1) Add 8 to the number 2345, divide by 3, take modulus with 5, multiply the result by 5. Display final result.

```
public class Question1 {
    public static void main(String[] args) {
        int num = 2345;
        num = num + 8;
        num = num / 3;
        num = num % 5;
        num = num * 5;
        System.out.println("Final result: " + num);
    }
}
```

---

2) **Solve the above using assignment operators.**

```
public class Question2 {
    public static void main(String[] args) {
        int num = 2345;
        num += 8;
        num /= 3;
        num %= 5;
        num *= 5;
        System.out.println("Final result: " + num);
    }
}
```

---

3) **Calculate total number of girls getting grade 'A'.**

```
public class Question3 {
    public static void main(String[] args) {
        int totalStudents = 90;
        int boys = 45;
        int gradeA = (totalStudents * 50) / 100;
        int boysWithGradeA = 20;
        int girlsWithGradeA = gradeA - boysWithGradeA;
        System.out.println("Number of girls who got grade A: " + girlsWithGradeA);
}
}
```

4) **Take name, roll number, and field of interest. Print formatted output.**

```
import java.util.Scanner;
```

```
public class Question4 {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter your name: ");
        String name = sc.nextLine();
        System.out.print("Enter your roll number: ");
        String roll = sc.nextLine();
        System.out.print("Enter your field of interest: ");
        String interest = sc.nextLine();


        System.out.println("Hey, my name is " + name + " and my roll number is " + roll + ". My field of interest are " +
interest + ".");
    }
}
```

---

5) **Bonus of 10% if service > 6 years. Input: salary and years of service.**

```
import java.util.Scanner;


public class Question5 {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter your salary: ");
        double salary = sc.nextDouble();
        System.out.print("Enter your years of service: ");
        int years = sc.nextInt();
        if (years > 6) {
            double bonus = salary * 0.10;
            System.out.println("Bonus: " + bonus);
        } else {
            System.out.println("No bonus.");
        }
    }
}
```

---

6) **Grading system based on marks:**

```
import java.util.Scanner;
public class Question6 {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter your marks: ");
        int marks = sc.nextInt();
        if (marks < 25)
            System.out.println("Grade: F");
        else if (marks <= 45)
            System.out.println("Grade: E");
        else if (marks <= 50)
            System.out.println("Grade: D");
        else if (marks <= 60)
            System.out.println("Grade: C");
        else if (marks <= 80)
            System.out.println("Grade: B");
        else
            System.out.println("Grade: A");
    }
}
```

---

7) **Student attendance check.**

```
import java.util.Scanner;
public class Question7 {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Number of classes held: ");
        int held = sc.nextInt();
        System.out.print("Number of classes attended: ");
        int attended = sc.nextInt();
        double percentage = ((double) attended / held) * 100;
        System.out.println("Attendance Percentage: " + percentage);
        if (percentage >= 70)
            System.out.println("Allowed to sit in exam.");
        else
            System.out.println("Not allowed to sit in exam.");
    }
}
```

8) **Modify Q7 with medical cause option ('Y' or 'N').**

```
import java.util.Scanner;
public class Question8 {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Number of classes held: ");
        int held = sc.nextInt();
        System.out.print("Number of classes attended: ");
        int attended = sc.nextInt();
        double percentage = ((double) attended / held) * 100;
        System.out.println("Attendance Percentage: " + percentage);
        if (percentage >= 70) {
            System.out.println("Allowed to sit in exam.");
        } else {
            System.out.print("Do you have medical cause (Y/N): ");
            char cause = sc.next().charAt(0);
            if (cause == 'Y' || cause == 'y')
                System.out.println("Allowed to sit in exam due to medical cause.");
            else
                System.out.println("Not allowed to sit in exam.");
        }
    }
}
```

9) **Retail shop switch-case program.**

```
import java.util.Scanner;
public class Question9 {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        double total = 0;
        while (true) {
            System.out.print("Enter product number (1-3, 0 to exit): ");
            int product = sc.nextInt();
            if (product == 0) break;
            System.out.print("Enter quantity: ");
            int qty = sc.nextInt();
            double price = 0;
            switch (product) {
                case 1:
                    price = 22.50;
                    break;
                case 2:
                    price = 44.50;
                    break;
                case 3:
                    price = 9.98;
                    break;
                default:
                    System.out.println("Invalid product.");
                    continue;
            }
            total += price * qty;
        }
        System.out.println("Total retail value: " + total);
    }
}
```

10) **Eggs calculation in gross, dozen, leftover.**

```
public class Question10 {
    public static void main(String[] args) {
        int eggs = 1342;  // You can also use args[0] from command line input
        int gross = eggs / 144;
        int remainder = eggs % 144;
        int dozen = remainder / 12;
        int left = remainder % 12;
        System.out.println("Your number of eggs is " + gross + " gross, " + dozen + " dozen, and " + left);
    }
}
```

11) **Create a class Calculator with add, diff, mul, div methods.**

```
public class Calculator {
    public void add(int a, int b)
{
        System.out.println("Addition: " + (a + b));
    }
public void diff(int a, int b) {
        System.out.println("Difference: " + (a - b));
    }
    public void mul(int a, int b) {
        System.out.println("Multiplication: " + (a * b));
    }
    public void div(int a, int b) {
        if (b != 0)
            System.out.println("Division: " + (a / b));
        else
            System.out.println("Division by zero not allowed");
    }
    public static void main(String[] args) {
        Calculator calc = new Calculator();
        calc.add(10, 5);
        calc.diff(10, 5);
        calc.mul(10, 5);
        calc.div(10, 5);
    }
}
```

12) **Student class with RollNo auto-generation + class Standard with 8 students and multiple operations.**

```
class Student {
    private static int count = 1;
    private int rollNo;
    private String name;
    private int eng, maths, sci;
    public Student(String name, int eng, int maths, int sci) {
        this.rollNo = count++;
        this.name = name;
        this.eng = eng;
        this.maths = maths;
        this.sci = sci;
    }
    public int getTotal() {
        return eng + maths + sci;
    }
    public double getPercentage() {
        return getTotal() / 3.0;
    }
    public int getRollNo() {
        return rollNo;
    }
    public String getName() {
        return name;
    }
    public int getMaths() {
        return maths;
    }
    public void printStudent() {
        System.out.println("RollNo: " + rollNo + ", Name: " + name);
    }
}
```

Main Class:

```
import java.util.*;
public class Standard {
    private List<Student> students = new ArrayList<>();
    public Standard() {
        students.add(new Student("A", 80, 90, 85));
        students.add(new Student("B", 70, 88, 82));
        students.add(new Student("C", 65, 95, 89));
        students.add(new Student("D", 92, 78, 80));
        students.add(new Student("E", 55, 85, 75));
        students.add(new Student("F", 89, 79, 90));
        students.add(new Student("G", 76, 87, 88));
        students.add(new Student("H", 60, 82, 77));
    }
    public void displayByRollNo() {
        students.sort(Comparator.comparingInt(Student::getRollNo));
        students.forEach(Student::printStudent);
    }
    public void highestPercentage() {
        Student top = Collections.max(students, Comparator.comparingDouble(Student::getPercentage));
        System.out.println("Topper: " + top.getName());
    }
    public void highestMathMarks() {
        Student top = Collections.max(students, Comparator.comparingInt(Student::getMaths));
        System.out.println("Highest in Maths: " + top.getName());
    }
    public void mathSciSort() {
        students.sort(Comparator.comparingInt(s -> s.getMaths() + s.getTotal()));
        students.forEach(Student::printStudent);
    }
    public void displayRanks() {
        students.sort((s1, s2) -> Double.compare(s2.getPercentage(), s1.getPercentage()));
        int rank = 1;
        for (Student s : students) {
            System.out.println("Rank " + rank++ + ": " + s.getName() + " - " + s.getPercentage() + "%");
        }
    }
    public static void main(String[] args) {
        Standard std = new Standard();
        std.displayByRollNo();
        std.highestPercentage();
        std.highestMathMarks();
        std.mathSciSort();
        std.displayRanks();
    }
}
```

13) **Worker, DailyWorker, and SalariedWorker inheritance and pay logic.**

```
class Worker {
    String name;
    double salaryRate;
    public Worker(String name, double salaryRate) {
        this.name = name;
        this.salaryRate = salaryRate;
    }
    public double pay(int hours) {
        return 0;
    }
}
class DailyWorker extends Worker {
    public DailyWorker(String name, double rate) {
        super(name, rate);
    }
    @Override
    public double pay(int days) {
        return days * salaryRate;
    }
}
class SalariedWorker extends Worker {
    public SalariedWorker(String name, double rate) {
        super(name, rate);
    }
    @Override
    public double pay(int hours) {
        return 40 * salaryRate;
    }
}
```

Main Class:

```
public class Question13 {
    public static void main(String[] args) {
        Worker w1 = new DailyWorker("Ravi", 500);
        Worker w2 = new SalariedWorker("Teja", 600);
        System.out.println("DailyWorker Pay: " + w1.pay(6));
        System.out.println("SalariedWorker Pay: " + w2.pay(50));
    }
}
```

---

14) **Shape class with overloaded methods for square and rectangle.**

```
class Shape {
    public void area(int side) {
        System.out.println("Area of Square: " + (side * side));
    }
    public void area(int length, int breadth) {
        System.out.println("Area of Rectangle: " + (length * breadth));
    }
    public void perimeter(int side) {
        System.out.println("Perimeter of Square: " + (4 * side));
    }
    public void perimeter(int length, int breadth) {
        System.out.println("Perimeter of Rectangle: " + (2 * (length + breadth)));
    }
}
```

Main Class:

```
public class Question14 {
    public static void main(String[] args) {
        Shape s = new Shape();
        s.area(5);
        s.area(5, 3);
        s.perimeter(5);
        s.perimeter(5, 3);
    }
}
```

---

15) Count frequency of elements in an array.

```
import java.util.HashMap;
import java.util.Map;
public class Question15 {
    public static void main(String[] args) {
        int[] arr = {3, 5, 3, 7, 5, 5, 2, 1, 2, 3};
        Map<Integer, Integer> freq = new HashMap<>();
        for (int num : arr)
            freq.put(num, freq.getOrDefault(num, 0) + 1);
        for (Map.Entry<Integer, Integer> entry : freq.entrySet())
            System.out.println(entry.getKey() + " occurs " + entry.getValue() + " times");
    }
}
```

16) **Input marks for 3 students and compute average with validation.**

```
import java.util.Scanner;
public class Question16 {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int[] marks = new int[3];
        int i = 0;
        while (i < 3) {
            System.out.print("Enter mark (0–100) for student " + (i + 1) + ": ");
            int m = sc.nextInt();
            if (m >= 0 && m <= 100) {
                marks[i++] = m;
            } else {
                System.out.println("Invalid input, try again...");
            }
        }
        double avg = (marks[0] + marks[1] + marks[2]) / 3.0;
        System.out.printf("The average is: %.2f%n", avg);
    }
}
```

---

17) **Vehicle inheritance with common and specific methods.**

```
class Vehicle {
    String color;
    int wheels;
    String model;
    public Vehicle(String color, int wheels, String model) {
        this.color = color;
        this.wheels = wheels;
        this.model = model;
    }
    public void displayInfo() {
        System.out.println(model + " with color " + color + " and " + wheels + " wheels.");
    }
}
```

```
class Truck extends Vehicle {
    public Truck() {
        super("Red", 6, "Truck");
    }
}
class Car extends Vehicle {
    public Car() {
        super("Blue", 4, "Car");
    }
}
class Bus extends Vehicle {
    public Bus() {
        super("Yellow", 8, "Bus");
    }
}
```

Main Class:

```
public class Road {
    public static void main(String[] args) {
        Vehicle v1 = new Truck();
        Vehicle v2 = new Car();
        Vehicle v3 = new Bus();
        v1.displayInfo();
        v2.displayInfo();
        v3.displayInfo();
    }
}
```

---

18) **Package org.animals and zoo.VandalurZoo.**

```
package org.animals;
public class Lion {
    public void sound() {
        System.out.println("Lion Roars");
    }
    public boolean isVegetarian() {
        return false;
    }
public boolean canClimb() {
        return false;
    }
}
```

Main Class :

```
import org.animals.Lion;

public class VandalurZoo {
    public static void main(String[] args) {
        Lion lion = new Lion();
        lion.sound();
        System.out.println("Is vegetarian: " + lion.isVegetarian());
        System.out.println("Can climb: " + lion.canClimb());
    }
}
```

---

19) **Bank Superclass, Saving and Current subclasses, with balance calculation and customer search.**

```
abstract class Bank {
    String accNo;
    String custName;
    int custGender; // 1=Male, 2=Female
    String custJob;
    double curBal;
    Bank(String accNo, String custName, int custGender, String custJob, double curBal) {
        this.accNo = accNo;
        this.custName = custName;
        this.custGender = custGender;
        this.custJob = custJob;
        this.curBal = curBal;
    }
    public abstract double calcBalance();
    @Override
    public String toString() {
        return "AccNo: " + accNo + ", Name: " + custName + ", Gender: " + custGender + ", Job: " + custJob + ", Balance: " +
curBal;
    }
}
```

```
class Saving extends Bank {
    double savRate;
    Saving(String accNo, String custName, int gender, String job, double curBal, double savRate) {
        super(accNo, custName, gender, job, curBal);
        this.savRate = savRate;
    }
    public double calcBalance() {
        return curBal + (savRate * curBal);
    }
}
class Current extends Bank {
    boolean fixedDep;
    double curRate;
    Current(String accNo, String custName, int gender, String job, double curBal, boolean fixedDep, double curRate) {
        super(accNo, custName, gender, job, curBal);
        this.fixedDep = fixedDep;
        this.curRate = curRate;
    }
    public double calcBalance() {
        double balance = curBal + (curRate * curBal);
        if (fixedDep) {
            balance -= 150;
        }
        return balance;
    }
}
```

Main Class :

```
import java.util.*;
public class BankMain {
    public static void main(String[] args) {
        List<Bank> customers = new ArrayList<>();
        customers.add(new Saving("S101", "Ravi", 1, "Dev", 10000, 0.05));
        customers.add(new Current("C102", "Priya", 2, "HR", 20000, true, 0.04));
        customers.add(new Current("C103", "Arun", 1, "Sales", 15000, false, 0.03));
        // a) Calculate balances
        for (Bank b : customers) {
            System.out.println(b + ", Final Balance: " + b.calcBalance());
        }
        // b) Search by account number
        Scanner sc = new Scanner(System.in);
        System.out.print("\nEnter Account Number to search: ");
        String searchAcc = sc.next();
        boolean found = false;
        for (Bank b : customers) {
            if (b.accNo.equalsIgnoreCase(searchAcc)) {
                System.out.println("Customer found: " + b + ", Balance: " + b.calcBalance());
                found = true;
                break;
            }
        }
        if (!found) {
            System.out.println("Account not found.");
        }

        // c) Count Current account customers and total balance
        int count = 0;
        double totalBal = 0;
        for (Bank b : customers) {
            if (b instanceof Current) {
                count++;
                totalBal += b.calcBalance();
            }
        }
        System.out.println("\nCurrent account holders: " + count + ", Total balance: " + totalBal);
    }
}
```

20) **Abstract class Vehicle with subclasses Car and Motorcycle.**

```
abstract class Vehicle {
    abstract void startEngine();
    abstract void stopEngine();
}
class Car extends Vehicle {
    void startEngine() {
        System.out.println("Car engine started.");
    }
    void stopEngine() {
        System.out.println("Car engine stopped.");
    }
}
class Motorcycle extends Vehicle {
    void startEngine() {
        System.out.println("Motorcycle engine started.");
    }
    void stopEngine() {
        System.out.println("Motorcycle engine stopped.");
    }
}
```

Main Class :

```
public class Question20 {
    public static void main(String[] args) {
        Vehicle v1 = new Car();
        Vehicle v2 = new Motorcycle();

        v1.startEngine();
        v1.stopEngine();

        v2.startEngine();
        v2.stopEngine();
    }
}
```

---

21) **Abstract class Person with subclasses Athlete, LazyPerson.**

```
abstract class Person {
    abstract void eat();
    abstract void exercise();
}
class Athlete extends Person {
    void eat() {
        System.out.println("Athlete eats protein-rich food.");
    }
    void exercise() {
        System.out.println("Athlete exercises daily.");
    }
}
class LazyPerson extends Person {
    void eat() {
        System.out.println("Lazy person eats junk food.");
    }

    void exercise() {
        System.out.println("Lazy person rarely exercises.");
    }
}
```

Main Class:

```
public class Question21 {
    public static void main(String[] args) {
        Person p1 = new Athlete();
        Person p2 = new LazyPerson();

        p1.eat();
        p1.exercise();

        p2.eat();
        p2.exercise();
    }
}
```

---

22) **Interfaces Drawable and Fillable with implementations in shapes.**

```
interface Drawable {
    void drawingColor();
    void thickness();
}
interface Fillable {
    void fillingColor();
    void size();
}
class Circle implements Drawable, Fillable {
    public void drawingColor() {
        System.out.println("Circle Drawing Color: Red");
    }
    public void thickness() {
        System.out.println("Circle Thickness: 2px");
    }
    public void fillingColor() {
        System.out.println("Circle Filling Color: Blue");
    }
    public void size() {
        System.out.println("Circle Size: Medium");
    }
}
```

<u>Main Class:</u>

```
public class Question22 {
    public static void main(String[] args) {
        Circle c = new Circle();
        c.drawingColor();
        c.thickness();
        c.fillingColor();
        c.size();
    }
}
```

---

23) **Package house with classes Hall and Kitchen.**

```
import static java.lang.System.out;
public class Hall {
    public void enter() {
        out.println("This is the first room while entering the house.");
    }
}
// house/Kitchen.java
package house;
```

```
public class Kitchen {
    public void showAppliances() {
        String[] appliances = {"Microwave", "Oven", "Mixer"};
        System.out.println("Appliances: ");
        for (String app : appliances) {
            System.out.println(app);
        }
        // Copying to another array
        String[] copy = appliances.clone();
        System.out.println("Copied Array: ");
        for (String c : copy) {
            System.out.println(c);
        }
    }
}
```

Main Class:

```
// house/MainHouse.java
import house.Hall;
import house.Kitchen;

public class MainHouse {
    public static void main(String[] args) {
        Hall h = new Hall();
        Kitchen k = new Kitchen();

        h.enter();
        k.showAppliances();
    }
}
```

24) **5 bikers' speed — print those with speed > average.**

```
import java.util.Scanner;
public class Question24 {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int[] speed = new int[5];
        int total = 0;
        for (int i = 0; i < 5; i++) {
            System.out.print("Enter speed of biker " + (i + 1) + ": ");
            speed[i] = sc.nextInt();
            total += speed[i];
        }
        double avg = total / 5.0;
        System.out.println("Average speed: " + avg);
        System.out.println("Qualifying bikers:");

        for (int i = 0; i < 5; i++) {
            if (speed[i] > avg)
                System.out.println("Biker " + (i + 1) + ": " + speed[i]);
        }
    }
}
```

25) **Class MyTriangle with area, perimeter, isValid methods and validation.**

```
import java.util.Scanner;
public class MyTriangle {
    public static double area(double a, double b, double c) {
        double s = (a + b + c) / 2;
        return Math.sqrt(s * (s - a) * (s - b) * (s - c));
    }
    public static double perimeter(double a, double b, double c) {
        return a + b + c;
    }
    public static boolean isValid(double a, double b, double c) {
        return (a + b > c && a + c > b && b + c > a);
    }
```

Main Class:

```
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        while (true) {
            System.out.print("Enter side a (or -1 to exit): ");
            double a = sc.nextDouble();
            if (a == -1) {
                System.out.println("Bye~");
                break;
            }
            System.out.print("Enter side b: ");
            double b = sc.nextDouble();
            System.out.print("Enter side c: ");
            double c = sc.nextDouble();

            if (isValid(a, b, c)) {
                System.out.println("Area: " + area(a, b, c));
                System.out.println("Perimeter: " + perimeter(a, b, c));
            } else {
                System.out.println("The input is invalid.");
            }
        }
    }
}
```

---

26) **Remove duplicates from array and sum of even numbers.**

```
import java.util.*;
public class Question26 {
    public static void main(String[] args) {
        int[] input = {2, 3, 54, 1, 6, 7, 7};
        Set<Integer> set = new HashSet<>();
        int evenSum = 0;
        for (int num : input) {
            if (set.add(num) && num % 2 == 0) {
                evenSum += num;
            }
        }
        System.out.println("Sum of even unique numbers: " + evenSum);
    }
}
```

27) **Enum for paper currencies, loop and switch.**

```
enum Currency {
    ONE, FIVE, TEN, TWENTY, FIFTY, HUNDRED
}
```

Main Class:

```
public class Question27 {
    public static void main(String[] args) {
        for (Currency c : Currency.values()) {
            System.out.print(c + ": ");
            switch (c) {
                case ONE -> System.out.println("₹1 note");
                case FIVE -> System.out.println("₹5 note");
                case TEN -> System.out.println("₹10 note");
                case TWENTY -> System.out.println("₹20 note");
                case FIFTY -> System.out.println("₹50 note");
                case HUNDRED -> System.out.println("₹100 note");
            }
        }
    }
}
```

---

28) **Lambda expressions for isOdd, isPrime, isPalindrome.**

```
interface PerformOperation {
    boolean check(int a);
}
public class Question28 {
    public static void main(String[] args) {
        PerformOperation isOdd = (a) -> a % 2 != 0;
        PerformOperation isPrime = (a) -> {
            if (a < 2) return false;
            for (int i = 2; i <= Math.sqrt(a); i++) {
                if (a % i == 0) return false;
            }
            return true;
        };
        PerformOperation isPalindrome = (a) -> {
            String s = Integer.toString(a);
            return s.equals(new StringBuilder(s).reverse().toString());
        };
        System.out.println("Is 5 odd? " + isOdd.check(5));
```

```java
        System.out.println("Is 7 prime? " + isPrime.check(7));
        System.out.println("Is 121 palindrome? " + isPalindrome.check(121));
    }
}
```

---

29) **Validate register number (9 chars) and mobile number (10 digits). Throw exceptions.**

```java
import java.util.NoSuchElementException;
public class Question29 {
    public static void main(String[] args) {
        String regNo = "21ECE102A";
        String mobile = "9876543210";
        try {
            if (regNo.length() != 9)
                throw new IllegalArgumentException("Invalid Register Number length");
            if (!regNo.matches("[a-zA-Z0-9]+"))
                throw new NoSuchElementException("Register number has invalid characters");
            if (mobile.length() != 10)
                throw new IllegalArgumentException("Mobile number must be 10 digits");
            if (!mobile.matches("\\d+"))
                throw new NumberFormatException("Mobile number must be digits only");
            System.out.println("Valid");
        } catch (Exception e) {
            System.out.println("Invalid: " + e.getMessage());
        }
    }
}
```

---

30) **Interface method to return min of 3 float numbers using method reference.**

```java
import java.util.function.Function;
@FunctionalInterface
interface Minimum3 {
    float getMin(float a, float b, float c);
}
```

<u>Main Class:</u>

```java
public class Question30 {
    public static void main(String[] args) {
        Minimum3 min3 = (a, b, c) -> Math.min(a, Math.min(b, c));
        System.out.println("Minimum is: " + min3.getMin(1.1f, 0.5f, 2.0f)) }  }
```

31) **Demonstrate InputMismatchException and StringIndexOutOfBoundsException.**

```
import java.util.InputMismatchException;
import java.util.Scanner;
public class Question31 {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        try {
            System.out.print("Enter an integer: ");
            int num = sc.nextInt();
            String str = "Java";
            System.out.println("Character at index 10: " + str.charAt(10)); // Will throw exception
        } catch (InputMismatchException e) {
            System.out.println("Caught InputMismatchException: " + e);
        } catch (StringIndexOutOfBoundsException e) {
            System.out.println("Caught StringIndexOutOfBoundsException: " + e);
        }
    }
}
```

---

32) **Multiple catch blocks and stack trace for various exceptions.**

```
public class Question32 {
    public static void main(String[] args) {
        try {
            int[] arr = new int[-5]; // NegativeArraySizeException
            arr[10] = 5;             // ArrayIndexOutOfBoundsException
            String str = null;
            str.length();            // NullPointerException
            System.out.println(10 / 0); // ArithmeticException
        } catch (NegativeArraySizeException e) {
            e.printStackTrace();
        } catch (ArrayIndexOutOfBoundsException e) {
            e.printStackTrace();
        } catch (NullPointerException e) {
            e.printStackTrace();
        } catch (ArithmeticException e) {
            e.printStackTrace();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

---

33) **Class Emp with HRA based on designation and custom exception for low salary.**

```
class LowSalException extends Exception {
    public LowSalException(String msg) {
        super(msg);
    }
}
class Emp {
    int empId;
    String empName;
    String designation;
    double basic;
    double hra;
    public Emp(int empId, String designation, double basic) throws LowSalException {
        if (basic < 50000) {
            throw new LowSalException("Salary is less than 50000!");
        }
        this.empId = empId;
        this.designation = designation;
        this.basic = basic;
        calculateHRA();
    }
    void calculateHRA() {
        switch (designation) {
            case "Manager" -> hra = 0.10 * basic;
            case "TeamLeader" -> hra = 0.12 * basic;
            case "HR" -> hra = 0.05 * basic;
        }
    }
    void printDET() {
        System.out.println("Emp ID: " + empId + ", Designation: " + designation + ", Basic: " + basic + ", HRA: " + hra);
    }
}
```

Main Class :

```
public class Question33 {
    public static void main(String[] args) {
        try {
            Emp e = new Emp(101, "Manager", 60000);
            e.printDET();
        } catch (LowSalException e) {
            System.out.println(e.getMessage());
        }
    }
}
```

---

34) **Employee class with ID format year-designation-number.**

```
public class Employee {
    String empId;
    String name;
    int birthYear;

    public Employee(String empId, String name, int birthYear) {
        this.empId = empId;
        this.name = name;
        this.birthYear = birthYear;
    }

    public void display() {
        System.out.println("Employee ID: " + empId);
        System.out.println("Name: " + name);
        System.out.println("Birth Year: " + birthYear);
    }

    public static void main(String[] args) {
        Employee e = new Employee("81-F-112", "Ravi", 1981);
        e.display();
    }
}
```

---

35) **Validate employee code using custom exception.**

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

Main Class:

```
public class Question35 {
    public static void main(String[] args) {
        String empCode = "81-F-112";
        try {
            if (!empCode.matches("\\d{2}-[FS]-\\d{3}")) {
                throw new InvalidEmployeeCode("Invalid Employee Code!");
            } else {
                System.out.println("Valid Employee Code: " + empCode);
            }
        } catch (InvalidEmployeeCode e) {
            System.out.println(e.getMessage());
        }
    }
}
```

---

36) **Class Tank with finalize method and termination check.**

```
class Tank {
    boolean released = false;
    void releaseTank() {
        released = true;
        System.out.println("Tank released successfully.");
    }
    @Override
    protected void finalize() throws Throwable {
        if (!released) {
            System.out.println("Error: Tank not released!");
        }
        super.finalize();
    }
}
```

Main Class:

```
public class Question36 {
    public static void main(String[] args) {
        Tank t1 = new Tank();
        Tank t2 = new Tank();
        t2 = null;
        t1 = null;
        System.gc();
    }
}
```

---

37) **Create a file named "batchmates" and store names.**

```
import java.io.FileWriter;
import java.io.IOException;
public class Question37 {
    public static void main(String[] args) {
        try {
            FileWriter writer = new FileWriter("batchmates.txt");
            writer.write("Ravi\nTeja\nSai\nKiran\nAnil\n");
            writer.close();
            System.out.println("Batchmates written to file.");
        } catch (IOException e) {
            System.out.println("Error: " + e.getMessage());
        }
    }
}
```

---

38) **Serialize and deserialize Employee object excluding emp_sal.**

```
import java.io.*;
class Employee implements Serializable {
    int emp_id;
    String emp_name;
    transient double emp_sal; // will not be serialized
    public Employee(int emp_id, String emp_name, double emp_sal) {
        this.emp_id = emp_id;
        this.emp_name = emp_name;
        this.emp_sal = emp_sal;
    }}
```

Main Class:

```
public class Question38 {
    public static void main(String[] args) {
        Employee e1 = new Employee(101, "Ravi", 50000);

        try (ObjectOutputStream oos = new ObjectOutputStream(new FileOutputStream("emp.ser"))) {
            oos.writeObject(e1);
            System.out.println("Employee Serialized.");
        } catch (IOException e) {
            e.printStackTrace();
        }
        try (ObjectInputStream ois = new ObjectInputStream(new FileInputStream("emp.ser"))) {
            Employee emp = (Employee) ois.readObject();
            System.out.println("Deserialized: ID=" + emp.emp_id + ", Name=" + emp.emp_name + ", Salary=" + emp.emp_sal);
        } catch (IOException | ClassNotFoundException e) {
            e.printStackTrace();
        }
    }
}
```

39) **Media Library using generics and non-generics.**

```
// Generic version
class Library<T> {
    private List<T> items = new ArrayList<>();
    public void add(T item) {
        items.add(item);
    }
    public List<T> getAll() {
        return items;
    }
}
// Without generics
class NonGenericLibrary {
    private List items = new ArrayList();
    public void add(Object item) {
        items.add(item);
    }
    public List getAll() {
        return items;
    }
}
```

Main Class:

```
public class Question39 {
    public static void main(String[] args) {
        Library<String> bookLibrary = new Library<>();
        bookLibrary.add("Java Book");
        bookLibrary.add("OOP Concepts");
        for (String book : bookLibrary.getAll()) {
            System.out.println(book);
        }
        NonGenericLibrary videoLibrary = new NonGenericLibrary();
        videoLibrary.add("Java Tutorial");
        System.out.println(videoLibrary.getAll());
    }
}
```

---

40) **CD class with Comparable — sort by singer name.**

```
import java.util.*;
class CD implements Comparable<CD> {
    String title;
    String singer;
    public CD(String title, String singer) {
        this.title = title;
        this.singer = singer;
    }
    public int compareTo(CD other) {
        return this.singer.compareTo(other.singer);
    }
    public String toString() {
        return "CD Title: " + title + ", Singer: " + singer;
    }
}
```

Main Class:

```
public class Question40 {
    public static void main(String[] args) {
        List<CD> cds = new ArrayList<>();
        cds.add(new CD("Hits", "Zayn"));
        cds.add(new CD("Best of", "Arijit"));
        cds.add(new CD("Melody", "KK"));
        Collections.sort(cds);
        for (CD cd : cds) {
            System.out.println(cd);
        }
    }
}
```

---

41) **BookStore class — sort books by name and id using Comparator.**

```
import java.util.*;
class Book {
    int bookId;
    String bookName;
    public Book(int id, String name) {
        this.bookId = id;
        this.bookName = name;
    }
    @Override
    public String toString() {
        return "BookID: " + bookId + ", BookName: " + bookName;
    }
}
```

Main Class:

```
public class Question41 {
    public static void main(String[] args) {
        List<Book> books = new ArrayList<>();
        books.add(new Book(103, "Java"));
        books.add(new Book(101, "Python"));
        books.add(new Book(102, "C++"));
        System.out.println("Sort by book name:");
        books.sort(Comparator.comparing(b -> b.bookName));
        books.forEach(System.out::println);
        System.out.println("\nSort by book ID:");
        books.sort(Comparator.comparingInt(b -> b.bookId));
        books.forEach(System.out::println);
    }
}
```

42) **Rethrowing exception demo with someMethod and someMethod2.**

```
public class Question42 {
    static void someMethod2() throws Exception {
        throw new Exception("Exception from someMethod2");
    }
    static void someMethod() throws Exception {
        try {
            someMethod2();
        } catch (Exception e) {
            System.out.println("Caught in someMethod, rethrowing...");
            throw e;
        }
    }
```

Main Class:

```
    public static void main(String[] args) {
        try {
            someMethod();
        } catch (Exception e) {
            System.out.println("Caught in main:");
            e.printStackTrace();
        }
    }
}
```

43) **Read file using BufferedReader and try-with-resources.**

```
import java.io.*;
public class Question43 {
    public static void main(String[] args) {
        try (BufferedReader br = new BufferedReader(new FileReader("batchmates.txt"))) {
            String line;
            while ((line = br.readLine()) != null) {
                System.out.println(line);
            }
        } catch (IOException e) {
            System.out.println("Error reading file: " + e.getMessage());
        }
    }
}
```

---

44) **Custom sorting Employee list by salary (desc) and name (alpha).**

```
import java.util.*;
class Employee {
    int id;
    String name;
    double salary;
    public Employee(int id, String name, double salary) {
        this.id = id;
        this.name = name;
        this.salary = salary;
    }
    public String toString() {
        return id + " " + name + " " + salary;
    }
}
```

Main Class:

```
public class Question44 {
    public static void main(String[] args) {
        List<Employee> list = Arrays.asList(
            new Employee(1, "Ravi", 60000),
            new Employee(2, "Anil", 70000),
            new Employee(3, "Kiran", 50000)
        );
        System.out.println("Sorted by salary descending:");
        list.sort((e1, e2) -> Double.compare(e2.salary, e1.salary));
        list.forEach(System.out::println);
        System.out.println("\nSorted by name alphabetically:");
        list.sort(Comparator.comparing(e -> e.name));
        list.forEach(System.out::println); }}
```

45) **Group students by department using Java 8 Stream.**

```
import java.util.*;
import java.util.stream.*;
class Student {
    int id;
    String name;
    String department;
    public Student(int id, String name, String dept) {
        this.id = id;
        this.name = name;
        this.department = dept;
    }
    public String toString() {
        return name;
    }
}
```

Main Class:

```
public class Question45 {
    public static void main(String[] args) {
        List<Student> students = Arrays.asList(
            new Student(1, "Ravi", "CSE"),
            new Student(2, "Teja", "ECE"),
            new Student(3, "Anil", "CSE"),
            new Student(4, "Kiran", "IT")
        );
        Map<String, List<Student>> grouped = students.stream()
            .collect(Collectors.groupingBy(s -> s.department));
        grouped.forEach((dept, list) -> {
            System.out.println(dept + ": " + list);
        });
    }
}
```

46) **Generic Repository<T, ID> with Employee class.**

```
import java.util.*;
class Repository<T, ID> {
    Map<ID, T> store = new HashMap<>();
    void save(ID id, T entity) {
        store.put(id, entity);
    }
    T findById(ID id) {
        return store.get(id);
    }
    List<T> findAll() {
        return new ArrayList<>(store.values());
    }
    void deleteById(ID id) {
        store.remove(id);
    }
}
class Employee {
    int id;
    String name;
    double salary;
    Employee(int id, String name, double sal) {
        this.id = id;
        this.name = name;
        this.salary = sal;
    }
    public String toString() {
        return id + " " + name + " " + salary;
    }
}
```

Main Class:

```
public class Question46 {
    public static void main(String[] args) {
        Repository<Employee, Integer> repo = new Repository<>();
        repo.save(1, new Employee(1, "Ravi", 50000));
        repo.save(2, new Employee(2, "Teja", 60000));
        repo.save(3, new Employee(3, "Kiran", 55000));
        System.out.println("All Employees:");
        repo.findAll().forEach(System.out::println);
        System.out.println("\nFind by ID 2:");
        System.out.println(repo.findById(2));
        repo.deleteById(1);
        System.out.println("\nAfter Deletion:");
        repo.findAll().forEach(System.out::println);
    }
}
```

---

47) **Custom checked exception InvalidSalaryException.**

```
class InvalidSalaryException extends Exception {
    public InvalidSalaryException(String msg) {
        super(msg);
    }
}
class Employee {
    String name;
    double salary;
    Employee(String name, double salary) {
        this.name = name;
        this.salary = salary;
    }
}
class EmployeeService {
    void validateSalary(double salary) throws InvalidSalaryException {
        if (salary < 0)
            throw new InvalidSalaryException("Salary cannot be negative.");
    }
    void processSalary(Employee emp) throws InvalidSalaryException {
        validateSalary(emp.salary);
    }
    void startProcess(Employee emp) throws InvalidSalaryException {
        processSalary(emp);
    }
}
```

Main Class:

```
public class Question47 {
    public static void main(String[] args) {
        Employee emp = new Employee("Ravi", -20000);
        EmployeeService service = new EmployeeService();
        try {
            service.startProcess(emp);
        } catch (InvalidSalaryException e) {
            System.out.println("Error: " + e.getMessage());
        }
    }
}
```

---

48) **Functional interface Employee Processor.**

```
interface EmployeeProcessor {
    void process(Employee e);
}
class Employee {
    String name;
    double salary;
    Employee(String n, double s) {
        name = n;
        salary = s;
    }
}
```

Main Class:

```
public class Question48 {
    public static void main(String[] args) {
        List<Employee> employees = List.of(
            new Employee("Ravi", 50000),
            new Employee("Kiran", 60000)
        );
        EmployeeProcessor printDetails = (e) ->
            System.out.println("Name: " + e.name + ", Salary: " + e.salary);

        EmployeeProcessor printBonus = (e) ->
            System.out.println("Bonus: " + (e.salary * 0.10));
for (Employee e : employees) {
            printDetails.process(e);
            printBonus.process(e);
        }
    }
}
```

---

49) **Optional fields in Employee.**

```
import java.util.Optional;
class Employee {
    String name;
    Optional<String> email;
    Optional<String> department;
    Employee(String name, Optional<String> email, Optional<String> dept) {
        this.name = name;
        this.email = email;
        this.department = dept;
    }
    void printInfo() {
        System.out.println("Name: " + name);
        System.out.println("Email: " + email.orElse("Not Provided"));
        System.out.println("Department: " + department.orElse("General"));
    }
}
```

Main Class:

```
public class Question49 {
    public static void main(String[] args) {
        Employee e1 = new Employee("Ravi", Optional.of("ravi@gmail.com"), Optional.empty());
        Employee e2 = new Employee("Teja", Optional.empty(), Optional.of("HR"));

        e1.printInfo();
        e2.printInfo();
    }
}
```

---

50) **Stream API operations on Employee list (multiple queries).**

```
import java.util.*;
import java.util.stream.*;
class Employee {
    int id;
    String name;
    String department;
    double salary;
    Employee(int id, String name, String department, double salary) {
        this.id = id;
        this.name = name;
        this.department = department;
        this.salary = salary;
    }
    @Override
    public String toString() {
        return name + " - " + department + " - ₹" + salary;
    }
}
public class EmployeeStreamDemo {
    public static void main(String[] args) {
        List<Employee> employees = Arrays.asList(
            new Employee(101, "Ravi", "HR", 50000),
            new Employee(102, "Priya", "IT", 60000),
            new Employee(103, "Arun", "HR", 55000),
            new Employee(104, "Kavya", "IT", 70000),
            new Employee(105, "Divya", "Sales", 45000)
        );

        // Q1
        System.out.println("Q1. All employee names:");
        employees.stream().map(e -> e.name).forEach(System.out::println);

        // Q2
        System.out.println("\nQ2. Salary > 55000:");
        employees.stream().filter(e -> e.salary > 55000).forEach(System.out::println);


        // Q3

        System.out.println("\nQ3. HR count:");

        long hrCount = employees.stream().filter(e -> e.department.equals("HR")).count();

        System.out.println("HR employees: " + hrCount);


        // Q4

        System.out.println("\nQ4. Sort by salary descending:");

        employees.stream().sorted(Comparator.comparingDouble(e -> -e.salary)).forEach(System.out::println);


        // Q5

        System.out.println("\nQ5. Highest paid employee:");

        employees.stream().max(Comparator.comparingDouble(e -> e.salary)).ifPresent(System.out::println);


        // Q6

        System.out.println("\nQ6. Average salary:");

        double avg = employees.stream().mapToDouble(e -> e.salary).average().orElse(0);

        System.out.println("Average Salary: ₹" + avg);


        // Q7

        System.out.println("\nQ7. All names to List:");

        List<String> names = employees.stream().map(e -> e.name).collect(Collectors.toList());

        System.out.println(names);


        // Q8

        System.out.println("\nQ8. Group by department:");

        Map<String, List<Employee>> grouped = employees.stream().collect(Collectors.groupingBy(e -> e.department));

        grouped.forEach((k, v) -> System.out.println(k + ": " + v));


        // Q9

        System.out.println("\nQ9. Total salary per department:");

        Map<String, Double> totalSalary = employees.stream().collect(Collectors.groupingBy(e -> e.department,
Collectors.summingDouble(e -> e.salary)));

        totalSalary.forEach((k, v) -> System.out.println(k + ": ₹" + v));
```

```
// Q10

System.out.println("\nQ10. IT employees sorted by salary:");

employees.stream()

        .filter(e -> e.department.equals("IT"))

        .sorted(Comparator.comparingDouble(e -> e.salary))

        .map(e -> e.name)

        .forEach(System.out::println);


// Q11

System.out.println("\nQ11. Any employee earning < 40000?");

boolean anyLow = employees.stream().anyMatch(e -> e.salary < 40000);

System.out.println(anyLow);


// Q12

System.out.println("\nQ12. Comma-separated employee names:");

String joined = employees.stream().map(e -> e.name).collect(Collectors.joining(", "));

System.out.println(joined);


// Q13

System.out.println("\nQ13. Top 2 highest earning employees:");

employees.stream().sorted(Comparator.comparingDouble(e -> -e.salary)).limit(2).forEach(System.out::println);


// Q14

System.out.println("\nQ14. Skip first 2 employees:");

employees.stream().skip(2).forEach(System.out::println);


// Q15

System.out.println("\nQ15. First 3 employee names:");

employees.stream().limit(3).map(e -> e.name).forEach(System.out::println);


// Q16

System.out.println("\nQ16. Min salary in HR department:");

employees.stream()

        .filter(e -> e.department.equals("HR"))
```

```
            .min(Comparator.comparingDouble(e -> e.salary))
            .ifPresent(System.out::println);

        // Q17
        System.out.println("\nQ17. Partition salary > 55000:");
        Map<Boolean, List<Employee>> partition = employees.stream().collect(Collectors.partitioningBy(e -> e.salary >
55000));
        partition.forEach((k, v) -> System.out.println((k ? "Above 55k" : "55k or below") + ": " + v));

        // Q18
        System.out.println("\nQ18. Average salary per department:");
        Map<String, Double> avgSalary = employees.stream().collect(Collectors.groupingBy(e -> e.department,
Collectors.averagingDouble(e -> e.salary)));
        avgSalary.forEach((k, v) -> System.out.println(k + ": ₹" + v));

        // Q19
        System.out.println("\nQ19. Sort by name then salary:");
        employees.stream()
            .sorted(Comparator.comparing(Employee::name).thenComparing(e -> e.salary))
            .forEach(System.out::println);

        // Q20
        System.out.println("\nQ20. Convert to Map<Id, Name>:");
        Map<Integer, String> idNameMap = employees.stream().collect(Collectors.toMap(e -> e.id, e -> e.name));
        idNameMap.forEach((k, v) -> System.out.println(k + ": " + v));

        // ◈ Challenge 1
        System.out.println("\nChallenge 1: Names starting with D and ending with a:");
        employees.stream()
            .filter(e -> e.name.startsWith("D") && e.name.endsWith("a"))
            .forEach(System.out::println);

        // ◈ Challenge 2
        System.out.println("\nChallenge 2: Departments with more than 1 employee:");
        grouped.entrySet().stream()
           .filter(entry -> entry.getValue().size() > 1)
```

```
                .forEach(e -> System.out.println(e.getKey() + ": " + e.getValue()));


        // ◈ Challenge 3
        System.out.println("\nChallenge 3: Second highest salary:");
        employees.stream()
                .map(e -> e.salary)
                .distinct()
                .sorted(Comparator.reverseOrder())
                .skip(1)
                .findFirst()
                .ifPresent(s -> System.out.println("Second Highest Salary: ₹" + s));
    }
}
```