends Panel. Panel class extends Container which is the subclass of Component.

Lifecycle of Java Applet

- Applet is initialized.
- Applet is started.
- Applet is painted.
- Applet is stopped.
- Applet is destroyed.

Lifecycle methods for Applet: The java.applet.Applet class 4 life cycle methods and java.awt.Component class provides 1 life cycle methods for an applet.

java.applet.Applet class: For creating any applet java.applet.Applet class must be inherited. It provides 4 life cycle methods of applet.

- public void init(): is used to initialized the Applet. It is invoked only once.
- public void start(): is invoked after the init() method or browser is maximized. It is used to start the Applet.
- public void stop(): is used to stop the Applet. It is invoked when Applet is stop or browser is minimized.
- public void destroy(): is used to destroy the Applet. It is invoked only once.

java.awt.Component class: The Component class provides 1 life cycle method of applet.

- public void paint(Graphics g): is used to paint the Applet. It provides Graphics class object that can be used for drawing oval, rectangle, arc etc.

To run an Apple: There are two ways to run an applet

- By html file.
- By appletViewer tool (for testing purpose).

Simple example of Applet by html file: To execute the applet by html file, create an applet and compile it. After that create an html file and place the applet code in html file. Now click the html file.

```
//First.java
import java.applet.Applet;
import java.awt.Graphics;
public class First extends Applet{
      public void paint(Graphics g){
            g.drawString("welcome",150,150);
      }

}
```

myapplet.html

```
<html>
      <body>
            <applet code="First.class" width="300" height="300">
            </applet>
      </body>
</html>
```

Simple example of Applet by appletviewer tool: To execute the applet by appletviewer tool, create an applet that contains applet tag in comment and compile it. After that run it by: appletviewer First.java. Now Html file is not required but it is for testing purpose only.

```
//First.java
import java.applet.Applet;
import java.awt.Graphics;
public class First extends Applet{
      public void paint(Graphics g){
            g.drawString("welcome to applet",150,150);
      }

}
/*
      <applet code="First.class" width="300" height="300">
      </applet>
*/
```

Displaying Graphics in Applet: java.awt.Graphics class provides many methods for graphics programming.Commonly used methods of Graphics class:

- public abstract void drawString(String str, int x, int y): is used to draw the specified string.
- public void drawRect(int x, int y, int width, int height): draws a rectangle with the specified width and height.
- public abstract void fillRect(int x, int y, int width, int height): is used to fill rectangle with the default color and specified width and height.
- public abstract void drawOval(int x, int y, int width, int height): is used to draw oval with the specified width and height.
- public abstract void fillOval(int x, int y, int width, int height): is used to fill oval with the default color and specified width and height.
- public abstract void drawLine(int x1, int y1, int x2, int y2): is used to draw line between the points(x1, y1) and (x2, y2).
- public abstract boolean drawImage(Image img, int x, int y, ImageObserver observer): is used draw the specified image.
- public abstract void drawArc(int x, int y, int width, int height, int startAngle, int arcAngle): is used draw a circular or elliptical arc.
- public abstract void fillArc(int x, int y, int width, int height, int startAngle, int arcAngle): is used to fill a circular or elliptical arc.
- public abstract void setColor(Color c): is used to set the graphics current color to the specified color.
- public abstract void setFont(Font font): is used to set the graphics current font to the specified font.

Displaying Image in Applet: Applet is mostly used in games and animation. For this purpose image is required to be displayed. The java.awt.Graphics class provide a method drawImage() to display the image. Syntax of drawImage() method:

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

How to get the object of Image: The java.applet.Applet class provides getImage() method that returns the object of Image. Syntax:

- public Image getImage(URL u, String image){}

Other required methods of Applet class to display image:

- public URL getDocumentBase(): is used to return the URL of the document in which applet is embedded.
- public URL getCodeBase(): is used to return the base URL.

Parameter in Applet: We can get any information from the HTML file as a parameter. For this purpose, Applet class provides a method named getParameter(). Syntax:

```
public String getParameter(String parameterName)
```

Java LayoutManagers

The LayoutManagers are used to arrange components in a particular manner. The Java LayoutManagers facilitates us to control the positioning and size of the components in GUI forms. LayoutManager is an interface that is implemented by all the classes of layout managers. There are the following classes that represent the layout managers:

1. java.awt.BorderLayout
2. java.awt.FlowLayout
3. java.awt.GridLayout
4. java.awt.CardLayout
5. java.awt.GridBagLayout
6. javax.swing.BoxLayout
7. javax.swing.GroupLayout
8. javax.swing.ScrollPaneLayout
9. javax.swing.SpringLayout

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…

```
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();
}
}
```

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

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

1)public void next(Container parent): is used to flip to the next card of the given container.
2)public void previous(Container parent): is used to flip to the previous card of the given container.
3)public void first(Container parent): is used to flip to the first card of the given container.
4)public void last(Container parent): is used to flip to the last card of the given container.
5)public void show(Container parent, String name): is used to flip to the specified card with the given name.

Example..

```
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);
    }
}
```

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

1)GridLayout(): creates a grid layout with one column per component in a row.

2)GridLayout(int rows, int columns): creates a grid layout with the given rows and columns but no gaps between the components.

3)GridLayout(int rows, int columns, int hgap, int vgap): creates a grid layout with the given rows and columns alongwith given horizontal and vertical gaps.

Example..

```
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 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	PREFERREDSIZE	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);

            }
    }
```

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

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

Fields of BoxLayout class

1)public static final int X_AXIS

2)public static final int Y_AXIS

3)public static final int LINE_AXIS

4)public static final int PAGE_AXIS

Constructor of BoxLayout class:

`BoxLayout(Container c, int axis)`: creates a box layout that arranges the components with the given axis. The following is the Exampe:

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

    public class BoxLayoutExample1 extends Frame {
     Button buttons[];

     public BoxLayoutExample1 () {
       buttons = new Button [5];

       for (int i = 0;i<5;i++) {
          buttons[i] = new Button ("Button " + (i + 1));
          add (buttons[i]);
        }

    setLayout (new BoxLayout (this, BoxLayout.Y_AXIS));
    setSize(400,400);
    setVisible(true);
    }
```

```
public static void main(String args[]){
BoxLayoutExample1 b=new BoxLayoutExample1();
}
}
```

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 constrains 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. Constraints, 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.Constraints	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 calculates the maximum size dimensions for the specified container, given the components it contains.
Dimension	minimumLayoutSize(Container parent)	It is used to calculates the minimum size dimensions for the specified container, given the components it contains.
Dimension	preferredLayoutSize(Container parent)	It is used to calculates 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();
                }
            });
        }
    }
```

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 exists in both a horizontal and vertical group, otherwise an IllegalStateException is thrown during layout, or when the minimum, preferred or maximum size is requested.

Constructors

GroupLayout(Container host)	It creates a GroupLayout for the specified Container.

Fields

Modifier and Type	Field	Description
static int	DEFAULT_SIZE	It indicates the size from the component or gap should be used for a particular range value.
static int	PREFERRED_SIZE	It indicates the preferred size from the component or gap should be used for a particular range value.

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 visiblity 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);
```

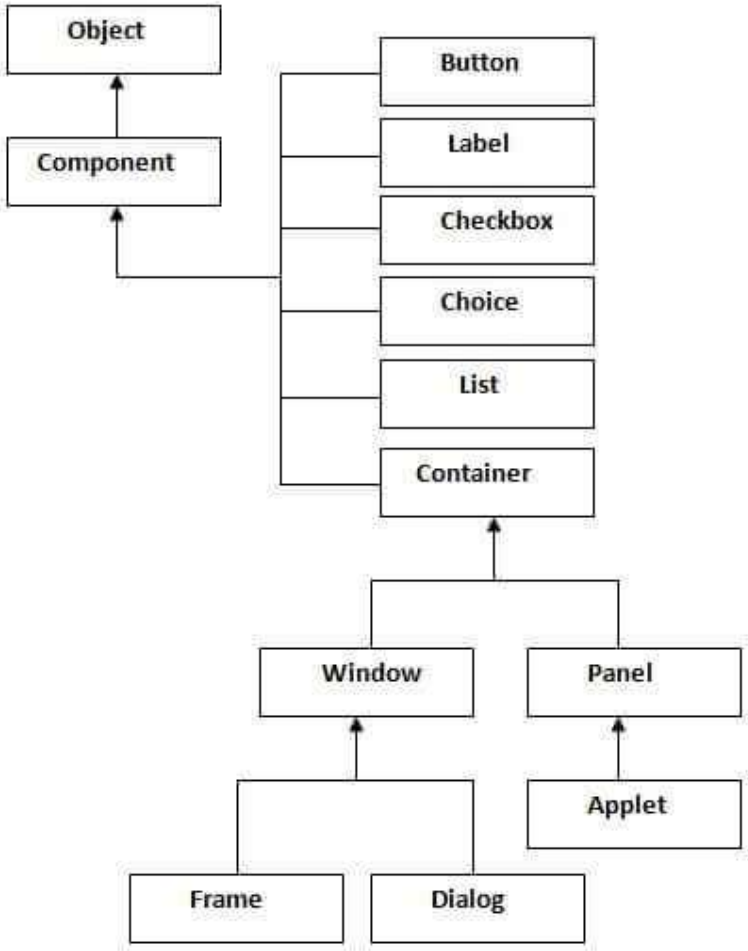
```
        JLabel clickMe = new JLabel("Click Here");
        JButton button = new JButton("This Button");

        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);
    }
}
```

Unit 5

GUI using Swing
Event Handling

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. 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.

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

Java AWT	Java Swing
AWT components are **platform-dependent**.	Java swing components are **platform-independent**.
AWT components are **heavyweight**.	Swing components are **lightweight**.
AWT **doesn't support pluggable look and feel**.	Swing **supports pluggable look and feel**.
AWT provides **less components** than Swing.	Swing provides **more powerful components** such as tables, lists, scrollpanes, colorchooser, tabbedpane etc.
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**.

Swing Components

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. Constructors:

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

Useful Methods

Modifier and Type	Method	Description
protected void	addImpl(Component comp, Object constraints, int index)	Adds the specified child Component.
protected JRootPane	createRootPane()	Called by the constructor methods to create the default rootPane.
protected void	frameInit()	Called by the constructors to init the JFrame properly.

void	setContentPane(Containe contentPane)	It sets the contentPane property
static void	setDefaultLookAndFeelDecorated(boolean defaultLookAndFeelDecorated)	Provides a hint as to whether or not newly created JFrames should have their Window decorations (such as borders, widgets to close the window, title...) provided by the current look and feel.
void	setIconImage(Image image)	It sets the image to be displayed as the icon for this window.
void	setJMenuBar(JMenuBar menubar)	It sets the menubar for this frame.
void	setLayeredPane(JLayeredPane layeredPane)	It sets the layeredPane property.
JRootPane	getRootPane()	It returns the rootPane object for this frame.
TransferHandler	getTransferHandler()	It gets the transferHandler property.

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
```

Commonly used Constructors:

Constructor	Description
JPanel()	It is used to create a new JPanel with a double buffer and a flow layout.
JPanel(boolean isDoubleBuffered)	It is used to create a new JPanel with FlowLayout and the specified buffering strategy.
JPanel(LayoutManager layout)	It is used to create a new JPanel with the specified layout manager.

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

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

Commonly used Constructors:

Constructor	Description
JLabel()	Creates a JLabel instance with no image and with an empty string for the title.
JLabel(String s)	Creates a JLabel instance with the specified text.
JLabel(Icon i)	Creates a JLabel instance with the specified image.
JLabel(String s, Icon i, int horizontalAlignment)	Creates a JLabel instance with the specified text, image, and horizontal alignment.

Commonly used Methods:

Methods	Description
String getText()	t returns the text string that a label display.
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.

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

```
public class JButton extends AbstractButton implements Accessible
```

Commonly used Constructors:

Constructor	Description
JButton()	It creates a button with no text and icon.
JButton(String s)	It creates a button with the specified text.
JButton(Icon i)	It creates a button with the specified icon object.

Commonly used Methods of AbstractButton class:

Methods	Description
void setText(String s)	It is used to set specified text on button
String getText()	It is used to return the text of the button.
void setEnabled(boolean b)	It is used to enable or disable the button.
void setIcon(Icon b)	It is used to set the specified Icon on the button.
Icon getIcon()	It is used to get the Icon of the button.
void setMnemonic(int a)	It is used to set the mnemonic on the button.
void addActionListener(ActionListener a)	It is used to add the action listener to this object.

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

```
public class JRadioButton extends JToggleButton implements Accessible
```

Commonly used Constructors:

Constructor	Description
JRadioButton()	Creates an unselected radio button with no text.
JRadioButton(String s)	Creates an unselected radio button with specified text.
JRadioButton(String s, boolean selected)	Creates a radio button with the specified text and selected status.

Commonly used Methods:

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.

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

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

Commonly used Constructors:

Constructor	Description
JCheckBox()	Creates an initially unselected check box button with no text, no icon.
JChechBox(String s)	Creates an initially unselected check box with text.
JCheckBox(String text, boolean selected)	Creates a check box with text and specifies whether or not it is initially selected.
JCheckBox(Action a)	Creates a check box where properties are taken from the Action supplied.

Commonly used Methods:

Methods	Description
AccessibleContext getAccessibleContext()	It is used to get the AccessibleContext associated with this JCheckBox.
protected String paramString()	It returns a string representation of this JCheckBox.

JProgressBar: The JProgressBar class is used to display the progress of the task. It inherits JComponent class. Let's see the declaration for javax.swing.JProgressBar class.

```
public class JProgressBar extends JComponent implements SwingConstants, Accessible
```

Commonly used Constructors:

Constructor	Description
JProgressBar()	It is used to create a horizontal progress bar but no string text.
JProgressBar(int min, int max)	It is used to create a horizontal progress bar with the specified minimum and maximum value.
JProgressBar(int orient)	It is used to create a progress bar with the specified orientation, it can be either Vertical or Horizontal by using SwingConstants.VERTICAL and SwingConstants.HORIZONTAL constants.
JProgressBar(int orient, int min, int max)	It is used to create a progress bar with the specified orientation, minimum and maximum value.

Commonly used Methods:

Method	Description
void setStringPainted(boolean b)	It is used to determine whether string should be displayed.
void setString(String s)	It is used to set value to the progress string.
void setOrientation(int orientation)	It is used to set the orientation, it may be either vertical or horizontal by using SwingConstants.VERTICAL and SwingConstants.HORIZONTAL constants.
void setValue(int value)	It is used to set the current value on the progress bar.

JFileChooser: The object of JFileChooser class represents a dialog window from which the user can select file. It inherits JComponent class. Let's see the declaration for javax.swing.JFileChooser class.

```
public class JFileChooser extends JComponent implements Accessible
```

Commonly used Constructors:

Constructor	Description
JFileChooser()	Constructs a JFileChooser pointing to the user's default directory.
JFileChooser(File currentDirectory)	Constructs a JFileChooser using the given File as the path.
JFileChooser(String currentDirectoryPath)	Constructs a JFileChooser using the given path.

JTextField: The object of a JTextField class is a text component that allows the editing of a single line text. It inherits JTextComponent class. Let's see the declaration for javax.swing.JTextField class.

```
public class JTextField extends JTextComponent implements SwingConstants
```

Commonly used Constructors:

Constructor	Description
JTextField()	Creates a new TextField
JTextField(String text)	Creates a new TextField initialized with the specified text.
JTextField(String text, int columns)	Creates a new TextField initialized with the specified text and columns.
JTextField(int columns)	Creates a new empty TextField with the specified number of columns.

Commonly used Methods:

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

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

```
public class JPasswordField extends JTextField
```

Commonly used Constructors:

Constructor	Description
JPasswordField()	Constructs a new JPasswordField, with a default document, null starting text string, and 0 column width.
JPasswordField(int columns)	Constructs a new empty JPasswordField with the specified number of columns.
JPasswordField(String text)	Constructs a new JPasswordField initialized with the specified text.
JPasswordField(String text, int columns)	Construct a new JPasswordField initialized with the specified text and columns.

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

```
public class JTextArea extends JTextComponent
```

Commonly used Constructors:

Constructor	Description
JTextArea()	Creates a text area that displays no text initially.
JTextArea(String s)	Creates a text area that displays specified text initially.
JTextArea(int row, int column)	Creates a text area with the specified number of rows and columns that displays no text initially.

JTextArea(String s, int row, int column)	Creates a text area with the specified number of rows and columns that displays specified text.

Commonly used Methods:

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.

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

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

Commonly used Constructors:

Constructor	Description
JScrollBar()	Creates a vertical scrollbar with the initial values.
JScrollBar(int orientation)	Creates a scrollbar with the specified orientation and the initial values.
JScrollBar(int orientation, int value, int extent, int min, int max)	Creates a scrollbar with the specified orientation, value, extent, minimum, and maximum.

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

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

Commonly used Constructors:

Constructor	Description
JComboBox()	Creates a JComboBox with a default data model.
JComboBox(Object[] items)	Creates a JComboBox that contains the elements in the specified array.
JComboBox(Vector<?> items)	Creates a JComboBox that contains the elements in the specified Vector.

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.

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

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

Commonly used Constructors:

Constructor	Description
JList()	Creates a JList with an empty, read-only, model.
JList(ary[] listData)	Creates a JList that displays the elements in the specified array.
JList(ListModel<ary> dataModel)	Creates a JList that displays elements from the specified, non-null, model.

Commonly used Methods:

Methods	Description
Void addListSelectionListener(ListSelectionListener listener)	It is used to add a listener to the list, to be notified each time a change to the selection occurs.
int getSelectedIndex()	It is used to return the smallest selected cell index.
ListModel getModel()	It is used to return the data model that holds a list of items displayed by the JList component.
void setListData(Object[] listData)	It is used to create a read-only ListModel from an array of objects.

Menus (JMenuBar, JMenu, 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
```

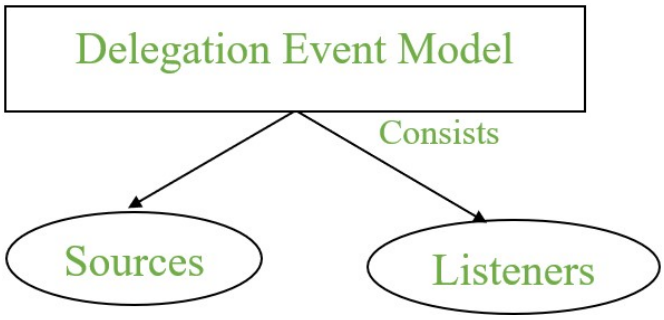
JMenuItem class declaration

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

Introduction to Event Handling

Event Handling is the mechanism that controls the event and decides what should happen if an event occurs. This mechanism has the code which is known as event handler that is executed when an event occurs. Java Uses the Delegation Event Model to handle the events. This model defines the standard mechanism to generate and handle the events.

An event can be defined as changing the state of an object or behavior by performing actions. Actions can be a button click, cursor movement, keypress through keyboard or page scrolling, etc. The java.awt.event package can be used to provide various event classes. It is a mechanism to control the events and to decide what should happen after an event occur. To handle the events, Java follows the Delegation Event model.



Delegation Event model: It has Sources and Listeners. **Source:** Events are generated from the source. There are various sources like buttons, checkboxes, list, menu-item, choice, scrollbar, text components, windows, etc., to generate events. **Listeners:** Listeners are used for handling the events generated from the source. Each of these listeners represents interfaces that are responsible for handling events. To perform Event Handling, we need to register the source with the listener. Different Classes provide different registration methods. The following is generic Syntax:

```
addTypeListener()
```

where Type represents the type of event.

Event Classes and Interfaces in Java

Event Class	Listener Interface	Description
ActionEvent	ActionListener	An event that indicates that a component-defined action occurred like a button click or selecting an item from the menu-item list.
AdjustmentEvent	AdjustmentListener	The adjustment event is emitted by an Adjustable object like Scrollbar.

Event Class	Listener Interface	Description
ComponentEvent	ComponentListener	An event that indicates that a component moved, the size changed or changed its visibility.
ContainerEvent	ContainerListener	When a component is added to a container (or) removed from it, then this event is generated by a container object.
FocusEvent	FocusListener	These are focus-related events, which include focus, focusin, focusout, and blur.
ItemEvent	ItemListener	An event that indicates whether an item was selected or not.
KeyEvent	KeyListener	An event that occurs due to a sequence of keypresses on the keyboard.
MouseEvent	MouseListener & MouseMotionListener	The events that occur due to the user interaction with the mouse (Pointing Device).
MouseWheelEvent	MouseWheelListener	An event that specifies that the mouse wheel was rotated in a component.
TextEvent	TextListener	An event that occurs when an object's text changes.
WindowEvent	WindowListener	An event which indicates whether a window has changed its status or not.

Different interfaces consists of different methods which are specified below.

Listener Interface	Methods
ActionListener	• actionPerformed()
AdjustmentListener	• adjustmentValueChanged()
ComponentListener	• componentResized() • componentShown()

Listener Interface	Methods
	• componentMoved() • componentHidden()
ContainerListener	• componentAdded() • componentRemoved()
FocusListener	• focusGained() • focusLost()
ItemListener	• itemStateChanged()
KeyListener	• keyTyped() • keyPressed() • keyReleased()
MouseListener	• mousePressed() • mouseClicked() • mouseEntered() • mouseExited() • mouseReleased()
MouseMotionListener	• mouseMoved() • mouseDragged()
MouseWheelListener	• mouseWheelMoved()
TextListener	• textChanged()
WindowListener	• windowActivated() • windowDeactivated() • windowOpened() • windowClosed() • windowClosing() • windowIconified() • windowDeiconified()

Flow of Event Handling

1. User Interaction with a component is required to generate an event.
2. The object of the respective event class is created automatically after event generation, and it holds all information of the event source.
3. The newly created object is passed to the methods of the registered listener.

4. The method executes and returns the result.

Code-Approaches: The three approaches for performing event handling are by placing the event handling code in one of the below-specified places.

- Within Class
- Other Class
- Anonymous Class

- **Within Class**

```
1  // Java program to demonstrate the
2  // event handling within the class
3
4  import java.awt.*;
5  import java.awt.event.*;
6
7  class EventHandling1 extends Frame implements ActionListener {
8
9       TextField textField;
10
11      EventHandling1()
12      {
13           // Component Creation
14           textField = new TextField();
15
16           // setBounds method is used to provide
17           // position and size of the component
18           textField.setBounds(60, 50, 180, 25);
19           Button button = new Button("click Here");
20           button.setBounds(100, 120, 80, 30);
21
22           // Registering component with listener
23           // this refers to current instance
24           button.addActionListener(this);
25
26           // add Components
27           add(textField);
28           add(button);
29
30           // set visibility
31           setVisible(true);
32      }
33
34      // implementing method of actionListener
35      public void actionPerformed(ActionEvent e)
36      {
37           // Setting text to field
38           textField.setText("Java Events");
39      }
40
41      public static void main(String[] args)
42      {
43      new EventHandling1();
44      }
45 }
```

Explanation

- Firstly extend the class with the applet and implement the respective listener.
- Create Text-Field and Button components.

- Registered the button component with respective event. i.e. ActionEvent by addActionListener().
- In the end, implement the abstract method.

- **Anonymous Class**

```
// Java program to demonstrate the
// event handling by the anonymous class

import java.awt.*;
import java.awt.event.*;

class EventHandling2 extends Frame {

	TextField textField;

	EventHandling2()
	{
		// Component Creation
		textField = new TextField();

		// setBounds method is used to provide
		// position and size of component
		textField.setBounds(60, 50, 180, 25);
		Button button = new Button("click Here");
		button.setBounds(100, 120, 80, 30);

		// Registering component with listener anonymously
		button.addActionListener(new ActionListener() {
			public void actionPerformed(ActionEvent e)
			{
				// Setting text to field
				textField.setText("Anonymous");
			}
		});

		// add Components
		add(textField);
		add(button);

		// set visibility
		setVisible(true);
	}

	public static void main(String[] args)
	{
		new EventHandling2();
	}
}
```