Module 1- core java

❖ What is java?

- Java is high-level object-oriented and platform-independent programming language.

❖ History of Java

- Java was developed by sun microsystems in 1991.
- The language was initially named as Oak, but in 1995 it was renamed as Java.
- The first official version of Java was released in 1996.
- James Gosling known as “Father Of Java”.
- Java is one of the most widely used programming languages in the world and consistently ranks in the top 3 most popular programming languages.
- Twitter, LinkedIn, Netflix and many enterprise applications were developed using Java.

❖ Features of Java

- Simple
- Object-oriented
- Portable
- Platform independent
- Secured
- Robust
- Architecture neutral
- Interpreted
- High performance
- Multithreaded
- Distributed
- Dynamic

❖ JDK

- JDK stands for “Java Devlopment Kit”.
- JDK is the complete package used to devlop and run Java application.
- It contains JRE(JVM+Libraries) and Devlopment tools
- Used for both development and execution of java programs.

❖ JVM

- JVM stands for “Java Virtual Machine”.
- JVM is runtime environment that executes Java bytecode.
- It makes java plantform-independent by converting byte code into machine code(.class file) of the operating system.
- It runs the java programs

❖ JRE

- Java Runtime Environment is the software package that provides the environment to run java applications.
- It contains JVM and Libraries but it does not have development tools like compiler.
- Allows running of java programs but does not provide development tools.

❖ Setting up the Java environment

- Go to the Oracle JDK download page or use OpenJDK.
- Download the latest **JDK** for your OS (Windows/Mac/Linux).
- Install it by following the installation wizard.
- Set JAVA_HOME environment variable (Windows).
- Search for "Environment Variables".
- Under System Variables click New →JAVA_HOME → set it to your JDK path(e.g. C:\PROGRAM_FILES\JAVA\JDK-21)

❖ Setting up the IDE

- Go to https://www.eclipse.org/downloads.
- Download "Eclipse IDE for Java Developers"**.**
- Run the installer and follow the instructions

❖ Java Program Structure

Package Statement
Import statement
Comments[optional]
Class Declaration
{
    Variable Declaration
    Constructors
    Methods
    Nested Classes
    Nested Interfaces
    Enums
}

These things can be written in any order and all are optional

❖ Primitive Data Types in Java

- Primitive data types are the most basic data type and they store simple values directly in memory.
- Java has 8 primitive data types.
    1. Byte
    2. Short
    3. Int
    4. Long
    5. Float
    6. Double
    7. Char
    8. Boolean

❖ Variable Declaration and Initialization

- A variable is names memory location that stores a value.
- Its acts like a container that holds data while a program is running.
- **Declaration** means telling java what type of data the variable will hold and giving it a name.

Datatype variable_name;
Example: Int number;
Float price;

- **Initialization** means assigning a value to the variable for the first time.

  variable_name = value;
  Example: number=10;
  Price=99.99;

- **Declaration + Initialization** means declare and initialize a variable in a single step.

  Datatype variable_name = value;
  Example: int number=10;
  Float Price=99.99;

❖ Operators

- An operator in java is a symbol that performs an operation one one or more operands.

1. Arithmetic Operator

- Used for basic mathematical operations.

Operator	Meaning	Example	Result
+	Addition	10+5	15
-	Subtraction	10-5	5
*	Multiplication	10*5	50
/	Division	10/5	2
%	Modulus	10%5	0

2. Relational Operator

- Used to compare two values. Return True or False.

Operator	Meaning	Example	Result
==	Equal to	10==5	false
!=	Not Equal to	10!=5	true
>	Greater than	10>5	true
<	Less than	10<5	false
>=	Greater than or equal	10>=5	true
<=	Less than Or equal	10<=5	false

3. Logical Operators

- Used for combining multiple conditions.

Operator	Meaning	Example	Result
&&	Logical AND	10>5 && 5<10	true
||	Logical OR	10==5 || 10>5	true
!	Logical NOT	!(10>5)	false

4. Assignment Operators

- Used to assign values to variables.

Operator	Example	Result
=	x=10	x=10
+=	x+=10	x=x+10
-=	x-=10	x=x-10
*=	x*=10	x=x*10
/=	x/=10	x=x/10
%=	x%=10	x=x%10

5. Unary Operator

- Work on single operand.
- perform operation like increment and decrement.

Operator	Meaning	Example	Result
+	Positive	+10	10
-	negative	-10	-10
++	Increment	++10	11
--	Decrement	--10	9

6. Ternary Operator

- The ternary operator in java represented by the symnol "**? :**".
- Its also known as Conditional Operator.
- Shortcut for "if-else".

```
int a=10, b=20;
int max=(a>b)?a:b;
System.out.println(max);
```

7. Bitwise Operator

- Bitwise operator used to perform operations at bit level.
- Here we use binary values 5=0101 and 3= 0011

Operator	Meaning	Example	Result
&	AND	5&3	1
|	OR	5 | 3	7
^	XOR	5^3	6
~	Not	5~3	-6
<<	Left shift	5<<3	40
>>	Right shift	5>>3	0
>>>	unsigned Right shift	5>>>3	0

❖ Type Conversion and Type Casting

- Converting one data type to another data type call type conversion or type casting.

1) Type Conversion(/widening/Implicit casting)

- Also called type promotion.
- Java automatically converts a smaller data type into larger one (to avoid data loss).
- This happens implicitly(no need to write anything extra).
- Order of widening (smallest to largest).

Byte → Short → int → long → float → double

2) Type casting (narrowing/Explicit casting)

- Converting a large data type into smaller one
- Must do it manually using (type) its not safe way.
- May cause data loss or overflow.
- Risk of data loss.

Byte ← Short ← int ← long ← float ← double

❖ If-Else Statements

1) if:

- This decision control structure only deals with the true part of the condition and it works only with one condition.

```
If(condition)
{
        statment
}
```

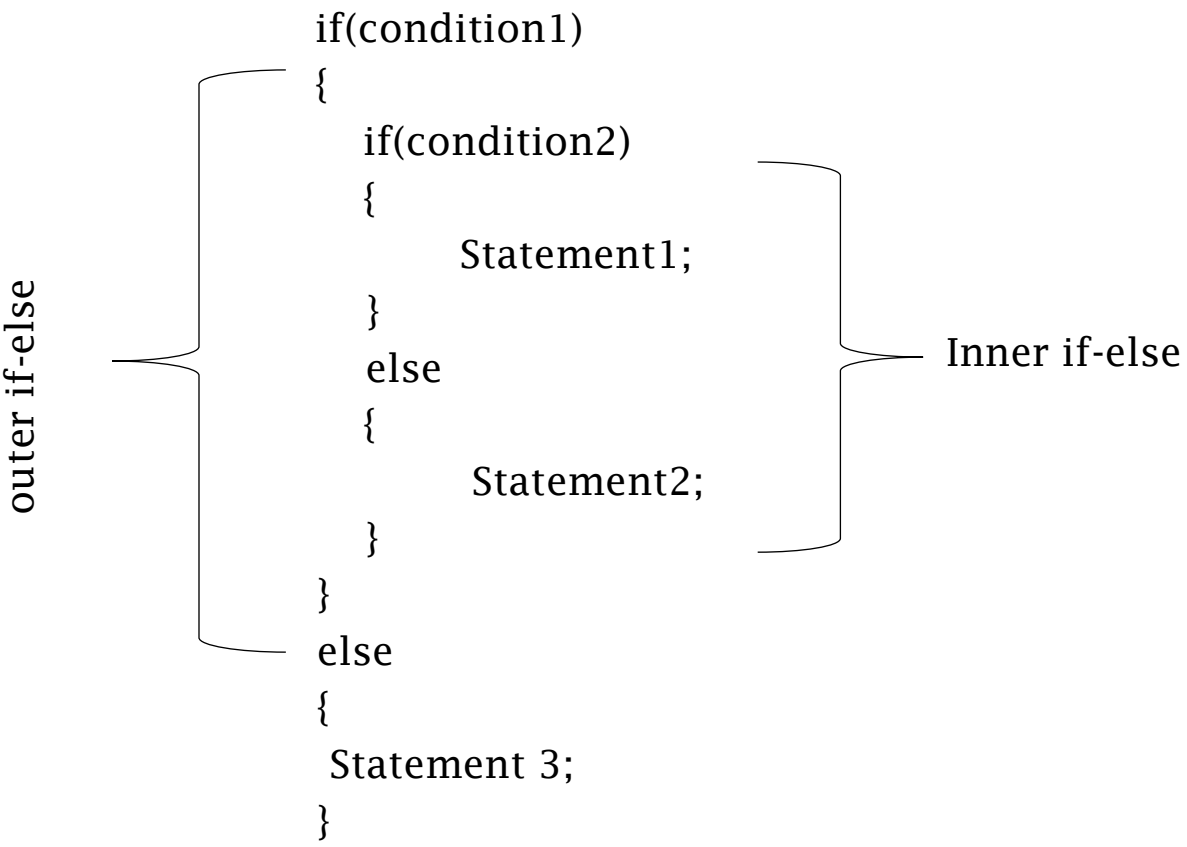
2) if-else:

- This decision control structure deals with the true and false part of the condition and it works only with one condition.
- If the condition is true then statements inside if are executed and if the condition is false then statements inside false are executed.

```
if(condition)
{
        statement1;
}
else
{
        statement 2;
}
```

3) Nested if:

- Nested if means one if inside another if. It is used when you have more than 1 condition.
- In Nested if, there are two if-else, one is inner if-else and other is outer if-else.



4) else-if ladder:

- If-elseif-elseif-elseif….else is known as else-if ladder.
- It is used when you have more than 2 conditions.
- In else-if ladder, when all the conditions are false then else part is executed.

```
if(condition1) {
      statement1;
} elseif(condition2) {
      statement2;
} elseif(condition3) {
      statement3;
}
 .... ....
else {
      statement4;
 }
```

❖ Switch Case Statements

- switch case is used with multiple options.
- In switch case, the value of the variable is passed which is compared with the different cases, the case which matches the value is executed.
- switch case have 4 keywords: switch, case, default, break
- break keyword is used to exit from the particular case
- When no case is executed, at that time default case is executed.

- **Syntax:**

```
switch(variable)
{
    case 1: statement1;
    break;
    case 2: statement2;
    break;
    case 3: statement3;
    break;
    case4: statement4;
    break;
    default: statement5;
    break;
}
```

❖ Loops

- When same task is to be performed multiple times, then in that case loop is used.

1) Entry Controlled Loop:

- When the condition is checked in the starting of the loop, it is known as entry controlled loop.
- for loop and while loop are known as entry controlled loop.

I. for loop:

- The FOR loop is the most popular loop
- It takes initialization, condition and increment or decrement of variable as arguments.
- **Syntax:**

```
for(initialization ;condition ;increment/decrement)
{
        Statements;
}
```

II. while loop:

- The while loop is java's most fundamental looping statement.
- It repeat a statement or block while its controlling expression is true
- **Syntax:**

```
Initialization;
while(condition) {
        Statements;
        Increment/decrement;
}
```

2) Exit Controlled Loop:

- When the condition is checked in the end of the loop, it is known as exit controlled loop.
- do-while loop is example of exit controlled loop
- In do-while loop, statements are executed atleast once without checking the condition.
- The only difference between while and do-while loop is do-while loop is at least execute body of the loop once, no matter if the condition is true or false.
- **Syntax :**

```
do {
   // Body of the loop
    }
while(condition);
```

❖ Break and Continue Keywords

- Java support basic three jump statement BREAK, CONTINUE.
- These statement transfer control to the another part of the program.

1) BREAK

- This statement is used to exit immediately from the loop or program.
- When the break statement is encountered in the program, the control directly moves to the end of the program.
- **Syntax:**

```
for(; ;)
{
    if(condition)
    break;
}
```

2) contine statement:

- This statement is used to continue back to the re-evaluation of the condition.
- Using continue statement, certain statements are bypassed.
- Sometimes it is useful to force an early iteration of a loop. That is, you might want to continue running loop, but stop processing the remainder of the code in its body for this particular iteration
- **Syntax:**

```
for(; ;)
{
    if(condition)
    continue;
}
```

❖ Defining a Class and Object in Java

1)class

- *A class is blueprint or template that defines state and behavior of objects and it can also include constructors, static member and nested class.*
- A class is a template from which objects are created. That is objects are instance of a class
- A class is container that groups variables and methods together in one unit.
- When you create a class, you are creating a new data-type. You can use this type to declare objects of that type.

2)object

- *An object in java is a runtime instance of a class that encapsulates state and behavior, and is created in heap memory using the class as a blueprint.*
- An object is an instance of a class.
- Each object contains its own copy of each variable defined by the class
- Class defines structure and behavior (data & code) that will be shared by a set of objects

❖ Object Creation, Accessing Members of the Class

- A class is blueprint or template, to use the class you need to create an object of that class.

```
ClassName ObjectName = new ClassName();
```

- ClassName is name of the class you want to create an object of.
- ObjectName is a Reference variable name you can choose any name.
- **New** keyword that allocates memory for the object.
- ClassName() is constructor special method used to initialize the object.
- To access member of class like variables and methods use below syntax

```
ObjectName.memberName
```

- **Example:** s1.name="vaibhav";
  s1.display();

❖ This and Super Keyword

1) This Keyword

- In java "this" keyword is a reference variable that refers to the current object.
- "this" in java is keyword that refers to the current object instance.
- Its used to call current class methods and fields
- To pass an instance of the current class as a parameter
- To differentiate between the local and instance variables.
- This keyword not used in static context
- In constructor first statement must be this If using **this** keyword.

2) Super Keyword

- The super keyword in java is a reference variable that is used to refer to the parent class when we are working with objects.
- Super keyword came into the picture with the concept of inheritance.
- It is used to call the parent class constructor.
- Also used for accessing parent class methods.
- Accessing parent class fields.
- In constructor first statement must be this If using **super** keyword.
- Not allowed in static methods/contexts.

❖ Defining Methods

- A method in java is block of code that perform a specific task.
- Methods help in code reusability and organixation.
- Instead of writing the same logic multiple times you put it inside a method and call it whenever needed.

```
Modifier returnType methodname(parameterList){
        //method body
        return value;
}
```

- **Syntax**
- Modifier define access(public,private,protected)
- Return type: type of value method returns if nothing return use void
- Methodname is any valid name
- **Parameter**
  - Parameter are variables that a method accepts as input
  - They let you pass data from outside into the method
- **ReturnType**
  - Return type tells what type if value a method will give you back after execution.
  - If method does not use anything use void

❖ Static Methods and Variables

1)Static variables

- Static variable belongs to the class rather than to any object of the class.
- It is declared using the keyword static.
- Only one copy of static variable exists, and it is shared among all objects of the class.
- It is created once in memory when the class is loaded, not every time an object is created.

2)Static Methods

- A static method also belongs to the class, not to any object.
- It can be called without creating an object.
- Declare using the keyword static.
- Static method can only access static variable and call other static methods directly.
- They cannot use **this** and **super** because they do not belong to any specific objects.

❖ Basics of OOP:

1) Encapsulation

- **In simple words encapsulation means wrapping data into single unit.**
- Encapsulation in Java is a mechanism of 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.
- **Advantage:**
  - The fields of a class can be made read-only or write-only.
  - class can have total control over what is stored in its fields.
- **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.

2) Inheritance

- Inheritance is the mechanism in java by which one class is allowed to inherits feature or properties(fields and methods) of another class.
- In java inheritance means means creating new class based on existing class.
- Subclass is a class which inherits the other class. It is also called a derived class, extended class, or child class.
- Superclass is the class from where a subclass inherits the features. It is also called a base class or a parent class
- **Advantage:** reuse the fields and methods of the existing class when you create a new class. You can use the same fields and methods already defined in the previous class.

➢ Types of Inheritance:

1) Single Inheritance

- When a class inherits another class, it is known as a single inheritance.

2) Multilevel Inheritance

- When there is a chain of inheritance, it is known as multilevel inheritance.
- In Multilevel inheritance, one class inherits from another class, another class inherits from next class and so on.

3) Hierarchical Inheritance

- When two or more classes inherits a single class, it is known as hierarchical inheritance.

4) Hybrid Inheritance

- Hybrid inheritance is the combination of more than 1 type of inheritance.

5) Multiple Inheritance

- Multiple inheritance is not supported in java by class. The reason is suppose A, B, and C are three classes. The C class inherits A and B classes. If A and B classes have the same method and you call it from child class object, there will be **ambiguity** to call the method of A or B class.

➢ Benefits of Inheritance in Java

- Common code written in a parent class can be reused in child classes, reducing duplication.
- Changes made in the parent class automatically reflect in child classes, making maintenance easier.
- Existing classes can be extended with new features without modifying the original class.
- Inheritance helps in achieving runtime polymorphism
- Since we reuse existing tested code, development becomes faster.
- Allows child classes to provide specific implementations of methods defined in the parent class

3) Abstraction

- Hiding internal details and showing essential information to user called abstraction.
- To achieve abstraction class should be declare with abstract keyword
- If function is abstract than class also should be abstract
- In abstract class we can declare abstract, default and static function.
- To define abstract method abstract class should be inherited to another class.
- By using child class object we can access static and default function of parent abstract class.
- If abstract class is inherited than abstract function compulsory override define in child class.
- We can not create object of abstract class
- Abstraction provide near 100% abstraction.

4) Polymorphism

- Same function name but having different functionalities called polymorphism.
- Polymorphism in Java is a concept by which we can perform a single action in different ways.
- Polymorphism is derived from 2 Greek words: poly and morphs. The word "poly" means many and "morphs" means forms. So polymorphism means many forms.
- **There are two types of polymorphism**

1) Compile time(Method overloading)

- It is also known as static polymorphism which is achieved by method overloading.
- Method overloading means methods with same name but different parameters.
- Method can be overloaded by change in number of arguments or change in type of arguments.
- **Method overloading is when a class declares multiple methods with the same name but different parameter list in same class.**

- Overloading does not depend in the return type of the method, two methods cannot be overloaded by just changing the return type.

2) Run-time(Method overriding)

- It is also known as dynamic polymorphism which is achieved by method overriding.
- It is a process in which a function call to the overridden method is resolved at Runtime.
- Same function name same parameter into different class, same return type and inheritance compulsory.
- Static methods can not be overridden.
- Name, parameter and return type must match the parent class method.

❖ Dynamic method dispatch

- Dynamic method dispatch is the mechanism where java decides which overridden method implementation to call at run time based on actual object not the reference type
- Means if a super class reference points to a subclass object, calling an instance method on that reference executes the subclass's overridden method.
- Upcasting directly related to dynamic method dispatch. Its usual setup that makes runtime polymorphism useful.

❖ Constructor

- A constructor is a special method which have same name as class name
- It is automatically called when object of class is created
- **Rules:**
  - Constructor name must be the same as its class name
  - A Constructor must have no explicit return type
  - A Java constructor cannot be abstract, static, final, and synchronized

❖ Types of Java constructor

- There are three types of constructors in Java:

1) Default constructor

- if you do not write any constructor in your class, the java compiler automatically provides one.
- This automatically created constructor is called default constructor
- A constructor with zero parameter or no argument is known as default constructor
- Its job is to initialize object with default values (0, null, false etc. depending on data type)
- If you write manually any constructor java compiler not provide a default constructor.

2) Parameterized constructor

- A constructor that takes arguments(parameters) to initialize an object with user-defined values known as parameterized constructor
- A constructor with parameters is known as parameterized constructor.
- Parameterized constructor allows you to pass values at the time of object creation.

3) Copy constructor

- A constructor that creates a new object by copying the values of another object of the same class.
- Java does not support copy constructors directly, but they can be **emulated** by creating a constructor that copies values.
- Java does not provide a built-in copy constructor(like c++) but you can define your own.
- If you want to copy an object you must define your own constructor.
- Copy constructor usually takes an object of the same class as a parameter.
- **Shallow copy:** only copies references for objects(e.g. arrays, collections).
- **Deep copy:** creates new copies of referenced objects as well.

❖ Constructor Overloading

- Constructor overloading means defining multiple constructor in the same class with different parameter lists(different number or type of parameters).
- It provides flexibility to create an objects in different ways.
- Constructors must have same names but different parameters.
- You can call one constructor from another using this() keyword(constructor chaining).

❖ Object Life Cycle and Garbage Collection

- The **object life cycle** in Java represents the different stages an object passes through, from its creation to its destruction by the JVM.

1) Object Creation

- Objects are created using the **new** keyword.
- Memory for the object is allocated in the heap area, and a reference to the object is stored in a variable.

```
Student s1 = new Student();
```

2) Object Usage

- Once created, the object can be used to call methods and access data members.
- Multiple references can also point to the same object.

```
s1.displayDetails();
```

3) Object Becoming Unreachable

- An object becomes unreachable when there are no active references pointing to it.
- This can happen when a reference is set to null reassigned to another object, or when the reference goes out of scope.
- At this stage, the object is no longer accessible by the program.

4) Object Destruction (Garbage Collection)

- In Java, memory management is handled automatically through a process called **Garbage Collection**

- Garbage Collection is the process by which the **Java Virtual Machine (JVM)** identifies and removes objects from the heap memory that are no longer reachable by any reference. This frees up memory space and prevents memory leaks.
- Java does not require manual memory management. The GC reclaims memory and makes it available for future object creation.
- Improves efficiency and reliability of programs.

❖ Arrays

- Array is the collection of elements that have same data type and elements are store in contiguous memory location. All the elements of array share same array name
- The main concept of array is index, and index start from zero index.
- Arrays in java are objects like all other objects in java, arrays implicitly inherit from the java.lang.Object class.
- Arrays have built-in length property, which provides the number of elements in the array.

1) One Dimensional Array

- The array having only one dimension is known as 1-d array.
- a one-dimensional array is like a list of elements stored in a single row.
- It is declared using square brackets [].
- Elements are accessed using single index.
- Memory is allocated in continuous block.
- One dimensional array; linear list, single index.

2) Multi-dimensional Array

- a multi dimensional array is an array of arrays.
- The most common is the two dimensional array, which represents a table(rows and columns).
- Elements are accessed using multiple indexes.
- Java supports arrays with more than 2 dimensions (3D,4D,etc.) but 2D is most commonly used.

❖ String

- String represents a sequence of characters.
- Strings are immutable, meaning once created they can not be modified.
- When you perform an operation like(concatenation), a new string object is created.
- Strings are widely used in java because they are safe, secure and efficient in memory handling.
- Java maintain a string constant pool(SCP) to store string literals for memory efficiency.
- Commonly used methods: length(), charAt(), substring(), equals(), compareTo(), toUpperCase(), toLowerCase(), trim().
  - **Length() :** this method is built-in method of the string class and its return the length of the string.
  - **charAt() :** this method return the character at the specified index in a string.
  - **Substring() :** in java substring method is used to extract portion of original string.
  - **equals() :** this method check if two string have the same content.it return Boolean values.
  - **compareTo() :** it is used to compare two objects (usually string or numbers) in order to determine their lexicographical(dictionary) order or natural ordering.
  - **toUpperCase() :** this method return the string in uppercase letters.
  - **toLowerCase() :** this method return the string in lowercase letters
  - **trim() :** the trim method is used to get rid of any extra spaces or control characters at beginning and end of string.

❖ StringBuffer

- A **mutable** sequence of characters. All **methods are synchronized** → thread-safe, usually slower.
- Unlike string it allows modification of content without creating new objects
- All important methods like append(), insert(), delete(), reverse() can modify the string directly.

- It is **synchronized**, which means it is **thread-safe** (can be used safely in multithreaded environments).
- Default capacity is 16 characters, but it grows dynamically when needed.
- Slightly **slower** than StringBuilder due to synchronization overhead.
- Its use when Building/Modifying text **from multiple threads** sharing the same buffer.

❖ StringBuilder

- StringBuilder is also a **mutable** sequence of characters like StringBuffer.
- Provides the same methods: append(), insert(), delete(), reverse().
- It offers similar functionality to StringBuffer, but without thread safety
- StringBuilder is not synchronized, making it faster and more efficient than StringBuffer in single-threaded applications.
- Best choice for performing repeated modifications in strings (such as concatenation in loops).

❖ Array of Objects

- An **array of objects** is a collection that can hold multiple references of objects of the same class.
- Just like arrays store primitive data (e.g., int[] for integers), arrays can also store objects**.**
- Objects inside the array are stored as references (addresses), not as actual objects.
- By default, if an array of objects is created but not initialized, its elements contain null.
- Each element of the array must be individually created using the new keyword.
- Array of objects is useful for managing large groups of similar data, like students, employees, products, etc.
- Objects in the array can be accessed using index values (arr[0], arr[1] ...).
- Helps in organizing and processing multiple objects together efficiently.

❖ Method Hiding

- Method hiding occurs when a static method in a subclass has the same signature as a static method in its superclass.
- It is not method overriding, because overriding only applies to instance (non-static) methods.
- In method hiding, the method resolution happens at compile time (based on the reference type), not at runtime.
- The version of the hidden method that gets called depends on the class type of the reference, not the object created.
- Method hiding does not support polymorphism, unlike method overriding.
- If a subclass defines a static method with the same signature as in its parent, the parent's method is simply hidden, not overridden.

❖ Interfaces

- An interface in Java is a collection of abstract methods (by default) and constants.
- It is used to achieve abstraction and multiple inheritance in Java.
- A class uses the implements keyword to provide the implementation of an interface.
- Interfaces cannot have instance variables or constructors.
- From Java 8 onward, interfaces can also contain default methods and static methods.

❖ Multiple Inheritance in Java

- Java does not support multiple inheritance with classes (to avoid ambiguity problem, known as the Diamond Problem).
- But Java supports multiple inheritance through interfaces.
- A class can implement multiple interfaces at the same time.
- This allows a class to inherit behavior from more than one source.

❖ Implementing Multiple Interfaces

- A class can implement multiple interfaces by separating them with a comma (,) after the implements keyword.
- While implementing, the class must override all abstract methods of all interfaces.
- If two interfaces have methods with the same signature, only one implementation is provided in the class.
- This is how Java achieves multiple inheritance safely.

❖ Access Modifiers

1) Private

- The most restrictive access modifier.
- Members declared private are accessible only within the same class.
- Cannot be accessed from outside the class, not even by subclasses.
- Used to achieve data hiding.

2) Default (No Modifier)

- When no modifier is specified, it is considered default access.
- Members with default access are accessible only within the same package.
- Not accessible outside the package.
- Useful for package-level visibility.

3) Protected

- Members declared protected are accessible Within the same package, and In subclasses (even if they are in different packages).
- Used when we want to allow inheritance with controlled access.

4) Public

- The least restrictive access modifier.

- Members declared public are accessible from anywhere in the program.
- Provides the widest scope of visibility.

❖ Package

- A package in Java is a way to group related classes, interfaces, and sub-packages together.
- Prevent naming conflicts by allowing classes with the same name to exist in different packages
- Make it easier to organize, locate and use classes, interfaces and other components
- Provide controlled access for Protected members that are accessible within the same package and by subclasses. Default members (no access specifier) are accessible only within the same package

❖ Built-in Package

- Built-in packages in Java are predefined libraries provided by the Java Development Kit (JDK) that contain a wide range of classes and interfaces.
- These packages are essential for performing common programming tasks and are categorized based on their functionality.
- java.lang – String, Math, Object classes.
- java.util – Collections, Scanner, Date.
- java.io – Input/Output classes like File, BufferedReader.
- java.sql – Database connectivity (JDBC).
- Built-in packages reduce coding effort by providing ready-made classes.

❖ User-Defined Packages

- User-defined packages in Java are custom namespaces created by developers to organize and manage related classes, interfaces, and sub-packages.
- It can be imported into other classes and used in the same way as we use built-in packages.
- Steps to create:

1. Define package at the top of the program.
2. Compile using javac -d . ClassName.java.
3. Import and use the package in another class.

❖ Importing Packages and Classpath

- To use classes from other packages, the import keyword is used.
- Import a single class → import java.util.Scanner;.
- Import whole package → import java.util.*;
- Use fully qualified name → java.util.Scanner sc = new java.util.Scanner(System.in);.
- Classpath tells Java where to find .class files and packages.
- By default, the current directory (.) is included in classpath.
- Can be set in two ways:
  - Temporary (command line): javac -cp . Test.java.
  - Permanent**:** by setting the CLASSPATH environment variable.
- Needed especially for user-defined packages.

❖ Exception

- An exception is an unwanted or unexpected event that disrupts the normal flow of a program during execution.
- Exceptions are represented as objects in Java, which contain information about the error.
- All exceptions are part of the Java Exception Hierarchy under the class Throwable.
- Types of Exceptions

1) Checked Exceptions

- Checked exceptions are the exceptions that are checked by the compiler at compile-time.
- These exceptions must be either handled using try-catch or declared using throws in the method signature.
- If a checked exception is not handled, the program will not compile.
- Checked exceptions are also called compile-time exceptions**.**
- Examples :
  - ✓ IOException → when file is not found or input/output fails.
  - ✓ SQLException → database-related errors.
  - ✓ ClassNotFoundException → when a class is not found at runtime.

2) Unchecked Exceptions

- Unchecked exceptions are those exceptions that are not checked by the compiler at compile-time, they occur only at runtime.
- They are subclasses of RuntimeException class.
- If an unchecked exception occurs and is not handled, the program terminates abnormally.
- They usually happen due to programming mistakes like wrong logic or invalid data.
- It is not mandatory to handle unchecked exceptions using try-catch or throws.
- Unchecked exceptions are also called runtime exceptions.
- **Examples**
  - ✓ ArithmeticException → division by zero.
  - ✓ NullPointerException → using a null object reference.
  - ✓ ArrayIndexOutOfBoundsException → accessing array with invalid index.
  - ✓ NumberFormatException → invalid conversion of string to number.

❖ **try, catch, finally, throw, throws**

➢ java provide five keywords to handle exception handling

1) try

- The try block contains the code that might throw an exception.
- It is followed by at least one catch or finally block.
- **try {**
  // risky code
  }

2) Catch

- The **catch block** is used to handle the exception thrown by the try block.
- Each catch block can handle a specific type of exception.
- Multiple catch blocks can be used.
- **catch(ExceptionType e) {**
  // handling code
  }

3) finally

- The finally block always executes whether an exception occurs or not.
- Commonly used to release resources (like closing files, database connections).
- It is optional but if present, it must follow try-catch.
- **finally {**
  // cleanup code
  }

4) throw

- The throw keyword is used to explicitly throw an exception inside a method or block of code.
- It is followed by an exception object.
- **throw new ExceptionType("Error Message");**

5) throws

- The **throws keyword** is used in the method signature to declare exceptions that a method may throw.
- It informs the caller method to handle the exception.
- **returnType methodName() throws ExceptionType {**
  // code
  }

❖ Custom Exception Classes

- A custom exception (also called user-defined exception) is a class created by the programmer to represent specific error conditions in an application.
- **In Java, all custom exceptions must extend either:**
  - ✓ Exception (for checked exceptions)
  - ✓ RuntimeException (for unchecked exceptions)
- Creating a custom exception improves readability and makes the error message more.
- **Steps to create a custom exception:**
  - ✓ Define a class that extends Exception or RuntimeException
  - ✓ Provide a constructor to pass a custom error message.
  - ✓ Use throw keyword to explicitly throw the custom exception.

❖ Introduction to Threads

- A thread is a lightweight sub-process in Java. It is the smallest unit of execution within a program.
- A Java program by default has at least one thread, called the main thread.
- Threads are used to perform multiple tasks simultaneously (multithreading).
- Multithreading improves performance by allowing CPU to execute multiple tasks concurrently.
- Each thread has its own Program Counter, Stack, Local variables but shares heap memory with other threads.
- create a thread in Java By extending the Thread class and By implementing the Runnable interface.
- **There are two main ways to create thread :**

1) By Extending the Thread Class

- Create a class that extends thread.
- Using thread class is Simple to use.
- Override the run() method to define the task for the thread.
- Create an object of the class and call start() to begin execution.
- Not flexible, because Java supports single inheritance → if a class already extends another class, it cannot extend **thread**.

2) By Implementing the runnable Interface

- Create a class that implements Runnable interface.
- Override the run() method.
- Pass the object of this class to a Thread object and call start().
- More flexible than extending Thread class.
- Since class implements an interface, it can still extend another class**.**
- Preferred in real-world applications.

❖ Thread Life Cycle

- A **thread life cycle** in Java refers to the different states a thread passes through during its execution.
- Java provides the `Thread` class which defines methods to control the life cycle (like start(), sleep(), join()).

1) New (Created State)

- Whenever a new thread is created, it is always in the new state
- When a thread object is created using **Thread t = new Thread();**.
- At this stage, the thread is not yet running.
- Moves to Runnable state when start() is called.

2) Active State

- When a thread invokes the start() method, it moves from the new state to the active state
- The active state contains two states within it: one is runnable, and the other is running.

- **Runnable state**
- A thread that is ready to run is then moved to the runnable state.
- In the runnable state, the thread may be running or may be ready to run at any given instant of time.
- It is the duty of the thread scheduler to provide the thread time to run, i.e., moving the thread the running state.

- **Running State**
- When the CPU scheduler selects a thread, it moves from the runnable to the running state.
- The run() method code is executed.

3) Blocked/Waiting State

- Whenever a thread is inactive for a span of time (not permanently) then, either the thread is in the blocked state or is in the waiting state.
- For example, a thread (let's say its name is A) may want to print some data from the printer. However, at the

same time, the other thread (let's say its name is B) is using the printer to print some data.
- Therefore, thread A has to wait for thread B to use the printer. Thus, thread A is in the blocked state.

4) Timed Waiting State
- Sometimes, waiting for leads to starvation. For example, a thread (its name is A) has entered the critical section of a code and is not willing to leave that critical section.
- In such a scenario, another thread (its name is B) has to wait forever, which leads to starvation. To avoid such scenario, a timed waiting state is given to thread B.
- Thus, thread lies in the waiting state for a specific span of time, and not forever.
- A real example of timed waiting is when we invoke the sleep() method on a specific thread.
- The sleep() method puts the thread in the timed wait state
- After the time runs out, the thread wakes up and start its execution from when it has left earlier

5) Terminated state
- When a thread has finished its job, then it exists or terminates normally.
- Abnormal termination: It occurs when some unusual events such as an unhandled exception or segmentation fault.
- A terminated thread cannot be restarted.

❖ Synchronization

- Synchronization is the process of controlling access to shared resources by multiple threads to prevent data inconsistency.
- When multiple threads access the same object at the same time, problems like race conditions may occur.
- Synchronization ensures that only one thread at a time can access the shared resource.
- Java provides synchronization using the synchronized keyword.

- **Advantages :** Data consistency, prevents race conditions.
- **Disadvantage**: Slows down performance because only one thread can access the resource at a time.

❖ Inter-thread Communication

- Inter-thread communication allows threads to communicate and cooperate with each other.
- It is mainly used when multiple threads are working on shared data**.**
- It helps in avoiding busy waiting (threads constantly checking conditions).
- Java provides methods for inter-thread communication inside the Object class:
  - wait() → makes the thread wait until another thread notifies it.
  - notify() → wakes up one waiting thread.
  - notifyAll() → wakes up all waiting threads.
- These methods can only be called inside a synchronized block/method**.**
- Used in problems like producer-consumer, where one thread produces data and another consumes it.

❖ Introduction to File I/O in Java

- File I/O (Input/Output) in Java is the process of reading data from files and writing data to files using Java's standard library.
- Java provides the java.io package and the java.nio package to perform file operations.
- File I/O useful for "Storing program output permanently", "Reading input from external sources", "Handling large amounts of data outside program memory".
- **File Handling Operations**:
  - Create a file.
  - Write data to a file.
  - Read data from a file.
  - Delete a file.
- Since file operations deal with external devices, they often throw checked exceptions like IOException, so handling is required.

❖ FileReader

- FileReader is a class in the java.io package used to read data from text files (character by character).
- It is a character stream class, which means it reads data in the form of characters instead of bytes.
- FileReader extends InputStreamReader, which in turn extends Reader.
- Since file operations are risky, FileReader methods may throw IOException (checked exception), so they must be handled.
- Commonly used for reading text files (.txt, .csv, etc.) where data is character-based.

❖ FileWriter

- FileWriter is a class in the java.io package used **to** write character data into text files.
- It is a character stream class, meaning it writes data in the form of characters (Unicode) instead of raw bytes.
- FileWriter extends OutputStreamWriter, which in turn extends Writer.
- Since file operations are external, methods may throw IOException (checked exception), which must be handled.
- Commonly used for writing data into text files (.txt, .csv, .log, etc.)**.**

❖ BufferedReader

- BufferedReader is a class in the java.io package that is used to read text from character input streams efficiently**.**
- It uses an internal buffer (default size = 8192 characters), which makes reading from a file or stream faster compared to directly using FileReader.
- **Advantages**:
  - Faster than FileReader because of buffering.
  - Provides readLine() method which FileReader does not have.
  - Reduces the number of disk access operations.
- BufferedReader extends Reader.
- Methods may throw IOException (checked exception).

- Mostly used for reading large text files or when reading line by line (like logs, configuration files, or CSV files).

❖ BufferedWriter

- BufferedWriter is a class in the java.io package used to write text efficiently to character output streams.
- It uses an internal buffer (default size = 8192 characters) which makes writing to a file or stream faster compared to using FileWriter directly.
- **Advantages:**
  - Faster than FileWriter due to buffering.
  - Provides newLine() method for easy line breaks.
  - Reduces the number of direct disk write operations.
- Methods may throw IOException (checked exception).
- Commonly used for writing large text files, logs, reports, or line-based data.

❖ Serialization

- Serialization is the process of converting an object into a byte stream so that it can be stored in a file, database, or transmitted over a network.
- It allows saving the state of an object for later use.
- Only classes that implement Serializable can be serialized.
- transient keyword → prevents some fields from being serialized
- In Java, to serialize an object, the class must implement the Serializable interface (marker interface from java.io).
- The ObjectOutputStream class is used for serialization.
- writeObject(Object obj) → writes the object to an output stream.
- **Use Cases:**
  - Saving objects to files.
  - Sending objects over the network (RMI, sockets).
  - Caching data.

❖ Deserialization

- Deserialization is the process of reconstructing an object from a byte stream.

- serialVersionUID → used to verify compatibility during deserialization.
- It restores the original object's state from the stored data.
- In Java, the ObjectInputStream class is used for deserialization.
- readObject() → reads the object from the input stream.
- **Use Cases**:
  - Retrieving saved objects from files.
  - Receiving objects over a network.

❖ Introduction to Collections Framework

- The Collections Framework in Java is a set of classes and interfaces that provide a ready-made architecture to store, manipulate, and process groups of objects**.**
- A collection in Java is a group of individual objects that are treated as a single unit.
- In Java, a separate framework named the "Collection Framework" was defined in JDK 1.2, which contains all the Java Collection Classes and interfaces
- All collection classes and interfaces are present in the java.util package.
- Instead of working with arrays (which have fixed size and limited functionality), collections provide dynamic, flexible, and efficient ways to handle data.
- Collections provides static methods like sort(), reverse(), shuffle(), etc.

❖ List Interface

- A List is an ordered collection that stores elements in a sequence.
- It allows duplicate elements, meaning the same value can appear multiple times.
- The insertion order is preserved, so elements are retrieved in the same order they were added.
- Elements can be accessed and modified using indexes, similar to arrays.
- Common implementations include ArrayList (dynamic array), LinkedList (doubly linked list), Vector, and Stack.

❖ Set Interface

- A Set is a collection that represents a group of unique elements only.
- It does not allow duplicate elements, ensuring every element is stored only once.
- The collection is generally unordered, but some implementations maintain order (like LinkedHashSet).
- Some implementations like TreeSet maintain elements in sorted order automatically.
- Common implementations are HashSet, LinkedHashSet, and TreeSet.

❖ Map Interface

- A Map stores data in the form of key–value pairs where each key maps to a specific value.
- Keys in a Map are always unique, but values can be repeated or duplicate.
- Unlike List and Set, Map does not extend the Collection interface; it has its own hierarchy.
- It is widely used in applications that require a dictionary, lookup table, or data mapping structure.
- Common implementations include HashMap (fast, unordered), LinkedHashMap (insertion order maintained), TreeMap (sorted by keys), and Hashtable (legacy, synchronized).

❖ Queue Interface

- A Queue is a collection used for storing elements before processing, often for scheduling tasks.
- By default, it follows the FIFO (First-In-First-Out) principle, but some variations use priority rules.
- A Queue allows duplicate elements, so the same value can appear multiple times.
- Specialized queues like PriorityQueue order elements according to priority rather than insertion order.
- Common implementations are LinkedList (works as a queue), PriorityQueue, and ArrayDeque (double-ended queue).

❖ ArrayList

- ArrayList is a resizable array implementation of the List interface.
- It maintains insertion order and allows duplicate elements.
- Elements can be accessed directly using an index, which makes searching very fast.
- Insertion and deletion are slower in the middle because elements need to be shifted.
- It is best suited when the main operation is searching rather than frequent insertions and deletions.
- The size() method is used to find size of ArrayList

❖ LinkedList

- LinkedList is a doubly linked list implementation of the List and Deque interfaces.
- It maintains insertion order and allows duplicate elements.
- Accessing elements by index is slower than ArrayList, because traversal is required.
- Insertion and deletion are faster, especially in the middle, since no shifting of elements is needed.
- It is best suited when the application requires frequent insertion and deletion operations.
- It implements a doubly linked list data structure where elements are not stored at contiguous memory.

❖ HashSet

- HashSet is a collection that implements the Set interface using a hash table.
- It does not allow duplicate elements, ensuring uniqueness of data.
- It does not maintain insertion order, the order of elements is unpredictable.
- Operations like insertion, deletion, and search are generally very fast (O(1)) due to hashing.
- It is best suited when we want to store unique elements without caring about order.

❖ TreeSet

- TreeSet is a collection that implements the Set interface using a TreeMap internally.
- It does not allow duplicates, and all elements must be unique.
- It maintains elements in sorted order (natural ordering or by a custom comparator).
- Operations like insertion, deletion, and search take O(log n) time, since it is based on a balanced tree.
- It is best suited when we want to store unique elements in a sorted manner.

❖ HashMap

- HashMap is a Map implementation that stores data in key-value pairs using a hash table.
- It allows one null key and multiple null values.
- Keys are unique, but values can be duplicate.
- It does not maintain insertion order, as elements are arranged based on hash codes.
- It is best suited for fast lookups and storing data in key-value form.

❖ TreeMap

- TreeMap is a Map implementation that stores data in key-value pairs using a Red-Black Tree.
- It does not allow null keys, but null values are allowed.
- Keys are stored in sorted order, either natural or by a custom comparator.
- Operations like insertion, deletion, and search take O(log n) time.
- It is best suited when we need a sorted map of key-value pairs.

❖ Iterator

- Iterator is a universal cursor used to traverse elements of a Collection (List, Set, Queue).
- It is present in the java.util package.

- It works in a forward-only direction (from first element to last element).
- It can be used with List, Set, and Queue, but not with Map directly.
- Methods available:
  - hasNext() → returns true if more elements are available.
  - next() → returns the next element.
  - remove() → removes the last returned element from the collection.
- It is a fail-fast cursor, meaning it throws ConcurrentModificationException if the collection is modified while iterating.

❖ ListIterator

- ListIterator is a specialized cursor for List collections only (like ArrayList, LinkedList).
- It is present in the java.util package.
- Unlike Iterator, it can traverse in both directions (forward and backward).
- It can only be used with List implementations, not with Set or Queue.
- Extra methods compared to Iterator:
  - hasPrevious() → checks if there are elements in backward direction.
  - previous() → returns the previous element.
  - nextIndex() → returns the index of the next element.
  - previousIndex() → returns the index of the previous element.
- It is also a fail-fast cursor, throwing ConcurrentModificationException on structural modifications outside iterator methods.

❖ Streams in Java

- in Java, streams are used to perform input and output (I/O) operations.
- A stream is a sequence of data that flows from a source to a destination.
- Streams can handle data like bytes, characters, or objects, depending on the type of stream.

- They are present in the java.io package.
- Streams in Java are mainly divided into two categories: Byte Streams and Character Streams.
- Byte Streams → Use InputStream and OutputStream (deal with raw binary data).
- Character Streams → Use Reader and Writer (deal with text/characters).

❖ InputStream

- InputStream is an abstract class in java.io package.
- It is the superclass of all classes representing an input byte stream.
- It is used to read data (in bytes) from a source, such as a file, keyboard, or network.
- Important methods:
  - int read() → reads one byte of data.
  - int read(byte[] b) → reads data into a byte array.
  - void close() → closes the input stream and releases resources.
- Common subclasses:
  - FileInputStream (reads data from files).
  - BufferedInputStream (adds buffering for efficiency).
  - ObjectInputStream (reads objects).

❖ OutputStream

- OutputStream is an abstract class in java.io package.
- It is the superclass of all classes representing an output byte stream.
- It is used to write data (in bytes) to a destination, such as a file, console, or network.
- Important methods:
  - void write(int b) → writes one byte of data.
  - void write(byte[] b) → writes an array of bytes.
  - void flush() → forces any buffered output to be written immediately.
  - void close() → closes the stream and releases resources.

❖ Reading Data Using InputStream

- To read data, we use classes that extend InputStream such as FileInputStream, BufferedInputStream, etc.
- First, an InputStream object is created and connected to a data source (like a file).
- The method read() is used to read data byte by byte.
- The method read(byte[] b) can be used to read multiple bytes at once into a byte array.
- When there is no more data to read, the read() method returns -1.
- After reading, the close() method must be called to release system resources.

❖ Writing Data Using OutputStream

- To write data, we use classes that extend OutputStream such as FileOutputStream, BufferedOutputStream, etc.
- First, an OutputStream object is created and connected to a destination (like a file).
- The method write(int b) is used to write a single byte to the output stream.
- The method write(byte[] b) is used to write an entire byte array at once.
- The flush() method is used to force any buffered data to be written immediately.
- After writing, the close() method must be called to release system resources.

❖ Handling File I/O Operations in Java

- File I/O in Java is performed using the java.io package, which provides classes to read from and write to files.
- A file in Java is represented by the File class, which allows creating, deleting, and checking file properties.
- To read data from a file, we use input classes like FileInputStream, FileReader, or BufferedReader.
- To write data to a file, we use output classes like FileOutputStream, FileWriter, or BufferedWriter.

- File I/O can be done using byte streams (InputStream, OutputStream) for binary data (like images, audio, video).
- For text data, we use character streams (Reader, Writer), which handle characters instead of raw bytes.
- When performing I/O operations, always use try-catch blocks to handle exceptions like IOException or FileNotFoundException.
- Java also provides the try-with-resources statement to automatically close file streams and avoid memory leaks.
- Buffered classes (like BufferedReader, BufferedWriter) improve performance by reducing the number of I/O operations.
- For object storage, Java provides Serialization and Deserialization using ObjectOutputStream and ObjectInputStream.
- File handling includes tasks such as creating new files, writing data, reading data, appending data, and deleting files.
- Always close file streams using the close() method after operations to free system resources.