OBJECT ORIENTED PROGRAMME USING JAVA

MODULE 1: INTRODUCTION

Approaches to Software Design- Functional Oriented Design, Object Oriented Design, Case Study of Automated Fire Alarm System.

Software design is a process to conceptualize the software requirements into software implementation. Software design takes the user requirements as challenges and tries to find optimum solution.

In function-oriented design, the system is comprised of many smaller sub-systems known as functions. These functions are capable of performing significant task in the system. The system is considered as top view of all functions. This design mechanism divides the whole system into smaller functions, which provides means of abstraction by concealing the information and their operation. These functional modules can share information among themselves by means of information passing and using information available globally. Another characteristic of functions is that when a program calls a function, the function changes the state of the program, which sometimes is not acceptable by other modules. Function oriented design works well where the system state does not matter and program/functions work on input rather than on a state.

Design Process

- The whole system is seen as how data flows in the system by means of data flow diagram.
- DFD depicts how functions changes data and state of entire system.
- The entire system is logically broken down into smaller units known as functions on the basis of their operation in the system.
- Each function is then described at large.

Object oriented design works around the entities and their characteristics instead of functions involved in the software system. This design strategies focuses on entities and its characteristics.

Design Process

- A solution design is created from requirement or previous used system and/or system sequence diagram.
- Objects are identified and grouped into classes on behalf of similarity in attribute characteristics.
- Class hierarchy and relation among them is defined.
- Application framework is defined.

Software Design Approaches

- **Top Down Design -** We know that a system is composed of more than one sub-systems and it contains a number of components. Further, these sub-systems and components may have their own set of sub-system and components and creates hierarchical structure in the system. Top-down design takes the whole software system as one entity and then decomposes it to achieve more than one sub-system or component based on some characteristics. Each sub-system or component is then treated as a system and decomposed further. This process keeps on running until the lowest level of system in the top-down hierarchy is achieved. Top-down design is more suitable when the software solution needs to be designed from scratch and specific details are unknown.
- **Bottom-up Design** - The bottom up design model starts with most specific and basic components. It proceeds with composing higher level of components by using basic or lower level components. It keeps creating higher level components until the desired system is not evolved as one single component. With each higher level, the amount of abstraction is increased. Bottom-up strategy is more suitable when a system needs to be created from some existing system, where the basic primitives can be used in the newer system.

Object Modeling Using UML – Basic object oriented concepts

UML stands for Unified Modeling Language. UML is a standard language for specifying, visualizing, constructing, and documenting the artifacts of software systems. UML is not a programming language but tools can be used to generate code in various languages using UML diagrams. UML has a direct relation with object oriented analysis and design. UML can be described as the successor of object-oriented (OO) analysis and design. UML is powerful enough to represent all the concepts that exist in object-oriented analysis and design.

Basic object oriented concepts

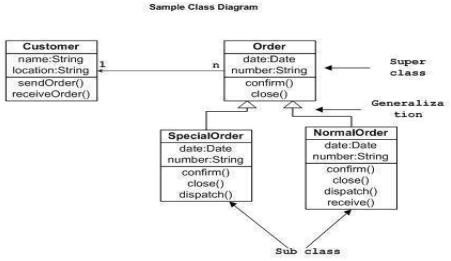
- **Inheritance -** Inheritance can be defined as the process where one (parent/super) class acquires the properties (methods and fields) of another (child/sub). With the use of inheritance, the information is made manageable in a hierarchical order.
- **Polymorphism -** Polymorphism is the ability of an object to perform different actions (or, exhibit different behaviors) based on the context.
- **Abstraction -** Abstraction is a process of hiding the implementation details from the user, only the functionality will be provided to the user. In other words, the user will have the information on what the object does instead of how it does it. In Java, abstraction is achieved using Abstract classes and interfaces.
- **Encapsulation -** Encapsulation in Java is a mechanism for wrapping the data (variables) and code acting on the data (methods) together as a single unit. In encapsulation, the variables of a class will be hidden from other classes and can be accessed only through the methods of their current class. Therefore, it is also known as **data hiding**. To achieve encapsulation in Java –
  - Declare the variables of a class as private.
  - Provide public setter and getter methods to modify and view the variables values.

access modifiers

UML diagrams - Use case models, Class diagram, Interaction diagram , Activity diagram, State chart diagram

The elements are like components which can be associated in different ways to make a complete UML picture, which is known as diagram. There are two broad categories of diagrams and they are again divided into subcategories – Structural Diagrams, Behavioral Diagrams. The structural diagrams represent the static aspect of the system.eg:- Class diagram.

- **Class diagram -** Class diagram consists of classes, interfaces, associations, and collaboration. Class diagrams basically represent the object-oriented view of a system, which is static in nature. Active class is used in a class diagram to represent the concurrency of the system. It is generally used for development purpose.



Behavioral diagrams basically capture the dynamic aspect of a system. Eg:- Use case diagram, Statechart diagram, Activity diagram, Interaction diagram.

- **Use case diagram -** Use case diagrams are a set of use cases, actors, and their relationships. They represent the use case view of a system. A use case represents a particular functionality of a system. Hence, use case diagram is used to describe the relationships among the functionalities and their internal/external controllers. These controllers are known as actors.
- **Statechart diagram -** Any real-time system is expected to be reacted by some kind of internal/external events. These events are responsible for state change of the system. Statechart diagram is used to represent the event driven state change of a system. It basically describes the state change of a class, interface, etc. State chart diagram is used to visualize the reaction of a system by internal/external factors.
- **Activity diagram -** Activity diagram describes the flow of control in a system. It consists of activities and links. The flow can be sequential, concurrent, or branched. Activities are nothing but the functions of a system. Numbers of activity diagrams are prepared to capture the entire flow in a system. Activity diagrams are used to visualize the flow of controls in a system. This is prepared to have an idea of how the system will work when executed.
- **Interaction diagram -** This interactive behavior is represented in UML by two diagrams known as **Sequence diagram** and **Collaboration diagram**. The purpose of interaction diagrams is to visualize the interactive behavior of the system. Sequence Diagram - The diagram deals with some sequences, which are the sequence of messages flowing from one object to another.
  Collaboration Diagram - It represents the structural organization of a system and the messages sent/received.

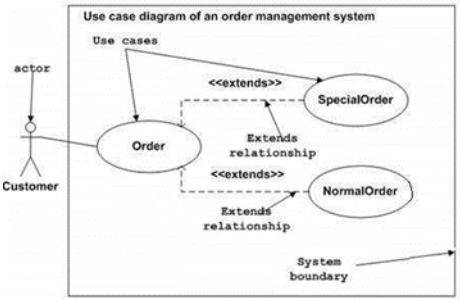
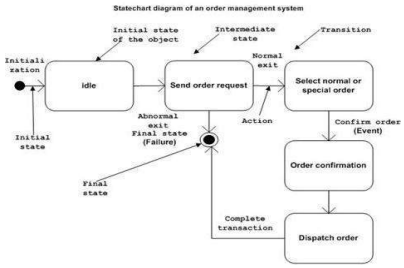
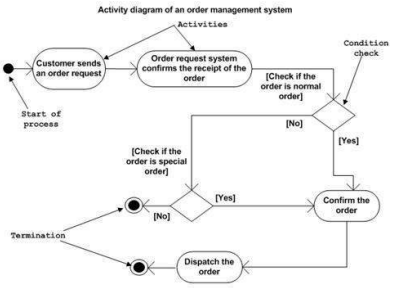
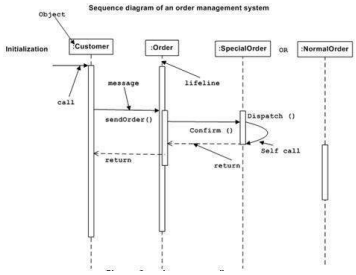
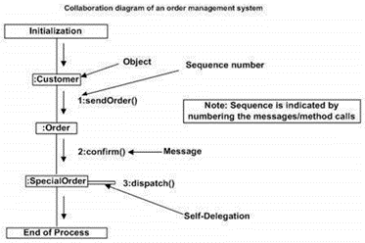


Figure: Sample Use Case diagram









Java programming Environment and Runtime Environment, Development Platforms -Standard, Enterprise. JVM, Java compiler, Bytecode

The Java Programming environment consists of —

- **Java Language** —
    - It is a human-readable language which is generally considered easy to read and write.
    - It is class-based and object-oriented in nature.
    - Java is intended to be easy to learn and to teach.
    - There are many different implementations of Java available, both proprietary and open source.
    - The Java Language Specification (JLS) defines how a confirming Java application must behave.
- **The Java Virtual Machine (JVM)** — The Java Virtual Machine is a program which provides the runtime environment to execute Java programs. Java programs cannot run if a supporting JVM is not available. JVM does not take Java source files as input. The java source code is first converted to bytecode by javac program. Javac takes in source files as input and outputs bytecode in the form of class files with .class extension. These class files are then interpreted (stepped through one at a time) by the JVM interpreter and the program is executed. JVM allows class files from one platform to run on a different environment without any modification or recompilation. This property is known as “write once, run anywhere” (WORA) and thus make Java an easily portable language.
- **The Java Ecosystem** — provides additional value to the developers using the programming language. Java has emerged to be a robust, portable and high performing language.

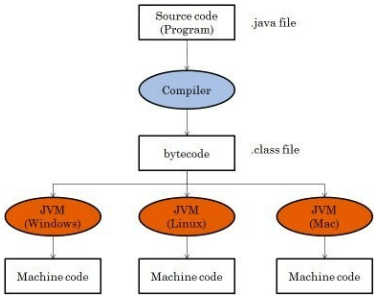
Java Runtime Environment(JRE) -- The Java Runtime Environment, or JRE, is a software layer that runs on top of a computer’s operating system software and provides the class libraries and other resources that a specific Java program needs to run. The JRE combines Java code created using the JDK with the necessary libraries required to run it on a JVM and then creates an instance of the JVM that executes the resulting program.

Java Development Platforms -Standard, Enterprise

- **Java SE -** When most people think of the Java programming language, they think of the Java SE API. Java SE's API provides the core functionality of the Java programming language. It defines everything from the basic types and objects of the Java programming language to high-level classes that are used for networking, security, database access, graphical user interface (GUI) development, and XML parsing. In addition to the core API, the Java SE platform consists of a virtual machine, development tools, deployment technologies, and other class libraries and toolkits commonly used in Java technology applications.
- **Java EE -** The Java EE platform is built on top of the Java SE platform. The Java EE platform provides an API and runtime environment for developing and running large-scale, multi-tiered, scalable, reliable, and secure network applications.

The Java Compiler (Javac) -- is a command line tool that reads java source code files and compiles them into executable Java bytecode classes. The Java source code must be contained in files whose file names end with .java extension. The Java compiler takes files with this extension and generates executable class files with .class extension. It is possible to define more than one class in a single source file and if it happens the Java compiler will generate exactly one class file for each class defined in the source file. The syntax for the Java Compiler is as follows.

Bytecode in Java -- is the reason java is platform-independent, as soon as a Java program is compiled bytecode is generated. To be more precise a Java bytecode is the machine code in the form of a .class file. A bytecode in Java is the instruction set for Java Virtual Machine and acts similar to an assembler.



Java applet, Java Buzzwords, Java program structure, Comments, Garbage Collection, Lexical Issues

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

JAVA APPLICATION	JAVA APPLET
Applications are just like a Java programs that can be execute independently without using the web browser.	Applets are small Java programs that are designed to be included with the HTML web document. They require a Java-enabled web browser for execution.
Application program requires a main function for its execution.	Applet does not require a main function for its execution.
Java application programs have the full access to the local file system and network.	Applets don't have local disk and network access.
Applications can access all kinds of resources available on the system.	Applets can only access the browser specific services. They don't have access to the local system.
Applications can executes the programs from the local system.	Applets cannot execute programs from the local machine.
An application program is needed to perform some task directly for the user.	An applet program is needed to perform small tasks or the part of it.

Java Buzzwords – Features of java

- **Simple -** Java programming language is very simple and easy to learn, understand, and code. Most of the syntaxes in java follow basic programming language C and object-oriented programming concepts are similar to C++. In a java programming language, many complicated features like pointers, operator overloading, structures, unions, etc. have been removed. One of the most useful features is the garbage collector it makes java more simple.
- **Secure -** Java is said to be more secure programming language because it does not have pointers concept, java provides a feature "applet" which can be embedded into a web application. The applet in java does not allow access to other parts of the computer, which keeps away from harmful programs like viruses and unauthorized access.
- **Portable -** Portability is one of the core features of java which enables the java programs to run on any computer or operating system. For example, an applet developed using java runs on a wide variety of CPUs, operating systems, and browsers connected to the Internet.
- **Object-oriented -** Java is said to be a pure object-oriented programming language. In java, everything is an object. It supports all the features of the object-oriented programming paradigm. The primitive data types java also implemented as objects using wrapper classes, but still, it allows primitive data types to archive high-performance.
- **Robust -** Java is more robust because the java code can be executed on a variety of environments, java has a strong memory management mechanism (garbage collector), java is a strictly typed language, it has a strong set of exception handling mechanism, and many more.
- **Architecture-neutral (or) Platform Independent -** Java has invented to archive "write once; run anywhere, any time, forever". The java provides JVM (Java Virtual Machine) to to archive architectural-neutral or platform-independent. The JVM allows the java program created using one operating system can be executed on any other operating system.
- **Multi-threaded -** Java supports multi-threading programming, which allows us to write programs that do multiple operations simultaneously.
- **Interpreted -** Java enables the creation of cross-platform programs by compiling into an intermediate representation called Java bytecode. The byte code is interpreted to any machine code so that it runs on the native machine.
- **High performance -** Java provides high performance with the help of features like JVM, interpretation, and its simplicity.
- **Distributed -** Java programming language supports TCP/IP protocols which enable the java to support the distributed environment of the Internet. Java also supports Remote Method Invocation (RMI), this feature enables a program to invoke methods across a network.
- **Dynamic -** Java is said to be dynamic because the java byte code may be dynamically updated on a running system and it has a dynamic memory allocation and deallocation (objects and garbage collector)

Java program structure

public class Hello	This creates a class called Hello. All class names must start with a capital letter. The public word means that it is accessible from any other classes.
/* Comments */	The compiler ignores comment block. Comment can be used anywhere in the program to add info about the program or code block, which will be helpful for developers to understand the existing code in the future easily.
Braces	Two curly brackets *{...}* are used to group all the commands, so it is known that the commands belong to that class or method.
public static void main	When the main method is declared *public*, it means that it can also be used by code outside of its class, due to which the main method is declared public. The word *static* used when we want to access a method without creating its object, as we call the main method, before creating any class objects. The word *void* indicates that a method does not return a value. main() is declared as void because it does not return a value. *main* is a method; this is a starting point of a Java program. You will notice that the main method code has been moved to some spaces left. It is called *indentation* which used to make a program easier to read and understand.
String[] args	It is an array where each element of it is a string, which has been named as "args". If your Java program is run through the console, you can pass the input parameter, and main() method takes it as input.
System.out.println();	This statement is used to print text on the screen as output, where *the system* is a predefined class, and *out* is an object of the PrintWriter class defined in the system. The method *println* prints the text on the screen with a new line. You can also use *print()* method instead of println() method. All Java statement ends with a semicolon.

Basic Java Programme Structure

Section	Description
Documentation Section	You can write a comment in this section. Comments are beneficial for the programmer because they help them understand the code. These are optional, but we suggest you use them because they are useful to understand the operation of the program, so you must write comments within the program.
Package statement	You can create a package with any name. A package is a group of classes that are defined by a name. That is, if you want to declare many classes within one element, then you can declare it within a package. It is an optional part of the program, i.e., if you do not want to declare any package, then there will be no problem with it, and you will not get any errors. Here, the package is a keyword that tells the compiler that package has been created. It is declared as: `package package_name;`
Import statements	This line indicates that if you want to use a class of another package, then you can do this by importing it directly into your program. Example: `import calc.add;`
Interface statement	Interfaces are like a class that includes a group of method declarations. It's an optional section and can be used when programmers want to implement multiple inheritances within a program.
Class Definition	A Java program may contain several class definitions. Classes are the main and essential elements of any Java program.
Main Method Class	Every Java stand-alone program requires the main method as the starting point of the program. This is an essential part of a Java program. There may be many classes in a Java program, and only one class defines the main method. Methods contain data type declaration and executable statements.

Java 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. here are three types of comments in Java.

- **Single Line Comment -** used to comment only one line.
- **Multi Line Comment -** used to comment multiple lines of code.
- **Documentation Comment -** used to create documentation API. To create documentation API, you need to use javadoc tool.

Garbage Collection -- Java garbage collection is the process by which Java programs perform automatic memory management. Java programs compile to bytecode that can be run on a Java Virtual Machine, or JVM for short. When Java programs run on the JVM, objects are created on the heap, which is a portion of memory dedicated to the program. Eventually, some objects will no longer be needed. The garbage collector finds these unused objects and deletes them to free up memory.

Advantage of Garbage Collection

- It makes java memory efficient because garbage collector removes the unreferenced objects from heap memory.
- It is automatically done by the garbage collector (a part of JVM) so we don't need to make extra efforts.

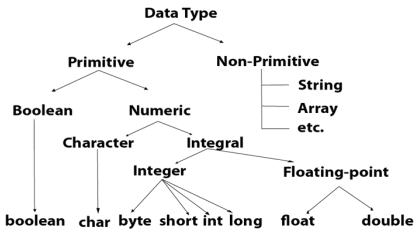
Lexical Issues --

- **White Spaces** - Java is a free form language. This means that you do not need to follow any special indentation rules. In java, white spaces is a space, tab or new line.
- **Identifiers** - Identifiers are used for class names, method names and variable names. An identifier may be any descriptive sequence of uppercase and lowercase letters, numbers or the underscore and dollar sign design.
- **Literals** - A constant value in java is created by using a literal representation of it. A literal can be used anywhere a value of its type is allowed.
- **Comments** - used to produce an HTML file that documents your program. It begins with a/** and ends with a*/.

- **Separators** - There are few symbols in java that are used as separators.The most commonly used separator in java is the **semicolon** ' ; '. some other separators are **Parentheses** '()', **Braces** ' {} ', **Bracket** ' [] ', **Comma** ', ', **Period** '. ' .
- **Java Keywords** - There are 49 reserved keywords currently defined in java. These keywords cannot be used as names for a variable, class or method. The **Keywords** are : abstract , assert , boolean , break , byte , case , catch , char , class , const , continue , default , do , double , else , extends , final , finally , float , for , goto , if , implements , import , instanceof , int interface , long , native , new , package , private , protected , public , return , short , static , strictfp , super , switch , synchronized , this , throw , throws , transient , try , void , volatile, while.
- **Java Class Libraries** - The java environment relies on several built-in class libraries that contain many built-in methods that provide support for such things as I/O , string handling , networking and graphics.

MODULE 2: CORE JAVA FUNDAMENTALS

Core Java Fundamentals: Primitive Data types, Integers, Floating Point Types, Characters, Boolean



char	-2
short	-2
int	-4
float	-4
long	-8
double	-8

- **Boolean Data Type -** 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
- **Byte Data Type -** The byte data type is an example of primitive data type. It isan 8-bit signed two's complement integer. Its value-range lies between -128 to 127 (inclusive). Its minimum value is -128 and maximum value is 127. Its default value is 0. The byte data type is used to save memory in large arrays where the memory savings is most required. It saves space because a byte is 4 times smaller than an integer. It can also be used in place of "int" data type. Example: byte a = 10, byte b = -20
- **Short Data Type -** The short data type is a 16-bit signed two's complement integer. Its value-range lies between -32,768 to 32,767 (inclusive). Its minimum value is -32,768 and maximum value is 32,767. Its default value is 0. The short data type can also be used to save memory just like byte data type. A short data type is 2 times smaller than an integer. Example: short s = 10000, short r = -5000
- **Int Data Type -** 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
- **Long Data Type -** The long data type is a 64-bit two's complement integer. Its value-range lies between - 9,223,372,036,854,775,808(-2^63) to 9,223,372,036,854,775,807(2^63 -1)(inclusive). Its minimum value is - 9,223,372,036,854,775,808and maximum value is 9,223,372,036,854,775,807. Its default value is 0. The long data type is used when you need a range of values more than those provided by int. Example: long a = 100000L, long b = -200000L
- **Float Data Type -** 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
- **Double Data Type -** The double data type is a double-precision 64-bit IEEE 754 floating point. Its value range is unlimited. The double data type is generally used for decimal values just like float. The double data type also should never be used for precise values, such as currency. Its default value is 0.0d. Example: double d1 = 12.3
- **Char Data Type -** 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'

Literals, Type Conversion and Casting, Variables, Arrays, Strings, Vector class.

Literals -- Literals are source code representation of a fixed value or the sequence of characters which represents the constant value that is to be stored in a variable. There are five types of literals in Java.

- **Integer Literals -** The integer literal can be used to create int value. They can be used to initialize the group data types like byte, short, int and long.
- **Boolean Literals -** The Boolean literal is used to represent the two values i.e. either true or false.
- **Character Literals -** The character literal is a 16 bit Unicode character where it is enclosed in single quotes.
- **String Literals -** The string literal are the sequence of characters which are enclosed in double quotes. The string literal should occur in single line.
- **Floating Point Literals -** The floating point literal can be used to represent the decimal values with a fractional component.

Type casting (type conversion) in Java -- Converting one primitive datatype into another. You can cast the primitive datatypes in two ways -

- **Widening** – Converting a lower datatype to a higher datatype is known as widening. In this case the casting/conversion is done automatically therefore, it is known as implicit type casting.
- **Narrowing** – Converting a higher datatype to a lower datatype is known as narrowing. In this case the casting/conversion is not done automatically, you need to convert explicitly using the cast operator "( )" explicitly. Therefore, it is known as explicit type casting.

Variable -- a container which holds the value while the Java program is executed. It is name of reserved area allocated in memory.

- **Local Variable -** A variable declared inside the body of the method is called local variable. You can use this variable only within that method and the other methods in the class aren't even aware that the variable exists. A local variable cannot be defined with "static" keyword.
- **Instance Variable -** A variable declared inside the class but outside the body of the method, is called instance variable. It is not declared as static. It is called instance variable because its value is instance specific and is not shared among instances.
- **Static variable -** A variable which is declared as static is called static variable. It cannot be local. You can create a single copy of static variable and share among all the instances of the class. Memory allocation for static variable happens only once when the class is loaded in the memory**.**

Java array is an object which contains elements of a similar data type. Additionally, The elements of an array are stored in a contiguous memory location. It is a data structure where we store similar elements. We can store only a fixed set of elements in a Java array. There are two types of array - Single Dimensional Array, Multidimensional Array **.**

Advantages

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

Disadvantages

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

Strings in Java -- are Objects that are backed internally by a char array. Since arrays are immutable(cannot grow), Strings are immutable as well. Whenever a change to a String is made, an entirely new String is created.

Below is the basic syntax for declaring a string in Java programming language.
Syntax: *<String_Type> <string_variable> = "<sequence_of_string>";*

Vector class -- implements a growable array of objects. Vectors basically fall in legacy classes but now it is fully compatible with collections. It is found in the java.util package.

Operators: Arithmetic Operators, Bitwise Operators, Relational Operators, Boolean Logical Operators, Assignment Operator, Conditional (Ternary) Operator, Operator Precedence.

Operators are used to perform operations on variables and values.

- **Arithmetic operators** - used in mathematical expressions in the same way that they are used in algebra.

Operator	Description
+ (Addition)	Adds values on either side of the operator.
- (Subtraction)	Subtracts right-hand operand from left-hand operand.
* (Multiplication)	Multiplies values on either side of the operator.
/ (Division)	Divides left-hand operand by right-hand operand.
% (Modulus)	Divides left-hand operand by right-hand operand and returns remainder.
++ (Increment)	Increases the value of operand by 1.
-- (Decrement)	Decreases the value of operand by 1.

- **Relational operators –**

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

- **Bitwise operators –**

Operator	Description
& (bitwise and)	Binary AND Operator copies a bit to the result if it exists in both operands.
| (bitwise or)	Binary OR Operator copies a bit if it exists in either operand.
^ (bitwise XOR)	Binary XOR Operator copies the bit if it is set in one operand but not both.
~ (bitwise compliment)	Binary Ones Complement Operator is unary and has the effect of 'flipping' bits.
<< (left shift)	Binary Left Shift Operator. The left operands value is moved left by the number of bits specified by the right operand.
>> (right shift)	Binary Right Shift Operator. The left operands value is moved right by the number of bits specified by the right operand.

>>> (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.

- **Boolean Logical operators –**

Operator	Description
&& (logical and)	Called Logical AND operator. If both the operands are non-zero, then the condition becomes true.
|| (logical or)	Called Logical OR Operator. If any of the two operands are non-zero, then the condition becomes 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.

- **Assignment operators –**

Operator	Description
=	Simple assignment operator. Assigns values from right side operands to left side operand.
+=	Add AND assignment operator. It adds right operand to the left operand and assign the result to left operand.
-=	Subtract AND assignment operator. It subtracts right operand from the left operand and assign the result to left operand.
*=	Multiply AND assignment operator. It multiplies right operand with the left operand and assign the result to left operand.
/=	Divide AND assignment operator. It divides left operand with the right operand and assign the result to left operand.
%=	Modulus AND assignment operator. It takes modulus using two operands and assign the result to left operand.
<<=	Left shift AND assignment operator.
>>=	Right shift AND assignment operator.
&=	Bitwise AND assignment operator.
^=	bitwise exclusive OR and assignment operator.

|=	bitwise inclusive OR and assignment operator.

- **Conditional (Ternary) Operator -** This operator consists of three operands and is used to evaluate Boolean expressions. The goal of the operator is to decide, which value should be assigned to the variable. The operator is written as – *variable x = (expression) ? value if true : value if false*

Precedence	Operator	Type	Associativity
15	() [] .	Parentheses Array subscript Member selection	Left to Right
14	++ --	Unary post-increment Unary post-decrement	Right to left
13	++ -- + - ! ~ (*type*)	Unary pre-increment Unary pre-decrement Unary plus Unary minus Unary logical negation Unary bitwise complement Unary type cast	Right to left
12	* / %	Multiplication Division Modulus	Left to right
11	+ -	Addition Subtraction	Left to right
10	<< >> >>>	Bitwise left shift Bitwise right shift with sign extension Bitwise right shift with zero extension	Left to right
9	< <= > >= instanceof	Relational less than Relational less than or equal Relational greater than Relational greater than or equal Type comparison (objects only)	Left to right
8	== !=	Relational is equal to Relational is not equal to	Left to right
7	&	Bitwise AND	Left to right
6	^	Bitwise exclusive OR	Left to right
5	|	Bitwise inclusive OR	Left to right
4	&&	Logical AND	Left to right
3	||	Logical OR	Left to right
2	? :	Ternary conditional	Right to left
1	= += -= *= /= %=	Assignment Addition assignment Subtraction assignment Multiplication assignment Division assignment Modulus assignment	Right to left

Larger number means higher precedence.

Control Statements: Selection Statements, Iteration Statements and Jump Statements.

A program executes from top to bottom, except when we use control statements. Then, we can control the order of execution of the program, based on logic and the values. In Java, control statements can be divided into the following three categories:

- **Selection Statements -** Selection statements allow you to control the flow of program execution, on the basis of the outcome of an expression or state of a variable, known during runtime. Selection statements can be divided into the following categories:
    - **The if and if-else statements -** The first contained statement (that can be a block) of an if statement, only executes when the specified condition is true. If the condition is false and there is no keyword, then the first contained statement will be skipped and execution continues with the rest of the program. The condition is an expression that returns a boolean value.
    - **The if-else statements** - In if-else statements, if the specified condition in the if the statement is false, then the statement after the else keyword (that can be a block) will execute.
    - **The if-else-if statements -** The statement following the else keyword can be another if or if-else statement. Whenever the condition is true, the associated statement will be executed and the remaining conditions will be bypassed. If none of the conditions are true, then the else block will execute.
    - **The switch statements** - The switch statement is a multi-way branch statement. The switch statement of Java is another selection statement that defines multiple paths of execution, of a program. It provides a better alternative than a large series of if-else-if statements. The break statement is used inside the switch to terminate a statement sequence.

- **Iteration Statements -** Repeating the same code fragment several times, until a specified condition is satisfied, is called iteration. Iteration statements execute the same set of instructions, until a termination condition is met. Java provides the following loop for iteration statements:

    - **The while loop** - It continually executes a statement (that is usually a block) while a condition is true. The condition must return a boolean value.
    - **The for loop -** A for loop executes a statement (that is usually a block) as long as the boolean condition evaluates to true. A for loop is a combination of the three elements initialization statement, boolean expression and increment or decrement statement.
    - **The do-while loop -** The only difference between a while and a do-while loop, is that do-while evaluates its expression at the bottom of the loop, instead of the top. The do-while loop executes at least one time, then it will check the expression prior to the next iteration.
    - **The for-each loop -** This loop is basically used to traverse the array or collection elements.

- **Jump Statements -** Jump statements are used to unconditionally transfer the program control to another part of the program. Java provides the following jump statements:
    - **break statement - The break statement immediately quits the current iteration and goes to the first statement, following the loop. Another form of break is used in the switch statement. The break statement has the following two forms:**
        - **Labelled Break Statement - This is used for when we want to jump the program control out of nesting loops or multiple loops.**
        - **Unlabelled Break Statement -** This is used to jump program control out of the specific loop on the specific condition.
    - **continue statement** - The continue statement is used when you want to continue running the loop, with the next iteration, and want to skip the rest of the statements of the body, for the current iteration. The continue statement has the following two forms:
        - **Labelled Continue Statement -** This statement skips the current iteration of the loop with the specified label.
        - **Unlabelled Continue Statement** - This statement skips the current iteration of the innermost for, while and do-while loop.
    - **return statement -** The return statement is used to immediately quit the current method and return to the calling method. It is mandatory to use a return statement for non-void methods to return a value.

Object Oriented Programming in Java: Class Fundamentals, Declaring Objects, Object Reference, Introduction to Methods

Object-Oriented Programming or OOPs refers to languages that uses objects in programming. Object-oriented programming aims to implement real-world entities like inheritance, hiding, polymorphism etc in programming. The main aim of OOP is to bind together the data and the functions that operate on them so that no other part of the code can access this data except that function.

A class is a template or blueprint for how to build an object. A class is a prototype that defines state placeholders and behavior common to all objects of its kind. Each object is a member of a single class — there is no multiple inheritance in Java.

An object is an instance of a particular class. an object is created from a class. In Java, the new keyword is used to create new objects.

There are three steps when creating an object from a class –

- **Declaration** – A variable declaration with a variable name with an object type.
- **Instantiation** – The 'new' keyword is used to create the object.
- **Initialization** – The 'new' keyword is followed by a call to a constructor. This call initializes the new object.

// creating object of class Test

Test t = new Test();

Object referencing in java is basically a address in memory where all methods and variables associated with object resides. When you create an object like this...

Example a = new Example(); here a is actually a reference which is pointing to memory assigned using new keyword.

a.x=5.(consider x is some variable).

Now x associated with a has value 5.

A **method** is a block of code which only runs when it is called. A method must be declared within a class. It is defined with the name of the method, followed by parentheses (). Java provides some pre-defined methods, such as System.out.println(), but you can also create your own methods to perform certain actions. static means that the method belongs to the Main class and not an object of the Main class. void means that this method does not have a return value. To call a method in Java, write the method's name followed by two parentheses () and a semicolon**;**

Constructors, this Keyword, Method Overloading, Using Objects as Parameters

A constructor in Java is a special method that is used to initialize objects. The constructor is called when an object of a class is created. The constructor name must match the class name, and it cannot have a return type (like void). A Constructor must have no explicit return type. It calls a default constructor if there is no constructor available in the class. There are two types of constructors in Java: no-arg constructor, and parameterized constructor. The parameterized constructor is used to provide different values to distinct objects. Constructor overloading in Java is a technique of having more than one constructor with different parameter lists. They are arranged in a way that each constructor performs a different task. They are differentiated by the compiler by the number of parameters in the list and their types.

Constructors are different from methods in Java

- Constructor(s) must have the same name as the class within which it defined while it is not necessary for the method in java.
- Constructor(s) do not return any type while method(s) have the return type or void if does not return any value.
- Constructor is called only once at the time of Object creation while method(s) can be called any numbers of time.

The this keyword refers to the current object in a method or constructor. The most common use of the this keyword is to eliminate the confusion between class attributes and parameters with the same name (because a class attribute is shadowed by a method or constructor parameter). this can also be used to:

- Invoke current class constructor
- Invoke current class method
- Return the current class object
- Pass an argument in the method call
- Pass an argument in the constructor call

In Java, two or more methods can have same name if they differ in parameters (different number of parameters, different types of parameters, or both). These methods are called overloaded methods and this feature is called method overloading. There are different ways to perform method overloading: **Overloading by changing the number of arguments, By changing the datatype of parameters**

A method can take an objects as a parameter. For example, in the following program, the method setData( ) takes three parameter. The first parameter is an Data object. If you pass an object as an argument to a method, the mechanism that applies is called pass-by-reference, because a copy of the reference contained in the variable is transferred to the method, not a copy of the object itself.

Returning Objects, Recursion, Access Control, static Members

A method can return any type of data, including class types.

Recursion in java is a process in which a method calls itself continuously.

Access control is a mechanism, an attribute of encapsulation which restricts the access of certain members of a class to specific parts of a program. Access to members of a class can be controlled using the *access modifiers*. There are four access modifiers in Java. They are:

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

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

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.

Final Variables, Inner Classes, Command-Line Arguments, Variable Length Arguments

The **final keyword** in java is used to restrict the user.

- **Variable** - If you make any variable as final, you cannot change the value of final variable (It will be constant).
- **Method -** If you make any method as final, you cannot override it. Final method can be inherited.
- **Class -** If you make any class as final, you cannot extend it.

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**.** There are basically three advantages of inner classes in java. They are as follows:

- 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.

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.

Variable Argument (Varargs) - The varrags allows the method to accept zero or muliple arguments. Before varargs either we use overloaded method or take an array as the method parameter but it was not considered good because it leads to the maintenance problem. If we don't know how many argument we will have to pass in the method, varargs is the better approach. Advantage of Varargs is that we don't have to provide overloaded methods so less code. Syntax - return_type method_name(data_type... variableName){}

Inheritance: Super class, Sub class, the keywords super, protected Members

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

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

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.

The super keyword refers to superclass (parent) objects. It is used to call superclass methods, and to access the superclass constructor. The most common use of the super keyword is to eliminate the confusion between superclasses and subclasses that have methods with the same name.

If a member of a superclass needs to be (directly) accessed in a subclass and yet still prevent its direct access outside the class, you must declare that member **protected**.

Calling Order of Constructors, Method Overriding, the Object class

Order of constructor calls

- The base-class **constructor** is **called**. This step is repeated recursively such that the root of the hierarchy is constructed first, followed by the next-derived class, etc., until the most-derived class is reached.
- Member initializers are **called** in the **order** of declaration.

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. 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).

The **Object class** is the parent class of all the classes in java by default. In other words, it is the topmost class of java. The Object class is beneficial if you want to refer any object whose type you don't know. Notice that parent class reference variable can refer the child class object, know as upcasting**.**

Abstract Classes and Methods, Using final with Inheritance

Abstraction is a process of hiding the implementation details and showing only functionality to the user. There are two ways to achieve abstraction in java

- **Abstract class (0 to 100%) -**
- **Interface (100%)** - An interface in Java is a specification of method prototypes. Whenever you need to guide the programmer or, make a contract specifying how the methods and fields of a type should be you can define an interface. To create an object of this type you need to implement this interface, provide a body for all the abstract methods of the interface and obtain the object of the implementing class. The user who want to use the methods of the interface, he only knows the classes that implement this interface and their methods, information about the implementation is completely hidden from the user, thus achieving 100% abstraction.

A class which is declared as abstract is known as an **abstract class**. It can have abstract and non-abstract methods. It needs to be extended and its method implemented. It cannot be instantiated.

- An abstract class must be declared with an abstract keyword.
- It can have abstract and non-abstract methods.
- It cannot be instantiated.
- It can have constructors and static methods also.
- It can have final methods which will force the subclass not to change the body of the method.

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

If we use the final keyword for the inheritance that is if we declare any method with the final keyword in the base class so the implementation of the final method will be the same as in derived class

---

MODULE 3: MORE FEATURES OF JAVA

Packages and Interfaces: Defining Package, CLASSPATH, Access Protection, Importing Packages

A **java package** is a group of similar types of classes, interfaces and sub-packages. Package in java can be categorized in two form, built-in package and user-defined package. There are many built-in packages such as java, lang, awt, javax, swing, net, io, util, sql etc. Advantage of Java Package --

- Java package is used to categorize the classes and interfaces so that they can be easily maintained.
- Java package provides access protection.
- Java package removes naming collision.

Classpath is an environment variable which is used by Application ClassLoader to locate and load the .class files. The Classpath defines the path, to find third-party and user-defined classes that are not extensions or part of Java platform. Include all the directories which contain .class files and JAR files when setting the Classpath. There are two ways to ways to set CLASSPATH: through Command Prompt or by setting Environment Variable.

To import java package into a class, we need to use java **import** keyword which is used to access package and its classes into the java program. Use import to access built-in and user-defined packages into your java source file so that your class can refer to a class that is in another package by directly using its name. There are 3 different ways to refer to any class that is present in a different package:

- without import the package
- import package with specified class
- import package with all classes

Interfaces

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. There are mainly three reasons to use interface --

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

Input / Output: I/O Basics, Reading Console Input, Writing Console Output, PrintWriter Class

Java I/O (Input and Output) is used to process the input and produce the output. Java uses the concept of a stream to make I/O operation fast. The java.io package contains all the classes required for input and output operations. We can perform file handling in Java by Java I/O API.

There are three different ways to read the input from Java Console, they are –

- **Using Java Bufferedreader Class** - This is the Java traditional technique, introduced in JDK1.0. This strategy is utilized by wrapping the System.in (standard information stream) in an InputStreamReader which is wrapped in a Java BufferedReader, we can read result include from the user in the order line. **Pros –** The information is cradled for productive perusing. **Cons –** The wrapping code is difficult to recall.
- **Scanner Class in Java -** This is presumably the most favoured technique to take input. The primary reason for the Scanner class is to parse primitive composes and strings utilizing general expressions, in any case, it can utilize to peruse contribution from the client in the order line. **Pros** – Helpful strategies for parsing natives (nextInt(), nextFloat(), …) from the tokenized input ,General articulations can utilize to discover tokens. **Cons** – The reading methods are not synchronized.
- **Console Class in Java -** It has been turning into a favored route for perusing client's contribution from the command line. In addition, it can utilize for password key like contribution without resounding the characters entered by the client, the configuration string syntax structure can likewise utilize (like System.out.printf()). **Pros** – Reading secret word without reverberating the entered characters , Reading strategies that are synchronized, Format string sentence structure can utilize. **Cons** – Does not work in non-intelligent condition, (for example, in an IDE).

Console output is most easily accomplished with **print()** and **println()** methods. These methods are defined by the class **PrintStream** which is the type of object referenced by **System.in**. Even though **System.out** is a byte stream, using it for a simple program output is still acceptable.

The Java PrintWriter class (java.io.PrintWriter) enables you to write formatted data to an underlying Writer. For instance, writing int, long and other primitive data formatted as text, rather than as their byte values. The Java PrintWriter is useful if you are generating reports (or similar) where you have to mix text and numbers. The PrintWriter class has all the same methods as the PrintStream except for the methods to write raw bytes. Being a Writer subclass the PrintWriter is intended to write text.

Object Streams and Serialization

ObjectStreamClass act as a Serialization descriptor for class. This class contains the name and serialVersionUID of the class.

Serialization in Java is a mechanism of writing the state of an object into a byte-stream. It is mainly used in Hibernate, RMI, JPA, EJB and JMS technologies. The reverse operation of serialization is called deserialization where byte-stream is converted into an object. The serialization and deserialization process is platform-independent, it means you can serialize an object in a platform and deserialize in different platform. For serializing the object, we call the writeObject() method ObjectOutputStream, and for deserialization we call the readObject() method of ObjectInputStream class. It is mainly used to travel object's state on the network (which is known as marshaling).

Working with Files

File handling in Java implies reading from and writing data to a file. The File class from the java.io package, allows us to work with different formats of files. In order to use the File class, you need to create an object of the class and specify the filename or directory name. You can perform four operations on a file. They are as follows:

- **Create a File -** In this case, to create a file you can use the createNewFile() method. This method returns true if the file was successfully created, and false if the file already exists.
- **Get File Information -** when you execute the program, you will get the file information as output
- **Write To a File -** used the FileWriter class together with its write() method to write some text into the file
- **Read from a File** - When you execute the program, it will display the content present in the given file**.**

Exception Handling: Checked Exceptions, Unchecked Exceptions, try Block and catch Clause

An exception is an unwanted or unexpected event, which occurs during the execution of a program i.e at run time, that disrupts the normal flow of the program's instructions. Exception handling ensures that the flow of the program doesn't break when an exception occurs.

- **Checked exceptions -** All exceptions other than Runtime Exceptions are known as Checked exceptions as the compiler checks them during compilation to see whether the programmer has handled them or not. If these exceptions are not handled/declared in the program, you will get compilation error. For example, SQLException, IOException, ClassNotFoundException etc.

- **Unchecked Exceptions -** Runtime Exceptions are also known as Unchecked Exceptions. These exceptions are not checked at compile-time so compiler does not check whether the programmer has handled them or not but it's the responsibility of the programmer to handle these exceptions and provide a safe exit. For example, ArithmeticException, NullPointerException, ArrayIndexOutOfBoundsException etc.

Java **try** block is used to enclose the code that might throw an exception. It must be used within the method. If an exception occurs at the particular statement of try block, the rest of the block code will not execute. So, it is recommended not to keeping the code in try block that will not throw an exception. Java try block must be followed by either catch or finally block. Syntax of Java try-catch

```
try{
//code that may throw an exception
}catch(Exception_class_Name ref){}
```

Multiple catch Clauses, Nested try Statements

A try block can be followed by one or more catch blocks. Each catch block must contain a different exception handler. So, if you have to perform different tasks at the occurrence of different exceptions, use java multi-catch block. At a time only one

exception occurs and at a time only one catch block is executed. All catch blocks must be ordered from most specific to most general, i.e. catch for ArithmeticException must come before catch for Exception.
The try block within a try block is known as nested try block in java. Sometimes a situation may arise where a part of a block may cause one error and the entire block itself may cause another error. In such cases, exception handlers have to be nested.

```
1.  try
2.  {
3.      statement 1;
4.      statement 2;
5.      try
6.      {
7.          statement 1;
8.          statement 2;
9.      }
10.     catch(Exception e)
11.     {
12.     }
13. }
14. catch(Exception e)
15. {
16. }
17. ....
```

throw, throws and finally

Java exception handling is managed via five keywords: try, catch, throw, throws, and finally. Program statements that you think can raise exceptions are contained within a try block. If an exception occurs within the try block, it is thrown. Your code can catch this exception (using catch block) and handle it in some rational manner. System-generated exceptions are automatically thrown by the Java run-time system. To manually throw an exception, use the keyword throw. Any exception that is thrown out of a method must be specified as such by a throws clause. Any code that absolutely must be executed after a try block completes is put in a finally block.

The throw keyword is used to create a custom error. The throw statement is used together with an **exception type**. There are many exception types available in Java:
ArithmeticException, ClassNotFoundException, ArrayIndexOutOfBoundsException, SecurityException, etc.

The **Java throws keyword** is used to declare an exception. It gives an information to the programmer that there may occur an exception so it is better for the programmer to provide the exception handling code so that normal flow can be maintained.

throw	**throws**
throw keyword is used to throw an exception explicitly.	throws keyword is used to declare an exception possible during its execution.
throw keyword is followed by an instance of Throwable class or one of its sub-classes.	throws keyword is followed by one or more Exception class names separated by commas.

throw keyword is declared inside a method body.	throws keyword is used with method signature (method declaration).
We cannot throw multiple exceptions using throw keyword.	We can declare multiple exceptions (separated by commas) using throws keyword.

A finally keyword is used to create a block of code that follows a try block. A finally block of code is always executed whether an exception has occurred or not. Using a finally block, it lets you run any cleanup type statements that you want to execute, no matter what happens in the protected code. A finally block appears at the end of catch block.

MODULE 4: ADVANCED FEATURES OF JAVA

Java Library: String Handling – String Constructors, String Length, Special String Operations

String is basically an object that represents sequence of char values. An array of characters works same as Java string. Java String class provides a lot of methods to perform operations on strings such as compare (), concat (), equals(), split(), length(), replace(), compareTo(), intern(), substring() etc. The Java String class implements Serializable, Comparable and CharSequence interface. In Java, CharSequence Interface is used for representing a sequence of characters. CharSequence interface is implemented by String, StringBuffer and StringBuilder classes. This 3 classes can be used for creating strings in java.
The length property of a String object contains the length of the string, in UTF-16 code units. length is a read-only data property of string instances.

String "Length" Method in Java -This function is used to get the length of string in Java. The string length method returns the number of characters written in the String. This method returns the length of any string which is equal to the number of 16-bit Unicode characters in the string.

Special String Operations

- **String Literals** - Java automatically constructs a String object. Thus, you can use a string literal to initialize a String object.
  char chars[] = { 'a', 'b', 'c' }; String s1 = new String(chars);
  String s2 = "abc"; // use string literal
- **String Concatenation -** Java does not allow operators to be applied to String objects. The one exception to this rule is the + operator, which concatenates two strings, producing a String object as the result.
  String age = "9";
  String s = "He is " + age + " years old.";
  System.out.println(s);
  This displays the string "He is 9 years old."
- **String Conversion and toString( ) -**

Character Extraction, String Comparison, Searching Strings, Modifying Strings, using valueOf(), Comparison of StringBuffer and String.

- **Character Extraction -** To extract a single character from a String, you can refer directly to an individual character via the charAt( ) method. It has this general form: char charAt(int *where*) .Here, *where* is the index of the character that you want to obtain. The value of *where* must be nonnegative and specify a location within the string. charAt( ) returns the character at the specified location. For example, char ch; ch = "abc".charAt(1);

  If you need to extract more than one character at a time, you can use the getChars( ) method. It has this general form:void getChars(int sourceStart, int sourceEnd, char target[ ], int targetStart).

  There is an alternative to **getChars( )** that stores the characters in an array of bytes. This method is called **getBytes( )**, and it uses the default character-to-byte conversions provided by the platform. Here is its simplest form: byte[ ]

getBytes( ).

If you want to convert all the characters in a **String** object into a character array, the easiest way is to call **toCharArray( )** . It returns an array of characters for the entire string. It has this general form: char[ ] toCharArray( )

- **String Comparison -** To compare two strings for equality, use equals( ). It has this general form: Boolean equals(Object *str*). The comparison is case-sensitive.

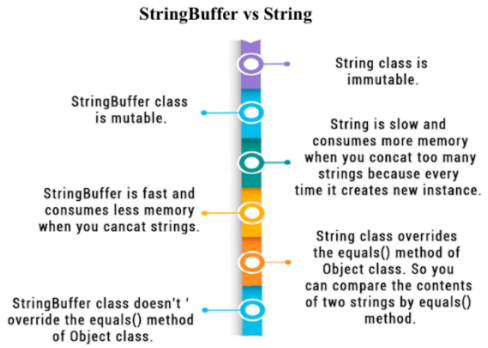
  To perform a comparison that ignores case differences, call **equalsIgnoreCase( )**. It has this general form: boolean equalsIgnoreCase(String *str*). The **regionMatches( )** method compares a specific region inside a string with another specific region in another string.

  String defines two methods that are, more or less, specialized forms of **regionMatches( )**. The **startsWith( )** method determines whether a given **String** begins with a specified string. Conversely, **endsWith( )** determines whether the **String** in question ends with a specified string. They have the following general forms: boolean startsWith(String *str*) boolean endsWith(String *str*)

- **Searching Strings -** The String class provides two methods that allow you to search a string for a specified character or substring:

  indexOf( ) Searches for the first occurrence of a character or substring.
  lastIndexOf( ) Searches for the last occurrence of a character or substring.

- **Modifying a String -** Because String objects are immutable, whenever you want to modify a String, you must either copy it into a StringBuffer or StringBuilder, or use a String method that constructs a new copy of the string with your modifications complete.
  You can extract a substring using **substring( )**. It has two forms.
  The first is String substring(int *startIndex*). Here, *startIndex* specifies the index at which the substring will begin. This form returns a copy of the substring that begins at *startIndex* and runs to the end of the invoking string.
  The second form of **substring( )** allows you to specify both the beginning and ending index of the substring: String substring(int *startIndex*, int *endIndex*) Here, *startIndex* specifies the beginning index, and *endIndex* specifies the stopping point. The string returned contains all the characters from the beginning index, up to, but not including, the ending index.
  You can concatenate two strings using **concat( )**, shown here: String concat(String *str*)
  This method creates a new object that contains the invoking string with the contents of *str* appended to the end. **concat( )** performs the same function as **+**.
  The **replace( )** method has two forms.
  The first replaces all occurrences of one character in the invoking string with another character. It has the following general form: String replace(char *original*, char *replacement*). Here, *original* specifies the character to be replaced by the character specified by *replacement*. The resulting string is returned.
  The second form of **replace( )** replaces one character sequence with another. It has this general form:
  String replace(CharSequence *original*, CharSequence *replacement*)
  The **trim( )** method returns a copy of the invoking string from which any leading and trailing whitespace has been removed. It has this general form: String trim( ). The **trim( )** method is quite useful when you process user commands.
- **Data Conversion Using valueOf( ) -** The valueOf( ) method converts data from its internal format into a human-readable form. It is a static method that is overloaded within String for all of Java's built-in types so that each type can be converted properly into a string. valueOf( ) is also overloaded for type Object, so an object of any class type you create can also be used as an argument. Here are a few of its forms:
  static String valueOf(double *num*) static String valueOf(long *num*) static String valueOf(Object *ob*) static String valueOf(char *chars*[ ])



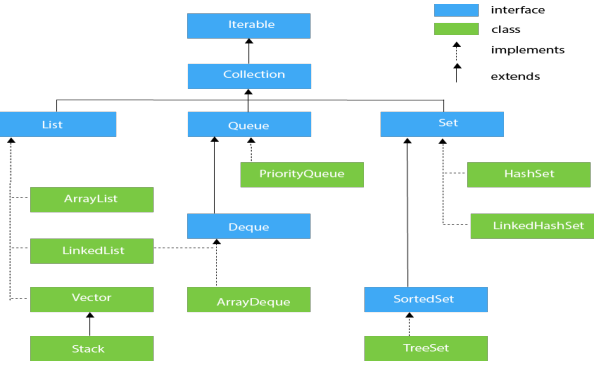
Collections framework - Collections overview, Collections Interfaces- Collection Interface

The Collection in Java is a framework that provides an architecture to store and manipulate the group of objects. Java Collections can achieve all the operations that you perform on a data such as searching, sorting, insertion, manipulation, and deletion. Java Collection means a single unit of objects. Java Collection framework provides many interfaces (Set, List, Queue, Deque) and classes (ArrayList, Vector, LinkedList, PriorityQueue, HashSet, LinkedHashSet, TreeSet).

framework in Java

- It provides readymade architecture.
- It represents a set of classes andinterfaces.
- It is optional.

java.util package contains all the classes and interfaces for the Collection framework.



The **Iterable interface** is the root interface for all the collection classes. The Collection interface extends the Iterable interface and therefore all the subclasses of Collection interface also implement the Iterable interface. It contains only one abstract method. i.e., Iterator<T> iterator()

methods of **Collection interface** are Boolean add (Object obj), Boolean addAll (Collection c), void clear(), etc. which are implemented by all the subclasses of Collection interface.

List Interface, Collections Class – ArrayList class.

- **List interface** is the child interface of Collection interface. It inhibits a list type data structure in which we can store the ordered collection of objects. It can have duplicate values. List interface is implemented by the classes ArrayList, LinkedList, Vector, and Stack.

  It uses a doubly linked list internally to store the elements. It can store the duplicate elements. It maintains the insertion order and is not synchronized. **In LinkedList**, the manipulation is fast because no shifting is required.

  Vector uses a dynamic array to store the data elements. It is similar to ArrayList. However, It is synchronized and contains many methods that are not the part of Collection framework.

  The stack is the subclass of Vector. It implements the last-in-first-out data structure, i.e., Stack. The stack contains all of the methods of Vector class and also provides its methods like boolean push(), boolean peek(), boolean push(object o), which defines its properties.
- **Queue** interface maintains the first-in-first-out order. It can be defined as an ordered list that is used to hold the elements which are about to be processed. There are various classes like PriorityQueue, Deque, and ArrayDeque which implements the Queue interface.

  The **PriorityQueue** class implements the Queue interface. It holds the elements or objects which are to be processed by their priorities. PriorityQueue doesn't allow null values to be stored in the queue.

Deque interface extends the Queue interface. In Deque, we can remove and add the elements from both the side. Deque stands for a double-ended queue which enables us to perform the operations at both the ends.
ArrayDeque class implements the Deque interface. It facilitates us to use the Deque. Unlike queue, we can add or delete the elements from both the ends. ArrayDeque is faster than ArrayList and Stack and has no capacity restrictions.

- **Set Interface** in Java is present in java.util package. It extends the Collection interface. It represents the unordered set of elements which doesn't allow us to store the duplicate items. We can store at most one null value in Set. Set is implemented by HashSet, LinkedHashSet, and TreeSet.

  It represents the collection that uses a hash table for storage. Hashing is used to store the elements in **the HashSet**. It contains unique items.
  LinkedHashSet class represents the LinkedList implementation of Set Interface. It extends the HashSet class and implements Set interface. Like HashSet, It also contains unique elements. It maintains the insertion order and permits null elements.
  SortedSet is the alternate of Set interface that provides a total ordering on its elements. The elements of the SortedSet are arranged in the increasing (ascending) order. The SortedSet provides the additional methods that inhibit the natural ordering of the elements.
  Java TreeSet class implements the Set interface that uses a tree for storage. Like HashSet, TreeSet also contains unique elements. However, the access and retrieval time of TreeSet is quite fast. The elements in TreeSet stored in ascending order.

It uses a dynamic array to store the duplicate element of different data types. The **ArrayList** class maintains the insertion order and is non-synchronized. The elements stored in the ArrayList class can be randomly accessed

- Java ArrayList class can contain duplicate elements.
- Java ArrayList class maintains insertion order.
- Java ArrayList class is non synchronized.
- Java ArrayList allows random access because array works at the index basis.
- In ArrayList, manipulation is little bit slower than the LinkedList in Java because a lot of shifting needs to occur if any element is removed from the array list.

Accessing a Collection via an Iterator.

following steps to access a collection of elements using the Iterator.
Step - 1: Create an object of the Iterator by calling collection.itertor( ) method.
Step - 2: Use the method hasNext( ) to access to check does the collection has the next element. (Use a loop).
Step - 3: Use the method next( ) to access each element from the collection. (use inside the loop).
The Iterator has the following methods.

Method	Description
Iterator iterator( )	Used to obtain an iterator to the start of the collection.
boolean hasNext( )	Returns true if the collection has the next element, otherwise, it returns false.
E next( )	Returns the next element available in the collection.

Event handling - Event Handling Mechanisms, Delegation Event Model, Event Classes

Event Handling is the mechanism that controls the event and decides what should happen if an event occurs. This mechanism have 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. Steps involved in event handling
•The User clicks the button and the event is generated.
•Now the object of concerned event class is created automatically and information about the source and the event get populated with in same object.
•Event object is forwarded to the method of registered listener class.
•the method is now get executed and returns.

Events

1. In the delegation model, an event is an object that describes a state change in a source.
2. It can be generated as a consequence of a person interacting with the elements in a graphical user interface.
3. Some of the activities that cause events to be generated are pressing a button, entering a character via the keyboard, selecting an item in a list, and clicking the mouse.
4. Event may also occur that are not directly caused by interactions with a user interface. For example, an event may be generated when a timer expires, a counter exceeds a value, a software or hardware failure occurs, or an operation is completed.

Event Class	Description
ActionEvent	Generated when a button is pressed, a list item is double clicked, a menu item is selected.
AdjustmentEvent	Generated when a scroll bar is manipulated.
ComponentEvent	Generated when a component is hidden, moved, resized, or becomes visible.
ContainerEvent	Generated when a component is added to or removed from a container.
FocusEvent	Generated when a component gains or loses keyboard focus.
InputEvent	Abstract superclass for all component input event classes.
ItemEvent	Generated when a check box or list item is clicked; also occurs when a choice selection is made or a checkable menu item is selected or deselected.
KeyEvent	Generated when input is received from the keyboard.
MouseEvent	Generated when the mouse is dragged, moved, clicked, pressed, or released; also generated when the mouse enters or exits a component.
MouseWheelEvent	Generated when a mouse wheel is moved.
TextEvent	Generated when the value of a text area or text field is changed.
WindowEvent	Generated when a window is activated, closed, deactivated, deiconified, iconified, opened, or quit.

Sources of Events, Event Listener Interfaces, Using the Delegation Model

Event Source	Description
Button	Generates action events when the button is pressed.
Check box	Generates item events when the check box is selected or deselected.
Choice	Generates item events when the choice is changed.
List	Generates action events when an item is double-clicked; generates item events when an item is selected or deselected.
Menu item	Generates action events when a menu item is selected; generates item events when a checkable menu item is selected or deselected.
Scroll bar	Generates adjustment events when the scroll bar is manipulated.
Text components	Generates text events when the user enters a character.
Window	Generates window events when a window is activated, closed, deactivated, deiconified, iconified, opened, or quit.

Interface	Description
ActionListener	Defines one method to receive action events.
AdjustmentListener	Defines one method to receive adjustment events.
ComponentListener	Defines four methods to recognize when a component is hidden, moved, resized, or shown.
ContainerListener	Defines two methods to recognize when a component is added to or removed from a container.
FocusListener	Defines two methods to recognize when a component gains or loses keyboard focus.
ItemListener	Defines one method to recognize when the state of an item changes.
KeyListener	Defines three methods to recognize when a key is pressed, released, or typed.
MouseListener	Defines five methods to recognize when the mouse is clicked, enters a component, exits a component, is pressed, or is released.
MouseMotionListener	Defines two methods to recognize when the mouse is dragged or moved.
MouseWheelListener	Defines one method to recognize when the mouse wheel is moved.
TextListener	Defines one method to recognize when a text value changes.
WindowFocusListener	Defines two methods to recognize when a window gains or loses input focus.
WindowListener	Defines seven methods to recognize when a window is activated, closed, deactivated, deiconified, iconified, opened, or quit.

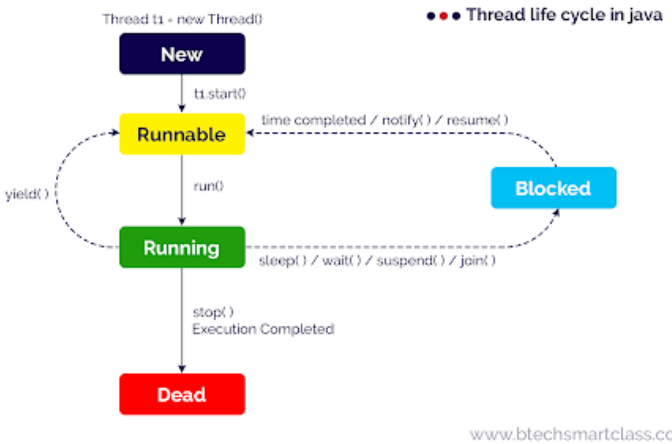
Multithreaded Programming - The Java Thread Model, The Main Thread, Creating Thread

Multithreading is a Java feature that allows concurrent execution of two or more parts of a program for maximum utilization of CPU. Each part of such program is called a thread. So, threads are light-weight processes within a process.
Threads can be created by using two mechanisms :
1. Extending the Thread class - We create a class that extends the **java.lang.Thread** class. This class overrides the run() method available in the Thread class. A thread begins its life inside run() method. We create an object of our new class and call start() method to start the execution of a thread. Start() invokes the run() method on the Thread object.

2. Implementing the Runnable Interface - We create a new class which implements java.lang.Runnable interface and override run() method. Then we instantiate a Thread object and call start() method on this object.

Thread Class vs Runnable Interface
1. If we extend the Thread class, our class cannot extend any other class because Java doesn't support multiple inheritance. But, if we implement the Runnable interface, our class can still extend other base classes.

2. We can achieve basic functionality of a thread by extending Thread class because it provides some inbuilt methods like yield(), interrupt() etc. that are not available in Runnable interface.



New -When a thread object is created using new, then the thread is said to be in the New state. This state is also known as Born state. **Example** : Thread t1 = new Thread();

Runnable / Ready -When a thread calls start( ) method, then the thread is said to be in the Runnable state. This state is also known as a Ready state. **Example:** t1.start( );

Running -When a thread calls run( ) method, then the thread is said to be Running. The run( ) method of a thread called automatically by the start( ) method.

Blocked / Waiting -A thread in the Running state may move into the blocked state due to various reasons like sleep( ) method called, wait( ) method called, suspend( ) method called, and join( ) method called, etc. When a thread is in the blocked or waiting state, it may move to Runnable state due to reasons like sleep time completed, waiting time completed, notify( ) or notifyAll( ) method called, resume( ) method called, etc. **Example** : Thread.sleep(1000); wait(1000); wait(); suspened(); notify(); notifyAll(); resume();

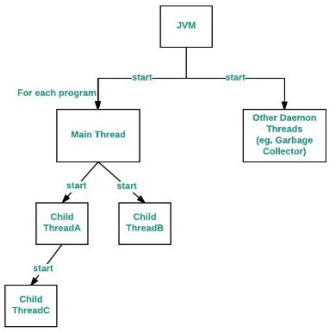
Dead / Terminated - A thread in the Running state may move into the dead state due to either its execution completed or the stop( ) method called. The dead state is also known as the terminated state.

When a Java program starts up, one thread begins running immediately. This is usually called the *main* thread of our program, because it is the one that is executed when our program begins.

Properties :

- It is the thread from which other “child” threads will be spawned.
- Often, it must be the last thread to finish execution because it performs various shutdown actions

Flow diagram :



How to control Main thread - The main thread is created automatically when our program is started. To control it we must obtain a reference to it. This can be done by calling the method currentThread( ) which is present in Thread class. This method returns a reference to the thread on which it is called. The default priority of Main thread is 5 and for all remaining user threads priority will be inherited from parent to child.

Creating Multiple Threads, Synchronization, Suspending, Resuming and Stopping Threads

Synchronization in java is the capability to control the access of multiple threads to any shared resource. Java Synchronization is better option where we want to allow only one thread to access the shared resource. The synchronization is mainly used to

1. To prevent thread interference.
2. To prevent consistency problem.

There are two types of synchronization

1. Process Synchronization
2. Thread Synchronization

There are two types of thread synchronization mutual exclusive and inter-thread communication.

1. Mutual Exclusive

1. Synchronized method.
2. Synchronized block.
3. static synchronization.

2. Cooperation (Inter-thread communication in java)

Mutual Exclusive helps keep threads from interfering with one another while sharing data. This can be done by three ways in java:

1. by synchronized method
2. by synchronized block
3. by static synchronization

Suspending - The suspend( ) method of the Thread class was deprecated by Java 2 several years ago. This was done because suspend( ) can sometimes cause serious system failures. Assume that a thread has obtained locks on critical data structures. If that thread is suspended at that point, those locks are not relinquished. Other threads that may be waiting for those resources can be deadlocked.

Resuming - The resume( ) method is also deprecated. It does not cause problems, but cannot be used without the suspend( ) method as its counterpart.

Stopping - The stop( ) method of the Thread class, too, was deprecated by Java 2. This was done because this method can sometimes cause serious system failures. Assume that a thread is writing to a critically important data structure and has completed only part of its changes. If that thread is stopped at that point, that data structure might be left in a corrupted state.

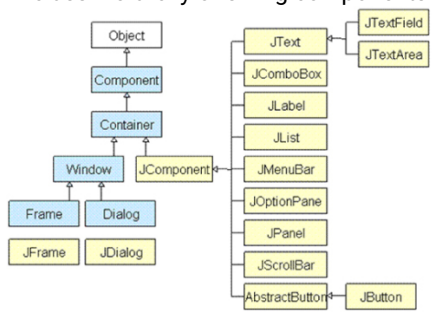
MODULE 5: Graphical User Interface and Database support of Java

Swings fundamentals - Swing Key Features

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.

No.	Java AWT	Java Swing
1)	AWT components are **platform-dependent.**	Java swing components are **platform-independent.**
2)	AWT components are **heavyweight.**	Swing components are **lightweight.**
3)	AWT **doesn't support pluggable look and feel.**	Swing **supports pluggable look and feel.**
4)	AWT provides **less components** than Swing.	Swing provides **more powerful components** such as tables, lists, scrollpanes, colorchooser, tabbedpane etc.
5)	AWT **doesn't follows MVC**(Model View Controller) where model represents data, view represents presentation and controller acts as an interface between model and view.	Swing **follows MVC.**

class hierarchy of swing components



The methods of Component class are widely used in java swing

Method	Description
public void add(Component c)	add a component on another component.
public void setSize(int width,int height)	sets size of the component.
public void setLayout(LayoutManager m)	sets the layout manager for the component.
public void setVisible(boolean b)	sets the visibility of the component. It is by default false.

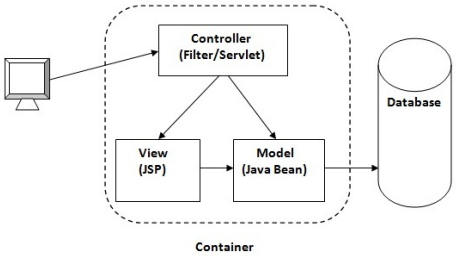
Main Features of Swing

- Platform Independent
- Customizable
- Extensible
- Configurable
- Lightweight
- Rich Controls
- Pluggable Look and Feel

Model View Controller (MVC), Swing Controls, Components and Containers

MVC stands for Model View and Controller. It is a **design pattern** that separates the business logic, presentation logic and data. **Controller** acts as an interface between View and Model. Controller intercepts all the incoming requests.**Model** represents the state of the application i.e. data. It can also have business logic.**View** represents the presentaion i.e. UI(User Interface). Advantage of MVC (Model 2) Architecture

1. Navigation Control is centralized
2. Easy to maintain the large application



A component is an independent visual control and Java Swing Framework contains a large set of these components which provide rich functionalities and allow high level of customization. They all are derived from JComponent class. All these components are lightweight components. This class provides some common functionality like pluggable look and feel, support for accessibility, drag and drop, layout, etc.

A container holds a group of components. It provides a space where a component can be managed and displayed. Containers are of two types:

1. **Top level Containers**

- It inherits Component and Container of AWT.
- It cannot be contained within other containers.
- Heavyweight.
- Example: JFrame, JDialog, JApplet

2. **Lightweight Containers**
    - It inherits JComponent class.
    - It is a general purpose container.
    - It can be used to organize related components together.
    - Example: JPanel

Swing Controls

JLabel - A JLabel object is a component for placing text in a container.

JButton - This class creates a labeled button.

JColorChooser - A JColorChooser provides a pane of controls designed to allow a user to manipulate and select a color.

JCheck Box - A JCheckBox is a graphical component that can be in either an on (true) or off (false) state.

JRadioButton- The JRadioButton class is a graphical component that can be in either an on (true) or off (false) state. in a group.

JList - A JList component presents the user with a scrolling list of text items.

JComboBox - A JComboBox component presents the user with a to show up menu of choices.

JTextField - A JTextField object is a text component that allows for the editing of a single line of text.

JPasswordField - A JPasswordField object is a text component specialized for password entry.

JTextArea - A JTextArea object is a text component that allows editing of a multiple lines of text.

ImageIcon - A ImageIcon control is an implementation of the Icon interface that paints Icons from Images

JScrollbar - A Scrollbar control represents a scroll bar component in order to enable the user to select from range of values.

JOptionPane - JOptionPane provides set of standard dialog boxes that prompt users for a value or informs them of something.

JFileChooser- A JFileChooser control represents a dialog window from which the user can select a file.

JProgressBar -As the task progresses towards completion, the progress bar displays the task's percentage of completion.

JSlider - A JSlider lets the user graphically select a value by sliding a knob within a bounded interval.

JSpinner - A JSpinner is a single line input field that lets the user select a number or an object value from an ordered sequence.

Swing Packages, Event Handling in Swings

The Swing Packages

- Swing is a very large subsystem and makes use of many packages. These are the packages used by Swing that are defined by Java SE 6.

javax.swing	javax.swing.border	javax.swing.colorchooser
javax.swing.event	javax.swing.filechooser	javax.swing.plaf
javax.swing.plaf.basic	javax.swing.plaf.metal	javax.swing.plaf.multi
javax.swing.plaf.synth	javax.swing.table	javax.swing.text
javax.swing.text.html	javax.swing.text.html.parser	javax.swing.text.rtf
javax.swing.tree	javax.swing.undo	

The main package is **javax.swing. This package must be imported into any program** that uses Swing. It contains the classes that implement the basic Swing components, such as push buttons, labels, and check boxes.

Swing Layout Managers

The LayoutManagers are used to arrange components in a particular manner. LayoutManager is an interface that is implemented by all the classes of layout managers. There are following classes that represents the layout managers:

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

BorderLayout is used, when we want to arrange the components in five regions. The five regions can be north, south, east, west and the centre. There are 5 types of constructor in Border Layout. They are as following:

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

Exploring Swings –JFrame, JLabel, The Swing Buttons, JTextField.

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.

JLabel - JLabel is a class of java Swing . JLabel is used to display a short string or an image icon. JLabel can display text, image or both . JLabel is only a display of text or image and it cannot get focus . JLabel is inactive to input events such a mouse focus or keyboard focus. By default labels are vertically centered but the user can change the alignment of label.

The Swing Buttons - 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.

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

Java DataBase Connectivity (JDBC) overview

JDBC stands for Java Database Connectivity. JDBC is a Java API to connect and execute the query with the database. It is a part of JavaSE (Java Standard Edition). JDBC API uses JDBC drivers to connect with the database. There are four types of JDBC drivers:

- JDBC-ODBC Bridge Driver,
- Native Driver,
- Network Protocol Driver, and
- Thin Driver

There are 5 steps to connect any java application with the database using JDBC. These steps are as follows:

- Register the Driver class - **forName()** method of Class class is used to register the driver class. This method is used to dynamically load the driver class. Syntax of forName() method **public static void** forName(String className)**throws** ClassNotFoundException

- Create connection - The **getConnection()** method of DriverManager class is used to establish connection with the database. Syntax of getConnection() method

  public static Connection getConnection(String url)**throws** SQLException
  public static Connection getConnection(String url,String name,String password) **throws** SQLException

- Create statement - The createStatement() method of Connection interface is used to create statement. The object of statement is responsible to execute queries with the database. **public** Statement createStatement()**throws** SQLException

- Execute queries - The executeQuery() method of Statement interface is used to execute queries to the database. This method returns the object of ResultSet that can be used to get all the records of a table. **public** ResultSet executeQuery(String sql)**throws** SQLException

- Close connection - By closing connection object statement and ResultSet will be closed automatically. The close() method of Connection interface is used to close the connection. **public void** close()**throws** SQLException

Creating and Executing Queries – create, table, delete, insert, select

Structured Query Language (SQL) is a standardized language that allows you to perform operations on a database, such as creating entries, reading content, updating content, and deleting entries.

SQL is supported by almost any database that is used, and it allows to write database code independently of the underlying database.

- Create Database - CREATE DATABASE EMP;
- Drop Database - DROP DATABASE DATABASE_NAME;
- Create Table - CREATE TABLE Employees (id INT NOT NULL, age INT NOT NULL, first VARCHAR(255), last VARCHAR(255), PRIMARY KEY (id));
- Drop Table - DROP TABLE table_name;
- INSERT Data - INSERT INTO Employees VALUES (100, 18, 'Zara', 'Ali');
- SELECT Data - SELECT first, last, age FROM Employees WHERE id =100;
- UPDATE Data - UPDATE Employees SET age=20 WHERE id=100;
- DELETE Data - DELETE FROM Employee WHERE id=100

A stream is a sequence of objects that supports various methods which can be pipelined to produce the desired result.
The features of Java stream are –
•A stream is not a data structure instead it takes input from the Collections, Arrays or I/O channels.
•Streams don't change the original data structure, they only provide the result as per the pipelined methods.
•Each intermediate operation is lazily executed and returns a stream as a result, hence various intermediate operations can be pipelined. Terminal operations mark the end of the stream and return the result.

Dynamic method dispatch is the mechanism by which a call to an overridden method is resolved at run time, rather than compile time

Compile-time polymorphism is achieved through method overloading

MySQL commands

1. Data base & table creats

```
CREATE DATABASE students;
USE students
CREATE TABLE Informats (firstname VARCHAR(20, lastname
        VARCHAR(20), gender CHAR(1), grade INT(10), dob DATE));

SHOW TABLES;
DESCRIBE Informats;
```

Field	Type	Null	key	Default
firstname	Varcha(20)	Yes		NULL
lastnae	"	"		"
gende	"	"		"
grade	int(10)	"		"
dob	date	"		"

(field → name of colum, type → data type, null – if attribute can have null value, key – attribute can be or foreign key, default – attribute comes with a predefined default value)

```
INSERT INTO Informats VALUES ('Amanda', 'Willians', 'f', '10', '1999-03-30');
    "
    "
SELECT * FROM Firstformats;
```

first name	last name	gende	grade	dob
Amanda	Wilkans	f	10	1999-03-30
"				
"				

```
ALTER TABLE Informats ADD COLUMN rollnumba INT(10);
ALTER TABLE Informats ADD PRIMARY KEY (rollnumba)
INSERT INTO Informats rollnumber VALUE ('001');
SELECT COUNT(*) FROM Informats (count no. of rows)
SELECT * FROM Informats. WHERE lastname = 'Willians';
```

first name	Last name	gende	grade	dob
Amanda	williams	f	10	1999-03-30

```
UPDATE Informats SET dob = '1999-02-02' WHERE Last name
                                = Wilkins';
DELETE FROM Informats WHERE dob = 1999-03-30;
```