Assignments/Tasks (TechOps)

Concept: OOPS And Collection

Covered the types of Inheritance.

Types: Single, Multiple, Multilevel, Hierarchical and Hybrid.

Task Given:

1. Create a class with constructor (default and parameterized) and create another class (Sub-class) with constructor and check how it execute.
2. Create a class with method definition and create another class, extend the class and call the methods.
3. Create multiple classes and extend the classes (example: Class A {}, Class B extends A { }, Class C extends D ….).
4. Create a Super class and create two sub-class and extend the Super class, override the method in super class and implement it. (Defined the use of **super** Keyword)

[Note: Java does not support Multiple. Instructed how to achieve the Multiple in Java using Inheritance.**]**

Topic: Polymorphism

Explained the Concept with examples.

Method Overloading (Compile time) and Overriding (Run time).

Task Given:

1. Create a class and define a default constructor. Define the same constructor with parameters and execute the program.
2. Create a call and define methods. Define the same methods with different parameter and execute the program.
3. Create a Super class and define a method. Create a Sub-class and override the method of the Super-class in the Sub-class.

[Try the method Overriding with multiple classes by extending the classes.]

Topic: Abstraction

Hiding Information and only shows the necessary details.

Example: Interface and Abstract Class

Task Given:

1. Create an Interface Class named Loan and declare methods. Create a class named ICICIBank and implement the Interface class. Also Create another class named SBIBank and implement the Interface class Loan. Create a class named Customer and extend the Bank class and calculate the Interest amount.

2. Create two Interface Classes named Malayalam and TamilMovies. Create three Theatre class, where one class should implement Malayalam, another one should implement TamilMovies and the last one should implement both the Interface classes. Create three customer class and extend the theatre classes, the output should print the cost and the movie.

3. Create an Interface class named Veg and NonVeg. Declare the methods in both classes. Create a new class named Hotel and implement both the interface classes. Create a new class named Customer and extend the Hotel class and execute the program.

4. Create an Interface class named Shape and declare the draw() method. Create multiple classes like Rectangle, Square, Triangle, Circle…and implement the Shape class and define the draw()

method. Check the object referencing on all the classes and check which methods can be accessed and which is not.

[Also try the object creation by referencing the interface class, extended class and the main class.**]**

Note:

An Interface which has single abstract method is called Functional Interface.

The function definition can be done using the Lambda Function.

Example:

```
Class Bird {

//methods

}

Class A {

main() {

Bird b=()->{System.out.println(“ “);};

}

}
```

Topic: Encapsulation

Binding or wrapping the data and code in a single unit.

Example: Declaring the variable or method as private

Explained how to set the values and get the values using Java Bean and the constructor.

Task Given:

1. Create a class and declare the variables or methods as private. Create another class and use the get and set methods to access the variables and methods. Print the value using the object referencing or by toString() method.

2. Create a class and declare the variables or methods as private. Create another class and create the object using constructor. Declare the parameterized constructor (method) in the class which has private variables and methods. Pass the data or input to the constructor and execute the program.

Note:

No need to manually create the toString(), get and set methods and the Constructor. Eclipse has the option called source which will automatically generate the methods.

Topic: Project

Task Given:

1. To understand the document and create the package and classes as specified in the document.
2. Use Array or ArrayList , use Scanner or JoptionPane to set and get the input.

1. Task Description:

Create a simple program that models animals in a zoo.
Use both **an interface** and **an abstract class**.

Requirements:

1. Create an **interface** named `Animal` with the following methods:
   - `void eat();`
   - `void sleep();`
2. Create an **abstract class** named `Mammal` that:
   - Implements the `Animal` interface.
   - Has an instance variable `String name;`
   - Includes a constructor that takes a `name`.
   - Implements the `sleep()` method (e.g., prints `"name is sleeping..."`).
   - Declares an abstract method `void makeSound();`
3. Create two subclasses of `Mammal`:
   - `Dog`
   - `Cat`

   Each should implement the `eat()` and `makeSound()` methods differently.
4. In a `Main` class, create objects of `Dog` and `Cat` and call all their methods.

Example Output:

```
Rex is eating bones.
Rex says: Woof!
Rex is sleeping...

Misty is eating fish.
Misty says: Meow!
Misty is sleeping...
```

2. Basic Task – Shape Interface

Goal: Understand how interfaces define behavior.

Description:

1. Create an interface `Shape` with methods:
   - `double area();`
   - `double perimeter();`
2. Implement two classes: `Circle` and `Rectangle`.
3. Use constructors to set dimensions.
4. In the main program, create objects of both and print their area and perimeter.

Example Output:

```
Circle area: 78.5
Circle perimeter: 31.4
Rectangle area: 20.0
Rectangle perimeter: 18.0
```

3. Interface + Abstract Class Combination

Goal: Use both together to build layered design.

Description:

1. Create an **interface** `Payment` with:
   - `void pay(double amount);`
2. Create an **abstract class** `OnlinePayment` that implements `Payment`:
   - Has field `String accountId;`

    - Concrete method `void connect()` → prints connection message.
    - Abstract method `void authenticate();`
3. Create subclasses:
    - `CreditCardPayment`
    - `UPIPayment`

    Each implements `authenticate()` and `pay()` differently.
4. In `main()`, demonstrate different payments.

4.Challenge Task – Smart Home System

Goal: Design a multi-layered, interface-driven system.

Description:

1. Create interface `SmartDevice`:
    - `void turnOn();`
    - `void turnOff();`
    - `String getStatus();`
2. Create abstract class `Appliance` implementing `SmartDevice`:
    - Field: `String brand`
    - Concrete `getStatus()` method → returns status string.
    - Abstract methods: `void adjustSetting(int level);`
3. Subclasses:
    - `SmartLight`
    - `SmartThermostat`
4. Each implements behavior differently.
5. In `main()`, simulate turning on devices and adjusting setting

5. Basic Inheritance – Person & Student

Goal: Understand simple inheritance (using `extends`).

Description:

1. Create a base class `Person` with:
    - Fields: `name`, `age`
    - Method: `void displayInfo()` → prints name and age.
2. Create subclass `Student` that:
    - Adds field: `String course`
    - Adds method: `void study()` → prints that the student is studying.
3. In `main()`, create a `Student` object and call both `displayInfo()` and `study()`.

Example Output:

```
Name: Alice
Age: 20
Alice is studying Computer Science.
```

6, Using `super` and Constructor Inheritance – Employee System

Goal: Learn constructor chaining and `super()`.

Description:

1. Base class `Employee`:
    - Fields: `name`, `salary`
    - Constructor to initialize both.
    - Method `displayInfo()`
2. Subclass `Manager`:
    - Additional field: `department`
    - Constructor calls `super(name, salary)`
    - Override `displayInfo()` to include department.
3. In `main()`, create both `Employee` and `Manager` objects.

Example Output:

```
Name: John, Salary: 50000.0
Name: Sarah, Salary: 80000.0, Department: IT
```

7. Inheritance + Overriding + super Keyword

Scenario:
Create a small hierarchy for a **Library System**.

Description:

1. Base class `LibraryItem`:
   - Fields: `title`, `author`
   - Method: `void displayInfo()`
2. Subclass `Book`:
   - Adds `int pages`
   - Overrides `displayInfo()` but also calls `super.displayInfo()`
3. Subclass `Magazine`:
   - Adds `String issueMonth`
4. In `main()`, create a few objects and display info.

Example Output:

```
Title: The Hobbit, Author: Tolkien, Pages: 310
Title: National Geographic, Author: Various,
Issue: October
```

8. HashSet – Removing Duplicates

Goal: Learn about uniqueness in Sets.

Description:

1. Create a `List<Integer>` with duplicates: `[10, 20, 30, 20, 10, 40]`.
2. Convert it into a `HashSet` to remove duplicates.
3. Print both the list and the set.
4. Observe that the `HashSet` has no duplicates and may not preserve order.

Example Output:

```
Original List: [10, 20, 30, 20, 10, 40]
Unique Values (Set): [40, 10, 20, 30]
```

9. Queue – Task Management System

Goal: Learn FIFO behavior using a `LinkedList` as a `Queue`.

Description:

1. Create a `Queue<String>` of tasks.
2. Add tasks: `"Check Emails"`, `"Attend Meeting"`, `"Write Report"`.
3. Use `peek()` to see the next task.
4. Use `poll()` to process (remove) each task until empty.

Example Output:

```
Next Task: Check Emails
Processing: Check Emails
Processing: Attend Meeting
Processing: Write Report
All tasks completed.
```

10. Advanced Challenge – Word Frequency Counter

Goal: Combine loops and `HashMap` to count words.

Description:

1. Take a sample sentence:
   `"Java is fun and Java is powerful"`
2. Split it into words.
3. Use a `HashMap<String, Integer>` to count each word's frequency.
4. Print word frequencies.

Example Output:

```
Java → 2
is → 2
fun → 1
and → 1
powerful → 1
```

11. Advanced Challenge – Word Frequency Counter

Goal: Combine loops and `HashMap` to count words.

Description:

1. Take a sample sentence:
   `"Java is fun and Java is powerful"`
2. Split it into words.
3. Use a `HashMap<String, Integer>` to count each word's frequency.
4. Print word frequencies.

Example Output:

```
Java → 2
is → 2
fun → 1
and → 1
powerful → 1
```

12. Bonus Challenge – Student Ranking with TreeMap

Goal: Use a sorted map to rank students by score.

Description:

1. Create a `TreeMap<Integer, String>` where key = marks, value = student name.

2. Add several entries.
3. Print students in ascending order of marks.
4. Display the student with highest marks using `lastEntry()`.

Example Output:

```
60 → John
75 → Lisa
85 → Arjun
Topper: 85 → Arjun
```

13. **Problem statement:**

Create a Java class `BankAccount` that represents a simple bank account.
Use **encapsulation** to ensure that the account details (like balance and account number) cannot be accessed or modified directly.

Requirements:

1. Create a class `BankAccount` with the following **private** fields:
   - `accountNumber` (String)
   - `balance` (double)
2. Provide **public getter and setter methods**:
   - Getter for `accountNumber`
   - Getter for `balance`
   - Setter for `balance` (should only allow positive values)
3. Add a constructor to initialize the account.
4. Add two **public methods**:
   - `deposit(double amount)` → increases the balance
   - `withdraw(double amount)` → decreases the balance if there's enough money
5. In the `Main` class, create a `BankAccount` object and demonstrate:
   - Depositing money

    - Withdrawing money
    - Displaying account details

14. Employee Salary Management

Create an `Employee` class with private fields:

- `empId`
- `name`
- `basicSalary`
- `bonusPercentage`

Add:

- `getGrossSalary()` → returns `basicSalary + (basicSalary * bonusPercentage / 100)`
- Setters that validate inputs (e.g., no negative salary/bonus).

Goal: Encapsulation with derived properties and computed data.

15. Payment System

Create a superclass `Payment` with method:

```
void pay(double amount)
```

Subclasses:

- `CreditCardPayment`
- `PaypalPayment`
- `BankTransferPayment`

Each overrides `pay()` and prints a message like *“Paid 100.0 via PayPal.”*

Goal: Understand polymorphism through common interfaces.

Assignment/Task (Common Tasks)

Task Topic: OOPS

1. Inheritance (Single, Multilevel and Hierarchical)
2. Polymorphism
3. Abstraction
4. Encapsulation

Inheritance:

Questions Taken:

1. Create a base class Person with attributes name and age.
   Derive two classes: Doctor and Patient. Write a program to read and print details of a doctor and a patient.
2. Extend the Doctor class into Surgeon and Dentist.
   Add a specialization field. Show how a Surgeon object can access methods from Doctor and Person.
3. Create a Staff class derived from Person.
   Include a method calculateSalary() that depends on working hours. Demonstrate multilevel inheritance using Person → Staff → Nurse.
4. Class Patient extends a class Hospital which implements an interface Insurance to enforce the properties of the Insurance in Hospital which is later on used by the Patient. (Multi-Level Inheritance)

5. Calculating Area for Different Geometric Shapes
   Using inheritance, this program models different shapes (Circle, Rectangle, Triangle) by extending a common Shape class. Each subclass calculates its area differently, and an ArrayList is used to store a collection of different shapes.
6. Different Types of Bank Accounts
   Creating a base class BankAccount and extending it with different types of accounts (e.g., Savings, Checking). Each subclass implements specific interest rate calculations. The accounts are stored in an ArrayList and interest is calculated for all accounts.
7. Using inheritance, model various vehicle types (Car, Truck, Motorcycle) with a common base class Vehicle that calculates fuel efficiency. The program stores a list of different vehicles in an ArrayList and calculates the average fuel efficiency of all vehicles.
8. **Vehicle:** Created a main class vehicle with a method CalculateRentalCost(). Each sub class Bike, Truck and Car extends the class vehicle and implements the CalculateRentalCost() method its own way showing method overriding.
9. In this program a class Electric Car inherits from class Car which in turn inherits from class Vehicle. Vehicle class has a method called Start, Car has a method Drive and Electric Car has a method called Charge. The Electric Car can access methods of Car and Vehicle classes as well as its own method. Car can access its own method and the method of Vehicle class.
10. Create a Person class with a method introduce (). Inherit Student and Teacher classes from Person, overriding introduce () for each.
11. Create an Animal class with a method makeSound(). Extend it to Dog and Cat, overriding makeSound() to print different sounds.

12. Create a system where Employee has basic details and Developer inherits from it to add specialization.
13. Design a SmartPhone that can take photos and connect to the internet, create Interface for Camera and Internet.
14. Parking lot management
15. Booking system - different types of transportation
16. Single Level Inheritance: A Son class inherits from a father class to reuse basic identity details.
17. Hierarchical Inheritance: Classes add & sub inherit from a common Number class class.
18. Create a class Device with method start (). Subclass Phone and Laptop override start () with different messages.
19. Create a class Game with method play (). Subclass Football and Chess override play () with their own rules.
20. Multiple Interface Inheritance: Printable & Scannable → PrinterScanner: Create two interfaces, Printable and Scannable, each with its own method. Class PrinterScanner implements both interfaces and defines both methods.
21. Demonstrates single inheritance in java, where the school class inherits properties and methods from the student class. It prints the student's name, age, and ID using method calls from both parent and child classes.
22. Creating a library system includes books and magazines.
23. Parent class: College

    Child class: Engineering.

    The parent class contains the constructor to display college details, and the child class includes the constructor to show department details. I used constructor chaining to call the parent class constructor from child class.

24. Child class: Floweringplant

    The parent class constructor displays the plants’ name and type. The child class adds method to set and display flower specific details like name and season. I used Constructor chaining with super ().

25. Create a class Instrument with a name and a method play(). Make two subclass Guitar and Piano that displays the name and uses the play () method.

26. Book Late Fee Calculation System

    Use inheritance to show different book types overriding the late-fee calculation behaviour based on the number of delayed days.

27. Online Course Management System with Different User Roles:

    Design a system where users (Students and Instructors) can enroll, create, or manage courses. Instructors can upload materials, while students can enroll and view course content.

    Requirements:

    Create a base class User with fields like String userId, String name, and an abstract method displayRole().

    Create subclasses:

    Instructor: Implements displayRole() and adds methods to upload materials to courses.

    Student: Implements displayRole() and adds methods to enroll in and view courses.

    Create a Course class with fields like courseName, Instructor, List enrolledStudents. Add methods to enroll students and assign instructors.

    Implement functionality in the Main class to simulate a student enrolling in a course, an instructor uploading materials, and printing student and instructor roles.

Polymorphism:

Questions Taken:

1. Overload a method bookAppointment() in a Clinic class:
    - bookAppointment(String patientName)
    - bookAppointment(String patientName, String doctorName)
    - bookAppointment(String patientName, String doctorName, String date)
2. Create a class hierarchy with Doctor (base) and SpecialistDoctor (child). Override a method getConsultationFee() to return different values for general and specialist doctors.
3. Demonstrate **runtime polymorphism** by storing different types of doctor objects (e.g., Surgeon, Dermatologist) in a doctor reference array and calling their overridden methods.
4. Method Overloading and Method Overriding calculate () is overloaded with different parameters and overridden in a subclass to customize logic.
5. overloading and overriding created simple classes and did the overloading and overriding also did the constructor overriding.
6. Method Overloading: Multiple area () methods for square, rectangle, and circle- Compile time polymorphism.
7. Method Overriding: Rainforest and ConiferousForest override describe ()-runtime polymorphism.
8. Demonstrates method overloading and inheritance in Java, where the product class defines two multiply () with different parameters. The Multiply class extends product and uses display () to introduce each multiplication operation.

Abstraction:

Questions Taken:

1. Create an **abstract class** Appointment with an abstract method calculateFee(). Implement it in GeneralAppointment and EmergencyAppointment classes.
2. Define an **interface** PaymentGateway with methods makePayment() and generateReceipt(). Implement it in classes CardPayment and UPIPayment.
3. Use an **abstract class** MedicalRecord with concrete and abstract methods.
    - Abstract: updateRecord()
    - Concrete: printRecordDetails()
    - Implement it for PatientRecord and DoctorRecord.
4. Create an abstract class Insect with appropriate abstract methods, provide concrete methods by extending this class.
5. Create an abstract class vehicle with both abstract and concrete methods.
6. Create an interface named Appliance, with methods turnOn(), turnoff (), status () and implement it.
7. Java Interface Practice Program: Displaying Different Smartwatch Properties

    Utilizing interface methods like price (), screenSize(), battery () to display the properties of different smartwatches and calculating the overall value based on these features.
8. Java Interface Practice Program: Employee Attendance Tracking

Using an interface to enforce common methods like markAttendance() and calculateLeaveBalance() for different employee types (e.g., Full-Time, Part-Time).
The program tracks attendance and calculates leave balances, storing employee data in an ArrayList.

9. Create an interface Bonus eligible with method calculatebonus. Class employee implements the interface and has variable name and salary. Subclass manager and developer extends employee and implements BonusEligible. Printed corresponding name and salary.
10. Implementing an interface ShippingService with methods like calculateShippingCost() and getDeliveryTime(). Different shipping services (e.g., Standard, Express, International) implement these methods to calculate shipping costs based on distance and package weight, with the data stored in an ArrayList.
11. Create an interface Drawable with a method draw (). Implement it in Circle and Rectangle classes.
12. Create an interface Playable with a method play (). Implement it in Piano and Guitar classes.
13. Create an interface Shape with methods getArea() and getPerimeter(). Implement it in Square and Rectangle classes.
14. Library System
    Use a common interface-Library which will be implemented by Classes of different types of books. CommonOpertions such as setDue(), updateDue(),calculateDue() etc can be declared in Library.
15. Sorting
    An Abstract class Sorting can be used by subclasses that will implement different sorting algorithms.

16. Design a system for smart vehicles that combines navigation, entertainment, and diagnostics using multiple interfaces.
    Requirements:
    - Create three interfaces:
    - Navigable with methods startNavigation(), stopNavigation()
    - Playable with methods playMusic(), stopMusic()
    - Diagnosable with methods runDiagnostics(), showReport()

    Create a class SmartCar that implements all three interfaces and provides concrete implementations for each method.
17. Animal definition: created an abstract class Animal with predefined functions no of legs and wild animals then, created a child class extending animal class and defining the abstract method name of the animal and type of food it eats.
18. Vehicle definition: created an abstract class Vehicle with abstract methods like fuelType and maxSpeed, then created child classes like Car that define these methods and also added a method hasWheels in the parent class.
19. Design an abstract Vehicle class for StartEngine and a Flyable interface for optional capabilities. Create a non-flyable class (Car) and a class that implements both(Plane).
20. Abstract class Document with method print (). Subclass PDF and WordDoc implement print () differently.
21. The simple mover has interface common to enforce movement, but each object moves in its own different way
22. Funtional Interfaces with Lambda: Greeting interface with sayHello() used in lambda for Dynamic behavior.
23. Food Items with Interface & ArrayList: Pizza, Burger to be implement Food interface and stored in ArrayList.

24. This program demonstrates interface implementation and inheritance in java where add and sub classes implement the Operations interface.

    The Calculations class extends sub and showcases method execution for subtraction and a custom display message.
25. Smart Home system using interface and abstract class.

    Interface name: Fruits
    Methods: wash (), peel (), eat ()
    Class Name: Apple implements Fruits
    Implemented Methods: wash (), peel(),eat()
    Own Method: slice ().
    Class Name: Color extends Apple
    Inherits All Methods from Apple and Fruits
26. Interface name: Sweet

    Methods: prepare (), decorate (), serve ()
    Class Name: Ladoo implements Sweet
    Implemented Methods: prepare (), decorate (), serve()
    Own Methods: shape ()
    Class Name: Festival extends Ladoo
    Inherits all methods from Ladoo and Sweet
    Own method: gift ().
27. Interface Name: Clean, Fragrant

    Class Name: SoapBar implements both interfaces
    Class Name: Herbs extends SoapBar
    This program shows the multiple interface and inheritance to add features.
28. In a payment system, define a common payment interface so classes like credit card, PayPal, and UPI can process payments in their own ways.

29. In a notification service, use an interface so email, SMS, and push notifications all follow the same method for sending messages.
30. This program uses a café class contains two interfaces to calculate and display prices where Drinks interface deals with Tea and Coffee while Snacks interface handles Cakes and Cookies.
31. This program uses a dress code class with two interfaces to describe dress code details of a company. The men's dress code interface deals with descriptions of men's formal dress code and men's informal dress code while women's dress code interface handles descriptions of women's formal dress code and women's casual dress code.
32. Two interfaces, and, are created. A class implements both, meaning it must define how it flies and swims. This shows how a class can implement multiple interfaces.
33. Create a course management system that combines interface and abstraction. Start by defining an interface named Enrollable with two methods: enroll() and complete (). Then, implement an abstract class called Course that includes a String courseName field, a constructor to initialize it, and a concrete complete () method that prints a message indicating the course has been completed. The abstract class should also declare an abstract method showContent() to be implemented by subclasses. Develop two subclasses:

    ProgrammingCourse and DesignCourse, each implementing the enroll() and showContent() methods with course-specific behavior. In the main method, create instances of both course types, enroll in them, and display their content and completion messages.

34. Create an authentication system using abstraction where Admin and Guest classes override the abstract authenticate method to provide different login validation behaviours. The main method checks user input against both types to determine login success.

Encapsulation:

Questions Taken:

1. Create a Patient class with private fields name, age, and disease. Use getters and setters to access these values and demonstrate data protection.
2. Implement a ClinicAccount class that stores a balance field.
   Only allow deposits and withdrawals through public methods (prevent direct access).
3. Build a DoctorProfile class where updating email requires validation (must contain '@'). If invalid, show an error message using setter logic.
4. Create JavaBean to store user profile data. Create a login system and store user credentials using encapsulation.
5. Student Grade Management System
   This program uses encapsulation to store and manage student grades securely. The program allows adding grades, calculating average scores, and checking pass/fail status, with student data stored in an ArrayList.
6. Employee Performance Evaluation
   Using encapsulation, employee performance data (ratings, bonuses) are kept private. Methods allow updating performance ratings and calculating yearly bonuses. The data for multiple employees is stored in an ArrayList and processed accordingly.
7. Inventory Management with Price Updates

Encapsulating product details such as name, price, and quantity in a Product class. The program allows updating prices and quantities while ensuring controlled access to product data, with the product inventory stored in an ArrayList.

8. Create a Car class with private fields make, model, and year. Provide getter and setter methods for these fields.
9. Create a Book class with private fields title, author, and price. Use getter and setter methods to modify and access the values
10. Smartphone with private instance variables brand, model, and storageCapacity.
11. Class Mobile with private field batteryLevel. Add method to charge and check battery.
12. House with private instance variables address, numberOfRooms, and area.
13. Movie with private instance variables title, director, and duration
14. Class Movie with private field rating. Add method to set rating only between 1 and 5.
15. Digital Wallet: Keep balance and transaction history private. Provides methods like addFunds(), makePayment(), and getBalance() with validation checks.
16. Fitness Tracker App: Encapsulate health metrics like heart rate, steps, and calories. Allow controlled updates via methods like updateSteps() or logWorkout().
17. Vehicle Speed Monitor: Keep current speed and max speed private. Provide methods like accelerate (, brake(), and getSpeed() with safety checks.
18. Bank account: created a class BankAccount with private variables accountNumber, balance and accountHolder, then added methods

like deposit and withdraw with validation and used getters and setters to access the variables.

19. Bank Account System with Interest Calculation and Transaction Fees

    Requirements:

    Create a BankAccount class with private fields: accountNumber, balance, and transactionHistory (a list of strings). Use getter and setter methods for balance and
    accountNumber.

    Add a setter for balance that ensures the balance can never be negative.

    Add a method deposit(double amount) to add money, and withdraw (double amount) to subtract money (if balance is sufficient). Every withdrawal should incur a fixed fee (e.g., $2 fee).

    Create a method calculateInterest(double interestRate) that returns the interest for the current balance.

    Implement a method getTransactionHistory() to return the list of all transactions.

    In the Main class, simulate a series of deposits a withdrawal, and display the transaction history along with the final balance and interest.

20. Library system: created a class Book with private variables title, author and isAvailable, then added methods borrowBook and returnBook to manage availability and used encapsulation to prevent borrowing unavailable books.
21. Course enrollment: created a class Course with private variables courseName, enrolledStudents and maxCapacity, then added methods enrollStudent and removeStudent and used encapsulation to enforce capacity limits.
22. ArrayList with Encapsulation: Icecream class with private fields, getters and setters, and stored in ArrayList.

23. This program defines a Food class with properties foodname and foodamt using constructors, getters, setters and toString() for object representation. It stores multiple food items in an ArrayList and prints their details using a loop.
24. Class Name: Exam
    Private Fields: String subject, Int marks, int total
    Public Methods: Accessed private fields using getter and setter methods.
25. Class Name: Hotel
    Private Fields: String Hotelname, int tablecount, double Price
    Public Methods: Accessed private fields using getter and setter methods.
26. In a user account system, hide sensitive data such as passwords and emails within a class and validate any changes before saving them.
27. In this program a class Bottle encapsulates details of a water bottle including its capacity and current water level. These fields are set as private, hence cannot be accessed directly from outside the class.
    The class has methods to interact with bottle like fill method to add water, drink which reduces water level and get level which shows current water level.
28. In this program a class Temp encapsulates a temperature value in Celsius stored privately. It can be accessed or modified only by the methods like setCelsius to set temperature, toFahrenheit to convert temperature to fahrenheit, toKelvin to convert to Kelvin.
29. In this program, a class Music Player encapsulates the details of a music player such as the current song and volume level which are marked as private, so they cannot be accessed or changed directly

from outside the class.setSong method is used to set the song name ,setVolume can be used to set volume for the current song and display shows the current song and volume level.

30. Password Protector using encapsulation for data security and internal logic hiding.
31. In a public class Person, the variable name and age are kept private, which is then used in its constructor using “this” keyword. Getters and setter are also defined. In main, you can now access the information using the class object of Person in getter setter methods.

Topic: Collection

List:

(ArrayList) Questions Taken:

1. Create an ArrayList<String> to store patient names.
   Add 5 names, remove one, and display all remaining names using both for and for-each loop. (Store and Display Patient Names)
2. Define a Patient class with attributes name, age, and disease.Create an ArrayList<Patient> and:
   - Add at least 3 patients
   - Display their details using an iterator

   (Store Patient Objects)
3. Ask the user to enter a patient’s name. Check if that name exists in the ArrayList. If yes, display “Patient Found”, otherwise “Patient Not Found”. (Search for a Patient)
4. Store items in a shopping cart using ArrayList.

5. Use ArrayList<String> to store names of fruits. Add, remove, and display them.
6. Managing a Bookstore Inventory

   This program uses an ArrayList to store a collection of Book objects, each with attributes like title, author, and price. It allows operations like adding new books, updating prices, and calculating the total value of all books in stock.
7. Customer Order Management System

   Using an ArrayList, this program stores customer orders and their details (items, quantities, prices). It calculates the total cost of an order and displays the order summary for each customer in the list.
8. Movie Rental System

   This program uses an ArrayList to manage a collection of movies in a rental store. It allows adding new movies, renting out movies, and calculating the total rental earnings, with each movie having attributes like title, genre, and rental price.
9. Create a List of student names and a Set of student IDs. Add some names and IDs, then print them.
10. Create a List of integers and find the largest number in the list using Collections methods.
11. Unique words in paragraph
12. College database: contains a class called collegedb which stores the private variables and Studentdetails class creates the arraylist of collegedb and adds the details using addDetails method and veiws each student using veiw details.
13. Storing different objects in Array list: created different types of collections and added values to it.

14. Employee details: created a class emp with private variables and created an Array list of emp and added details of the employee and removed the employees using terminate function
15. This program uses a class vegetablesList to implement ArrayList that performs operations like adding a vegetable, removing using index, etc.

(LinkedList) Questions Taken:

1. Use a LinkedList<String> to manage a patient waiting queue.
   Add patients, remove the one who just completed their appointment (removeFirst()), and display the updated queue. (Patient Queue System**)**
2. Store appointment IDs (like A101, A102, etc.) in a LinkedList.
   Display the list in normal and reverse order using a ListIterator. (Reverse Appointment List)
3. Maintain two LinkedLists: One for General Doctors, One for Specialists. Merge them into a single list of all doctors and display.
4. Task Scheduler using LinkedList to add new Tasks at the end or remove from beginning
5. Use LinkedList<String> to simulate a playlist. Add songs and play them in order.
6. LinkedList Operations: Demonstrates stack(LIFO) and queue (FIFO) using addFirst(), push (), offer ().

(Stack) Questions Taken:

1. Use a Stack<String> to store recent appointment activities ("New Booking", "Payment Done", "Record Updated").
   Pop the last action and show the remaining stack. (Appointment History)
2. Each time the user enters an action (e.g., "Added Patient", "Deleted Patient"), push it onto a stack. If the user types "undo", pop the last action. (Undo Feature Simulation)
3. Push 5 doctor names onto a Stack<String>.
   Pop and display them to show the reverse order of entry. (Reverse Display)
4. Infix to postfix stack
5. This program uses a Stack to manage a list of icecream flavours and items demonstrating stack operations like add (), push() and pop(). it prints the stack contents before and after removing the top element to show how pop() affects the collection.
6. This program creates a Stack called Animals with operations to add new animals to the stack, remove some animals from stack using index and remove animal from the top using the method pop.
7. This program uses a class Bucket where colored balls are added on top of another and is popped where the last one gets popped.

(Vector) Questions Taken:

1. Use a Vector<String> to store doctor names.
   Add 5 names and display them using Enumeration. (Store Doctor Names)
2. Create a Vector<Patient> (with name, age).
   Add, remove, and insert patient records dynamically, then display all. (Manage Patient List Dynamically)
3. Maintain a Vector to track the last 5 admin actions. If size exceeds 5, remove the oldest one. Display current actions. (Recent Activity Log)

4. This program creates a vector called Cities where user can add new cities, update the old city names with new names and delete the unwanted city names using index.

Set:

(HashSet, LinkedHashSet and Tree) Questions Taken:

1. Use a HashSet<Integer> to store doctor IDs.
   Try adding duplicates and observe that duplicates are ignored. (HashSet – Unique Doctor IDs)
2. Use a LinkedHashSet<String> to store patient names in the order they registered. Add, remove, and display them to confirm the insertion order is preserved. (LinkedHashSet – Patient Registration Order)
3. Use a TreeSet<String> to store doctor names.
   Add names in random order and display them sorted alphabetically. (TreeSet – Sorted Doctor Names)
4. Store unique email addresses using HashSet.
5. Create a system to store and display a sorted list of unique contact names using TreeSet.
6. Create a Set to store unique country names. Add some country names and try to add duplicates. Print the set.
7. Inventory system using list and set
8. Student Details: TreeSet Store student names or IDs in a TreeSet to maintain sorted order automatically.
9. Department Subjects: Maintain insertion order of subjects offered by a department using LinkedHashSet.
10. This program demonstrates the use of a TreeSet to store and automatically sort a collection of names in ascending order.

It prints the sorted set, iterates through each element, and displays the total number using size().

11. It uses a LinkedHashSet to store unique programming concepts while preserving insertion order. It demonstrates how duplicate values like "Java" are ignored and prints all elements sequentially.
12. Using Hashset to remove duplicates.
13. To manage student attendance using basic set operations in Hashset
14. Created a class book to store title and price. Added three books using hashmap. Displayed all book entries and also printed price for a specific book.
15. A set is used to store numbers like 10 and 20. If 10 is added again, it is ignored because sets do not allow duplicates. This is useful for storing unique values.
16. A map is created where each subject (e.g., Math, Science) is a key, and the corresponding marks are values. You can retrieve the marks for any subject using its name. This shows key-value storage.
17. Merge two lists into one and remove duplicates using a Set to get unique sorted values.
18. This program uses a class to build a HashSet just like array but it is unordered, unsorted and it doesn't allow index based programs or duplicate elements as it throws an error.
19. Order Management System Using HashMap
    Design an order management system where each customer can place multiple orders. Use a HashMap to store orders by customer ID, where each value is a list of orders.

Requirements:

Create a class Order with fields: orderId, productName, quantity, and price.

Create a customer class with fields: customerId, name, and a HashMap orders to store multiple orders.

Implement methods in Customer to:

Add an order.

View all orders.

Calculate the total value of all orders for a customer.

In the Main class, simulate adding orders for multiple customers, and display their order history along with the total value.

20. Hotel Booking System Using TreeSet

Design a hotel booking system where guests can book rooms. Use a TreeSet to manage
available rooms in sorted order. The system should allow checking room availability, book a room, and canceling a booking.

Requirements:

Create a Room class with fields: roomNumber, roomType, pricePerNight, and implement Comparable to allow sorting based on roomNumber.

Create a hotel class with a TreeSet to store available rooms.

Implement methods:

bookRoom(Room room): Books a room if it is available.

cancelBooking(Room room): Cancels a booking and makes the room available again.

viewAvailableRooms(): Displays all available rooms sorted by roomNumber.

In the Main class, simulate booking and canceling rooms, and display the status of available rooms.

21. Inventory System Using LinkedHashMap

Design an inventory system to manage product stock using LinkedHashMap to maintain insertion order. Implement functionality to add, remove, and update products.

Requirements:

Create a Product class with fields: productId, productName, quantity, and price.

Create an Inventory class with a LinkedHashMap to store products by their productName.

Implement methods to:

addProduct(Product product): Adds a product to the inventory.

removeProduct(String productName): Removes a product from the inventory.

updateProductQuantity(String productName, int quantity): Updates the quantity of an existing product.

getProductDetails(String productName): Returns details of a product by name.

In the Main class, simulate adding, updating, and removing products, and display the inventory.