1) A university is developing a student management system where each student has attributes like name, roll
number, and courses enrolled. The system needs to allow students to perform actions such as enrolling in courses
and checking grades.
Question: Explain the relationship between objects, classes, data members, and methods in OOP with reference to
this system.

ans=> In Object-Oriented Programming (OOP), classes, objects, data members,
and methods are fundamental concepts that work together to model real-world entities
In the context of a student management system, where each student has attributes
like name, roll number, and enrolled courses, and can perform actions such as enrolling in courses and checkinggrades,

An object is an instance of a class. It represents a real-world entity created from the class.

Data members are variables declared inside a class that store information related to the object.
In the Student class, name, rollNumber, and courses are data members that hold each student's personal details.

Methods are functions defined inside a class that describe the behaviors or actions the object can perform.
For example, the methods enrollInCourse() and checkGrades() define actions a student can take in the system.

```
code:- class Student {
   string name;
   int rollNumber;
   vector<string> courses;

   void enrollInCourse(string course);
   void checkGrades();
};
int main(){
Student s1;
}
```

2) A company is designing an employee management system. Some employee details, such as salary and bank details,
should be hidden from other employees but accessible to HR. Other general details, like name and department,
should be visible to everyone.
Question: How does the private modifier differ from the protected modifier in OOP, and how can they be used to
implement data security in this scenario?

ans=> In Object-Oriented Programming (OOP), access modifiers such as private and protected are used to control
the visibility and accessibility of class members (data and methods), ensuring proper data security and encapsulation.

Private Modifier:
The private modifier makes data accessible only within the same class.
It is not accessible from outside the class, including derived (child) classes.

It is used to hide sensitive data such as salary and bank details from other employees.
salary and bankAccount are private and can’t be accessed directly from outside the Employee class.

Protected Modifier:
The protected modifier allows access within the class itself and by derived (sub) classes.
It is useful when certain information needs to be hidden from the public but shared with subclasses, such as the HR class.
Here, HR can access salary because it is declared as protected in the Employee class.

code-
```
class Employee {
private:
    float salary;
    string bankAccount;

public:
    void setSalary(float s) { salary = s; }
    float getSalary() { return salary; }
};
class Employee {
protected:
    float salary;
};

class HR : public Employee {
public:
    void viewSalary() {
        cout << "Salary: " << salary;
    }
};
```

3) An e-commerce application allows users to apply different discount coupons. Some coupons apply a flat discount,
while others apply a percentage-based discount. The system has a method applyDiscount() that behaves differently
based on the type of coupon.
Question: Explain the difference between compile-time polymorphism and run-time polymorphism with reference
to this system.

ans=> Compile-Time Polymorphism (Static Binding):
Occurs at compile time.
Achieved using method overloading or operator overloading.
The compiler decides which method to execute based on the number or type of parameters.
Here, applyDiscount() is overloaded — same name, different parameter list.
This is compile-time polymorphism.

Run-Time Polymorphism (Dynamic Binding):
Occurs at runtime.
Achieved using method overriding through inheritance and virtual functions.
The method that gets executed depends on the object’s actual type, not the reference type.
the applyDiscount() method behaves differently based on the actual object type (FlatDiscount or PercentageDiscount).
This is run-time polymorphism.

code-
```
class Discount {
public:
    void applyDiscount(float amount) {
        cout << "Flat discount of  " << amount << " applied.";
    }

    void applyDiscount(float amount, float percentage) {
        float discount = amount * (percentage / 100);
        cout << "Percentage discount of  " << discount << " applied.";
    }
};
class Discount {
public:
    virtual void applyDiscount() {
        cout << "Default discount applied.";
    }
};

class FlatDiscount : public Discount {
public:
    void applyDiscount() override {
        cout << "Flat discount of  100 applied.";
    }
};

class PercentageDiscount : public Discount {
public:
    void applyDiscount() override {
        cout << "10% discount applied.";
    }
};
Discount* d;       // Base class pointer
d = new FlatDiscount();
d->applyDiscount();  // Calls FlatDiscount version at runtime

d = new PercentageDiscount();
d->applyDiscount();  // Calls PercentageDiscount version at runtime
```

4) A vehicle rental service provides different rental price calculations based on vehicle type. Cars have a base price
per day, trucks have an additional weight-based charge, and motorcycles have a mileage-based price.
Question: If a Vehicle class has an abstract method calculate_rental_price(), how would different vehicle types (Car,
Truck, Motorcycle) implement it?

ans->
In Object-Oriented Programming (OOP), an abstract class is a class that contains at least one abstract method
— a method that is declared but not implemented in the base class.

The method must be overridden by all subclasses to provide their own specific implementation.
This approach ensures that every subclass must define its own version of the behavior.
Cars: Rental is based on a fixed base price per day.
Trucks: Rental includes the base price + an additional charge based on weight.

Motorcycles: Rental is based on mileage (kilometers driven).
CODE-

```
// Abstract base class
class Vehicle {
public:
    virtual float calculate_rental_price() = 0;  // Pure virtual function
};
class Car : public Vehicle {
public:
    float calculate_rental_price() override {
        return 1000;  // Fixed base price per day
    }
};

class Truck : public Vehicle {
private:
    float weight;
public:
    Truck(float w) { weight = w; }

    float calculate_rental_price() override {
        return 1500 + (weight * 10);  // Base + weight-based charge
    }
};

class Motorcycle : public Vehicle {
private:
    float kilometers;
public:
    Motorcycle(float km) { kilometers = km; }

    float calculate_rental_price() override {
        return kilometers * 5;  // Mileage-based price
    }
};
```

5) A banking application needs to handle multiple types of exceptions, such as
InsufficientFundsException,
InvalidAccountException, and NetworkFailureException. These exceptions can occur simultaneously when processing
transactions.
Question: How can you handle multiple exceptions in Java to ensure smooth transaction processing?

ans:-
In Java, multiple exceptions can be handled using try-catch blocks, where each specific exception is caught
using a separate catch block. This ensures that each type of error can be handled appropriately without stopping
the execution of the entire program.
In a banking application, exceptions like:
InsufficientFundsException
InvalidAccountException
NetworkFailureException
might occur during a single transaction. Proper handling ensures the system remains stable and user-friendly.

code-

```
try {
    processTransaction();  // Might throw multiple exceptions
}
catch (InsufficientFundsException e) {
    System.out.println("Error: Not enough balance.");
}
catch (InvalidAccountException e) {
    System.out.println("Error: Account number is invalid.");
}
catch (NetworkFailureException e) {
    System.out.println("Error: Network issue. Please try again.");
}
finally {
    System.out.println("Transaction attempt complete. Logging out...");
}
```

6) Identify and Fix the Error in the Given Code

```
// Interface Animal
interface Animal {
void makeSound();
}
// Interface Cat
interface Cat {
void purr();
}
// Dog class implementing both interfaces
class Dog extends Animal, Cat {
public void makeSound() {
System.out.println("Dog barks!");
}
public void purr() {

System.out.println("Dog cannot purr, but method must be implemented!");
}
}
// Main class
public class Main {
public static void main(String[] args) {
Dog dog = new Dog();
dog.makeSound();
dog.purr();
}
}
```

ans-

```
// Interface Animal
interface Animal {
    void makeSound();
}

// Interface Cat
interface Cat {
    void purr();
}

// Dog class implementing both interfaces
class Dog implements Animal, Cat {
    public void makeSound() {
        System.out.println("Dog barks!");
    }

    public void purr() {
        System.out.println("Dog cannot purr, but method must be implemented!");
    }
}

// Main class
public class Main {
    public static void main(String[] args) {
        Dog dog = new Dog();
        dog.makeSound();
        dog.purr();
    }
}
```

7) A flight booking system allows customers to enter their age while booking a ticket. If the user enters a negative
number, the system should throw an exception. However, missing a required field should cause a different exception.
Question: Differentiate between checked and unchecked exceptions in Java and explain how they can be used to
handle such cases in the flight booking system.

ans:-
In Java, exceptions are categorized into checked and unchecked exceptions based on when they are detected and how
they must be handled.
 1. Checked Exceptions:
These are checked at compile time.
The compiler forces you to handle them using try-catch or throws.
Typically used for recoverable errors (e.g., file not found, missing input).
Example in Flight Booking:
If a required field like name or age is missing during booking, we can throw a checked exception like MissingFieldException.

```
class MissingFieldException extends Exception {
    public MissingFieldException(String message) {
        super(message);
    }
}
```

2. Unchecked Exceptions:
These are not checked at compile time.
Subclass of RuntimeException.
Usually used for programming errors like logic bugs or invalid data input.
Example in Flight Booking:

If the user enters a negative age, we can throw an unchecked exception like InvalidAgeException.

```
class InvalidAgeException extends RuntimeException {
    public InvalidAgeException(String message) {
        super(message);
    }
}
```

8) What will happen if an exception occurs and there is no catch block?

ans-
If an exception occurs in Java and there is no catch block to handle it, the following happens:
The JVM (Java Virtual Machine) will look for a matching catch block.
If no catch block is found, the JVM will:
Terminate the program abnormally.
Print an exception stack trace on the console.
Show the type of exception, the error message, and the line number where the exception occurred.

code-
```
public class Main {
    public static void main(String[] args) {
        int a = 5 / 0;  // This will throw ArithmeticException
        System.out.println("This line will not be executed.");
    }
}
```

9) A software development company is working on an employee attendance system. Each employee has a unique ID
and login time. The system initializes employee details when they log in and provides methods to retrieve and update
their attendance data.
Question: What is the difference between a method and a constructor in OOP? Explain with reference to the
employee attendance system.

ans-
In Object-Oriented Programming (OOP), both constructors and methods are used inside classes, but they serve different purposes
Constructor:
A constructor is a special method used to initialize an object.
It has the same name as the class and no return type, not even void.
It is called automatically when an object is created.
Example in Employee Attendance System:
When an employee logs in, we use a constructor to initialize their details like employee ID and login time:
code-
```
class Employee {
    int empId;
    String loginTime;

    // Constructor
    Employee(int id, String time) {
        empId = id;
        loginTime = time;
    }
}
```

Method:
A method is a function defined in a class to perform specific actions or operations.
It has a name, can return a value, and is called manually on an object.
Example in Employee Attendance System:
Methods can be used to retrieve or update attendance data after login:
code-
```
class Employee {
    // ...

    void updateLoginTime(String newTime) {
        loginTime = newTime;
    }

    String getLoginTime() {
        return loginTime;
    }
}
```

10) A bank application stores customer account details securely. The account balance should not be directly
accessible but can be retrieved or updated using specific methods.
Question: Explain how getter and setter methods contribute to encapsulation. Provide a code example demonstrating how they can be used in this banking scenario.

ans-
Encapsulation is one of the fundamental concepts of Object-Oriented Programming (OOP). It is the mechanism of wrapping
data (variables) and code (methods) together as a single unit and restricting direct access to some of the object's
components

Role of Getter and Setter Methods:
Private variables can't be accessed directly from outside the class.
To safely access or modify them, we use:
get methods (getters) to retrieve the value.
set methods (setters) to modify the value.
This ensures data security and integrity, which is critical in a banking application.
code-
```
class BankAccount {
    private int accountNumber;
    private double balance;

    // Constructor
    public BankAccount(int accNo, double initialBalance) {
        accountNumber = accNo;
        balance = initialBalance;
    }

    // Getter for balance
    public double getBalance() {
        return balance;
    }

    // Setter for balance (with validation)
    public void setBalance(double newBalance) {
        if (newBalance >= 0) {
            balance = newBalance;
        } else {
            System.out.println("Invalid balance! Cannot be negative.");
        }
    }

    // Optional: Getter for account number (read-only)
    public int getAccountNumber() {
        return accountNumber;
    }
}
public class Main {
    public static void main(String[] args) {
        BankAccount customer1 = new BankAccount(12345, 5000.00);

        // Accessing balance using getter
        System.out.println("Current Balance:  " + customer1.getBalance());

        // Updating balance using setter
        customer1.setBalance(6000.00);
        System.out.println("Updated Balance:  " + customer1.getBalance());

        // Trying to set a negative balance (invalid)
        customer1.setBalance(-1000.00); // Error message shown
    }
}
```

11) A university wants to model its staff hierarchy. A Professor is a type of Employee, and a Department has multiple
Professors.
Question: Describe the "is-a" and "has-a" relationships in OOP with reference to this scenario. Provide an example
for each.
ans-
In Object-Oriented Programming, "is-a" and "has-a" represent two different types of relationships between classes
"is-a" Relationship (Inheritance):
Used when one class is a specialized version of another.

Achieved through inheritance (extends keyword in Java).

It represents a parent-child or general-special relationship.
A Professor is an Employee, so we can create this relationship using inheritance:

"has-a" Relationship (Composition/Aggregation):
Used when one class contains another class.

Shows a "part-of" or "ownership" relationship.

Achieved by including objects as member variables.

A Department has multiple Professors, so we create a "has-a" relationship:
code-

```
class Employee {
    String name;
    int empId;
}

class Professor extends Employee {
    String subject;

    void teach() {
        System.out.println(name + " is teaching " + subject);
    }
}

class Department {
    String deptName;
    Professor[] professors;  // Array of Professor objects

    void listProfessors() {
        for (Professor p : professors) {
            System.out.println(p.name + " teaches " + p.subject);
        }
    }
}
```

12) A game development team is using both C++ and Java to design a game engine. They notice that Java requires
interface and implements keywords, while C++ does not.
Question: Why does C++ not require the interface or implements keyword like Java? Explain with a code example.

ans-
In Java, interfaces are explicitly defined using the interface keyword, and classes use the implements keyword to
implement them.
But in C++, interfaces are implemented indirectly using abstract classes with pure virtual functions —
so there's no need for special keywords like interface or implements.
Why Java needs interface and implements:
Java is strictly object-oriented, so everything is inside a class or interface.
Java uses the interface keyword to define a contract that classes must follow.
A class must use implements to promise that it will define all methods from that interface.
Why C++ doesn't need them:
C++ supports multiple inheritance and allows classes to inherit from abstract classes.
Interfaces in C++ are made using abstract classes with pure virtual functions.
No need for new keywords — just use virtual and = 0.

java code-
```
interface GameObject {
    void update();
}

class Player implements GameObject {
    public void update() {
        System.out.println("Player position updated.");
    }
}
```

cpp code-
```
#include <iostream>
using namespace std;

class GameObject {
public:
    virtual void update() = 0; // Pure virtual function
};

class Player : public GameObject {
public:
    void update() override {
        cout << "Player position updated." << endl;
    }
};
```

13) A social media platform is implementing user profiles. Some details, such as username and profile picture, are
visible to everyone, while other details, like contact number and email, are hidden and can only be accessed via a
request system.
Question: What is the difference between Encapsulation and Abstraction in OOP? How would you use these concepts
to design the privacy settings in this social media application?

ans-Encapsulation Encapsulation is the process of binding data and the methods that operate on that data within a single unit (class)
To protect data and restrict direct access to it.
Achieved using access modifiers like private, public, protected.
Focuses on how data is accessed or modified
A class having private variables and public getter/setter methods.

Abstraction Abstraction is the process of hiding the internal implementation details and showing only essential features
To reduce complexity and allow the user to interact with only necessary information
Achieved using abstract classes or interfaces
Focuses on what information is relevant to the user.
A user interface button that performs a function without showing the underlying code.

In a social media platform, user profiles have both public and private details. For example:
Public: Username, Profile Picture
Private: Contact Number, Email
Using Encapsulation:
Encapsulation is used to protect sensitive data like contact number and email by making them private variables inside a
class. Access is only given through controlled methods
code-
```
class UserProfile {
private:
    string email;
    string contactNumber;

public:
    string getEmail(bool isRequestApproved) {
        if (isRequestApproved) return email;
        else return "Access Denied";
    }

    string getContactNumber(bool isRequestApproved) {
        if (isRequestApproved) return contactNumber;
        else return "Access Denied";
    }
};
```

Abstraction is used to hide the complex logic of the permission request system from the user. The user simply calls
getEmail() or getContactNumber() and receives either the data or an access denied message, without knowing how access
was granted or denied internally.

14) A student information system needs to store student records in a file. It should allow reading student details
when needed and updating them efficiently.
Question: Explain how to read and write files in C++ using fstream. Provide an example of how the student
information system could implement this feature.

ans- Class	Purpose
ifstream	For reading from files
ofstream	For writing to files
fstream	For both reading and writing

To perform file operations:
Include the header: #include <fstream>
Use objects of the above classes
Always check if the file is open using .is_open()

code-
```
#include <iostream>
#include <fstream>
#include <string>
using namespace std;

class Student {
public:
    string name;
    int roll;
    float marks;

    void input() {
        cout << "Enter Name, Roll, Marks: ";
        cin >> name >> roll >> marks;
    }

    void display() {
        cout << "Name: " << name << ", Roll: " << roll << ", Marks: " << marks << endl;
    }
};
```

```
int main() {
    Student s;

    // Writing to file
    ofstream outFile("students.txt", ios::app); // app = append mode
    s.input();
    if (outFile.is_open()) {
        outFile << s.name << " " << s.roll << " " << s.marks << "\n";
        outFile.close();
    } else {
        cout << "Error opening file for writing.\n";
    }

    // Reading from file
    ifstream inFile("students.txt");
    if (inFile.is_open()) {
        while (inFile >> s.name >> s.roll >> s.marks) {
            s.display();
        }
        inFile.close();
    } else {
        cout << "Error opening file for reading.\n";
    }

    return 0;
}
```

Explanation:
ofstream is used to write student details into "students.txt" in append mode.
ifstream is used to read student records and display them.
You can also use fstream with ios::in | ios::out to implement updates by reading, modifying in memory, and writing back.

15) Identify and Fix the Error in the Given Code

```
public class ExceptionExample {
public static void main(String[] args) {
try {
int num = 5 / 0;
}
catch () {
System.out.println("Cannot divide by zero");
}
}
}
```

ans-The catch block is missing the exception type

```
public class ExceptionExample {
    public static void main(String[] args) {
        try {
            int num = 5 / 0;  // This will throw ArithmeticException
        }
        catch (ArithmeticException e) {
            System.out.println("Cannot divide by zero");
        }
    }
```

}

16) A payment gateway processes multiple types of transactions. While executing a transaction, different exceptions
may occur, such as InsufficientBalanceException, InvalidCardException, or TransactionTimeoutException. The system
must handle all these exceptions to ensure smooth transactions.
Question: How can multiple exceptions be handled in Java to make sure the payment gateway runs without failure?
Provide a suitable example.

ans-
In Java, multiple exceptions can be handled in two ways:

1. Using multiple catch blocks:
Each block handles a specific exception
The first matching block is executed.
This makes the program more reliable and avoids crashes during runtime.
2. Using multi-catch (|) in Java 7+:
One catch block can handle multiple exceptions using | (pipe symbol).

code-
```
// Custom Exceptions
class InsufficientBalanceException extends Exception {}
class InvalidCardException extends Exception {}
class TransactionTimeoutException extends Exception {}

public class PaymentGateway {

    // Simulate a transaction method
    public static void processTransaction(int code) throws InsufficientBalanceException,
InvalidCardException, TransactionTimeoutException {
        switch (code) {
            case 1:
                throw new InsufficientBalanceException();
            case 2:
                throw new InvalidCardException();
            case 3:
                throw new TransactionTimeoutException();
            default:
                System.out.println("Transaction Successful");
        }
    }

    public static void main(String[] args) {
        try {
            processTransaction(2);  // Change code to 1, 2, 3 to simulate different exceptions
        }
        catch (InsufficientBalanceException e) {
            System.out.println("Transaction failed: Insufficient balance.");
        }
        catch (InvalidCardException e) {
            System.out.println("Transaction failed: Invalid card.");
        }
        catch (TransactionTimeoutException e) {
            System.out.println("Transaction failed: Timeout occurred.");
        }
        catch (Exception e) {
            System.out.println("Transaction failed: Unknown error.");
        }
    }
}
```

17) A hospital management system stores patient records. Each patient has a unique ID, name, and medical history.
The system should allow doctors to create and access patient records efficiently.
Question: What is the difference between a Class and an Object in OOP? Explain using this hospital management
system as an example.

ans-

Class	Object
A class is a blueprint or template for creating objects.	An object is a real-world instance of a class.
It defines properties (data) and methods (functions).	It holds actual data and can use methods defined in the class.
No memory is allocated until an object is created.	Memory is allocated when an object is created.
Example: Patient class	Example: Patient p1 = new Patient();

Example: Hospital Management System
Let’s say we want to manage patient records. We can create a Patient class with properties like ID, name,
and medicalHistory.

```
class Patient {
    int id;
    String name;
    String medicalHistory;

    void displayInfo() {
        System.out.println("ID: " + id);
        System.out.println("Name: " + name);
        System.out.println("History: " + medicalHistory);
    }
}
public class HospitalSystem {
    public static void main(String[] args) {
        Patient p1 = new Patient();  // Object
        p1.id = 101;
        p1.name = "Aditya Shinde";
        p1.medicalHistory = "No allergies, previous surgery in 2022.";

        p1.displayInfo();
    }
}
```

18) A company is developing a role-based access control system where different users have different levels of access.
Admin can modify and delete data, Manager can view and edit, while Employees can only view certain details.

Question: Create a comparison table of access modifiers (private, protected, public, and default) and explain how
they can be used in this system to restrict access appropriately.

ans

Modifier	Access Level	Accessible From
private	Most restricted	Only within the same class
default	Package-level access	Within the same package
protected	Inherited + package-level	Same package + subclasses (even in different packages)
public	No restriction	Accessible from anywhere

Admin gets full access (via private methods).
Managers get limited edit access (via protected/default).
Employees get read-only access (via public methods).
Unauthorized access is automatically restricted at the language level.

19) A robotics startup is using C++ and Java to develop a robotic arm control system. While coding the interfaces for
different sensors, they observe that Java requires interface and implements keywords, whereas C++ does not.
Question: Why does C++ not require the interface or implements keyword like Java? Explain with a code example.

ans-
In Java, interfaces are defined using the interface keyword, and classes use implements to promise they'll define
all the methods.Java separates interface from classes strictly

In C++, there's no interface keyword. Instead, an abstract class with only pure virtual functions is used to achieve
the same functionality.C++ merges both using abstract classes

java code-
```
interface Sensor {
    void readData();
}

class TemperatureSensor implements Sensor {
    public void readData() {
        System.out.println("Reading temperature data...");
    }
}
```

cpp code-
```
#include <iostream>
using namespace std;

class Sensor {
public:
    virtual void readData() = 0;  // Pure virtual function
};
```

```
class TemperatureSensor : public Sensor {
public:
    void readData() override {
        cout << "Reading temperature data..." << endl;
    }
};
```

20) A school management system has a Student class that inherits from the Person class. The Person class has a
method getDetails(), which returns basic information, but the Student class needs to provide additional student-
specific details.

Question: How do the super and this keywords work in Java? Explain their use in this school management system.

ans-
this Refers to the current class instance (same object)Used in the constructor to refer to current object's studentId,
name, etc.
super Refers to the parent class (superclass) and is used to access it
Call the parent class constructor (super(name, age))
Call the parent's method (super.getDetails()) inside the child class |

You have a Person class with a getDetails() method, and a Student class that inherits from it and wants to add more info.

code-
```
class Person {
    String name;
    int age;

    Person(String name, int age) {
        this.name = name;      // 'this' refers to the current instance
        this.age = age;
    }

    void getDetails() {
        System.out.println("Name: " + name);
        System.out.println("Age: " + age);
    }
}
class Student extends Person {
    String studentId;

    Student(String name, int age, String studentId) {
        super(name, age); // Call the constructor of the Person class
        this.studentId = studentId; // 'this' refers to current Student object
    }

    @Override
    void getDetails() {
        super.getDetails(); // Call the parent's getDetails() first
        System.out.println("Student ID: " + studentId);
    }
}
```

21) A vehicle rental management system allows customers to rent vehicles. Customers can specify rental duration,
vehicle type, or even add extra services like insurance.
Question: How would you implement method overloading for a rent_vehicle() method to support different customer
needs? Provide an example.

ans-Method Overloading means defining multiple methods with the same name but with different parameters
(type, number, or order).
It helps provide different ways to call the same function based on user requirements!

Use Case in Vehicle Rental System:
Customers might:
Rent a vehicle for a fixed duration.
Choose a specific vehicle type.
Add extra services like insurance or GPS.
You can create overloaded versions of rent_vehicle() for each case!

code-
```
class VehicleRental {

    // 1. Basic rental: only duration
    void rent_vehicle(int days) {
        System.out.println("Vehicle rented for " + days + " day(s).");
    }

    // 2. Rental with vehicle type
    void rent_vehicle(int days, String vehicleType) {
        System.out.println("Rented a " + vehicleType + " for " + days + " day(s).");
    }

    // 3. Rental with vehicle type and insurance
    void rent_vehicle(int days, String vehicleType, boolean insurance) {
        System.out.print("Rented a " + vehicleType + " for " + days + " day(s)");
        if (insurance) {
            System.out.println(" with insurance.");
        } else {
            System.out.println(" without insurance.");
        }
    }
}
```

22) A banking application processes transactions. If an error occurs, such as insufficient funds or invalid account
number, the system should handle it gracefully instead of crashing.
Question: Define Exception Handling and Error Handling and explain how they help in building reliable banking
applications.

ans

Concept	Explanation
Exception Handling	A way to catch and manage unexpected conditions (like runtime errors) during program execution.
Error Handling	A broader term that includes detecting, reporting, and recovering from both errors and exceptions (like logic errors, syntax issues, or critical failures)

Banking applications deal with:
User money transactions
Sensitive data
High expectations of reliability and trust
So, you can’t afford your app to crash or misbehave when an error occurs.

Feature	Role in Banking App
Prevent Crashes	Avoids app shutdown during exceptions like InsufficientFundsException.
Better User Experience	Shows clear error messages instead of system errors.
Security	Prevents leaks of internal code or stack traces.
Debugging	Makes it easier to log errors and fix bugs.
Transaction Safety	Ensures money is not deducted or processed in case of a failed operation.

code-
```
class InsufficientFundsException extends Exception {
    public InsufficientFundsException(String message) {
        super(message);
    }
}

class BankAccount {
    int balance = 1000;

    void withdraw(int amount) throws InsufficientFundsException {
        if (amount > balance) {
            throw new InsufficientFundsException("Insufficient balance in account!");
        } else {
            balance -= amount;
            System.out.println("Withdrawal successful. Remaining balance: " + balance);
        }
    }
}
public class Main {
    public static void main(String[] args) {
        BankAccount account = new BankAccount();

        try {
            account.withdraw(1500);
        } catch (InsufficientFundsException e) {
            System.out.println("Error: " + e.getMessage());
        }
    }
}
```

23) Identify and Fix the Error in the Given Code

```
class Student {
private int age = 20;
}
public class Main {
public static void main(String[] args) {
Student s = new Student();
System.out.println(s.age);
}
```

ans-
age is marked as private, so it cannot be accessed directly from another class (Main in this case).

code-

```
class Student {
    private int age = 20;

    // Getter method to access private variable
    public int getAge() {
        return age;
    }
}

public class Main {
    public static void main(String[] args) {
        Student s = new Student();
        System.out.println(s.getAge());
    }
}
```

24) A flight booking system allows customers to book tickets. If an invalid seat number is entered, the system should
immediately generate an exception. However, some exceptions, like invalid passport numbers, should be declared in
the method signature for the calling function to handle.
Question: What is the difference between throw and throws in Java? Explain with reference to this flight booking
system.

ans-

Keyword	throw	throws
Meaning	Used to actually throw an exception in the code	Used to declare that a method might throw exception(s)
Placement	Inside a method or block	In the method signature
Type	Followed by a specific Exception object	Followed by Exception class name(s)
Example	throw new InvalidSeatException();	public void bookFlight() throws InvalidPassportException

code-

```
class InvalidSeatException extends Exception {
    public InvalidSeatException(String message) {
        super(message);
    }
}
```

```
public void selectSeat(int seatNumber) throws InvalidSeatException {
    if (seatNumber < 1 || seatNumber > 150) {
        throw new InvalidSeatException("Seat number is invalid!");
    }
    System.out.println("Seat " + seatNumber + " booked successfully!");
}
```

throw is used to create and throw the exception when seat number is wrong.

throws is used to declare that this method may throw InvalidSeatException.