1. Write a Program for Student with Grade Validation & Configuration

```
class Student {
    private String name;
    private int rollNumber;
    private int marks;

    public Student(String name, int rollNumber, int marks) {
        this.name = name;
        this.rollNumber = rollNumber;
        if (marks >= 0 && marks <= 100) {
            this.marks = marks;
        } else {
            this.marks = 0;
        }
    }

    public String getName() { return name; }
    public int getRollNumber() { return rollNumber; }
    public int getMarks() { return marks; }

    public void displayDetails() {
        System.out.println("Name: " + name);
        System.out.println("Roll No: " + rollNumber);
        System.out.println("Marks: " + marks);
    }
}

public class MainStudent {
    public static void main(String[] args) {
        Student s1 = new Student("Likitha", 101, 95);
        Student s2 = new Student("Ravi", 102, 120);
        s1.displayDetails();
        s2.displayDetails();
    }
}
```

Output:

```
Name: Likitha
Roll No: 101
Marks: 95
Name: Ravi
Roll No: 102
Marks: 0
```

---

2. Write a Program for Rectangle Enforced Positive Dimensions

```
class Rectangle {
    private double width;
    private double height;

    public Rectangle(double width, double height) {
        setWidth(width);
        setHeight(height);
    }

    public void setWidth(double width) {
        if (width > 0) this.width = width;
        else this.width = 1;
    }

    public void setHeight(double height) {
        if (height > 0) this.height = height;
        else this.height = 1;
    }

    public double getArea() { return width * height; }
    public double getPerimeter() { return 2 * (width + height); }

    public void displayDetails() {
        System.out.println("Width: " + width + ", Height: " + height);
        System.out.println("Area: " + getArea());
        System.out.println("Perimeter: " + getPerimeter());
    }
}

public class MainRectangle {
    public static void main(String[] args) {
        Rectangle r = new Rectangle(10, -5);
        r.displayDetails();
    }
}
```

Output:

```
Width: 10.0, Height: 1.0
Area: 10.0
Perimeter: 22.0
```

---

3. Write a Program for Bank Account with Deposit/Withdraw Logic

```
import java.util.ArrayList;
import java.util.List;

class BankAccount {
    private String accountNumber;
    private String accountHolder;
    private double balance;
    private List<String> transactionHistory = new ArrayList<>();

    public BankAccount(String accountNumber, String accountHolder, double balance) {
        this.accountNumber = accountNumber;
        this.accountHolder = accountHolder;
        this.balance = balance;
    }

    public void deposit(double amount) {
        if (amount > 0) {
            balance += amount;
            transactionHistory.add("Deposited: " + amount);
        }
    }

    public boolean withdraw(double amount) {
        if (amount > 0 && amount <= balance) {
            balance -= amount;
            transactionHistory.add("Withdrew: " + amount);
            return true;
        }
        return false;
    }

    public double getBalance() { return balance; }

    public String getLastTransaction() {
        if (!transactionHistory.isEmpty()) {
            return transactionHistory.get(transactionHistory.size() - 1);
        }
        return "No transactions yet.";
    }

    public String toString() {
        String maskedAcc = (“ “ + accountNumber.substring(accountNumber.length() - 4);
        return "Account: " + maskedAcc + ", Holder: " + accountHolder + ", Balance: " + balance;
    }
}

public class MainBank {
    public static void main(String[] args) {
        BankAccount acc = new BankAccount("1234567890", "Likitha", 5000);
        acc.deposit(2000);
        acc.withdraw(1000);
        System.out.println(acc);
        System.out.println("Last Transaction: " + acc.getLastTransaction());
    }
}
```

Output:

```
Account: 7890, Holder: Likitha, Balance: 6000.0
Last Transaction: Withdrew: 1000.0
```

---

4. Write a program for Inner Class Encapsulation: Secure Locker

```
class Locker {
    private String lockerId;
    private boolean isLocked;
    private String passcode;

    private class SecurityManager {
        private boolean verify(String code) {
            return passcode.equals(code);
        }
    }

    public Locker(String lockerId, String passcode) {
        this.lockerId = lockerId;
        this.passcode = passcode;
        this.isLocked = true;
    }

    public void lock() {
        isLocked = true;
        System.out.println("Locker locked.");
    }

    public void unlock(String code) {
        SecurityManager sm = new SecurityManager();
        if (sm.verify(code)) {
            isLocked = false;
            System.out.println("Locker unlocked.");
        } else {
            System.out.println("Invalid passcode.");
        }
    }

    public boolean isLocked() {
        return isLocked;
    }
}

public class MainLocker {
    public static void main(String[] args) {
        Locker locker = new Locker("L123", "secret");
        locker.unlock("wrong");
        locker.unlock("secret");
    }
}
```

Output:

```
Invalid passcode.
Locker unlocked.
```

5. Write a Program for Builder Pattern: Immutable Product

```
class Product {
    private final String name;
    private final String code;
    private final double price;
    private final String category;

    private Product(Builder builder) {
        this.name = builder.name;
        this.code = builder.code;
        this.price = builder.price;
        this.category = builder.category;
    }

    public String getName() { return name; }
    public String getCode() { return code; }
    public double getPrice() { return price; }
    public String getCategory() { return category; }

    public static class Builder {
        private String name;
        private String code;
        private double price;
        private String category;

        public Builder withName(String name) { this.name = name; return this; }
        public Builder withCode(String code) { this.code = code; return this; }
        public Builder withPrice(double price) { if (price >= 0) this.price = price; return this; }
        public Builder withCategory(String category) { this.category = category; return this; }

        public Product build() {
            if (name == null || code == null) throw new IllegalStateException("Name and Code are required.");

            return new Product(this);

        }

    }

}


public class MainProduct {

    public static void main(String[] args) {

        Product p = new Product.Builder()

                .withName("Laptop")

                .withCode("LP1001")

                .withPrice(55000)

                .withCategory("Electronics")

                .build();

        System.out.println("Product: " + p.getName() + ", Price: " + p.getPrice());

    }

}
```

Output:

```
Product: Laptop, Price: 55000.0
```

---

1. Write a Program for Reverse CharSequence

```
class BackwardSequence implements CharSequence {

    private String reversed;


    public BackwardSequence(String input) {

        this.reversed = new StringBuilder(input).reverse().toString();

    }


    public int length() { return reversed.length(); }

    public char charAt(int index) { return reversed.charAt(index); }

    public CharSequence subSequence(int start, int end) { return reversed.substring(start, end); }
    public String toString() { return reversed; }
}

public class MainBackward {
    public static void main(String[] args) {
        BackwardSequence b = new BackwardSequence("hello");
        System.out.println(b);
        System.out.println("Length: " + b.length());
        System.out.println("CharAt(1): " + b.charAt(1));
        System.out.println("SubSequence(1,4): " + b.subSequence(1,4));
    }
}
```

Output:

```
Length: 5
CharAt(1): l
SubSequence(1,4): lle
```

---

2. Write a Program for Moveable Shapes Simulation

```
interface Movable {
    void moveUp();
    void moveDown();
    void moveLeft();
    void moveRight();
}

class MovablePoint implements Movable {
    int x, y, xSpeed, ySpeed;

    public MovablePoint(int x, int y, int xSpeed, int ySpeed) {
        this.x = x;
        this.y = y;
        this.xSpeed = xSpeed;
        this.ySpeed = ySpeed;
    }

    public void moveUp() { y += ySpeed; }
    public void moveDown() { y -= ySpeed; }
    public void moveLeft() { x -= xSpeed; }
    public void moveRight() { x += xSpeed; }
    public String toString() { return "(" + x + "," + y + ")"; }
}

class MovableCircle implements Movable {
    int radius;
    MovablePoint center;

    public MovableCircle(int radius, MovablePoint center) {
        this.radius = radius;
        this.center = center;
    }

    public void moveUp() { center.moveUp(); }
    public void moveDown() { center.moveDown(); }
    public void moveLeft() { center.moveLeft(); }
    public void moveRight() { center.moveRight(); }
    public String toString() { return "Center: " + center + ", Radius: " + radius; }
}

public class MainMovable {
    public static void main(String[] args) {
        MovablePoint p = new MovablePoint(0, 0, 2, 2);
        MovableCircle c = new MovableCircle(5, p);
        System.out.println(c);
        c.moveUp();
        c.moveRight();
        System.out.println(c);
    }
}
```

Output:

```
Center: (0,0), Radius: 5
Center: (2,2), Radius: 5
```

---

3. Write a Program for Interface Printer Switch and its Subclasses

```
interface Printer {
    void print(String document);
}
class LaserPrinter implements Printer {
    public void print(String document) {
        System.out.println("Laser printing: " + document);
    }
}
class InkjetPrinter implements Printer {
    public void print(String document) {
        System.out.println("Inkjet printing: " + document);
    }
}
public class MainPrinter {
    public static void main(String[] args) {
        Printer p = new LaserPrinter();
        p.print("Hello World");
        p = new InkjetPrinter();
        p.print("Hello World");
    }
}
```

Output:

```
Laser printing: Hello World
```

Inkjet printing: Hello World

---

4. Write a program for Extended Interface Hierarchy

```
interface BaseVehicle {
    void start();
}
interface AdvancedVehicle extends BaseVehicle {
    void stop();
    boolean refuel(int amount);
}
class Car implements AdvancedVehicle {
    int fuel;
  public Car(int fuel) {
        this.fuel = fuel;
    }
public void start() {
        if (fuel > 0) System.out.println("Car started");
        else System.out.println("No fuel");
    }
  public void stop() {
        System.out.println("Car stopped");
    }
 public boolean refuel(int amount) {
        if (amount > 0) {
            fuel += amount;
            return true;
        }
        return false;
    }
}
public class MainVehicle {
    public static void main(String[] args) {
        Car car = new Car(10);
        car.start();
        car.stop();
        car.refuel(20);
        car.start();
    }
}
```

Output:

```
Car started
Car stopped
Car started
```

---

5. Write a Program for Nested Interface for Callback Handling

```
import java.time.LocalDateTime;
import java.util.ArrayList;
import java.util.List;
class TimeServer {
    public static interface Client {
        void updateTime(LocalDateTime now);
    }
private List<Client> clients = new ArrayList<>();
public void registerClient(Client client) {
        clients.add(client);
    }
 public void notifyClients() {
        LocalDateTime now = LocalDateTime.now();
        for (Client c : clients) {
            c.updateTime(now);
        }
    }
}
class ClientA implements TimeServer.Client {
    public void updateTime(LocalDateTime now) {
        System.out.println("ClientA time: " + now);
    }
}
class ClientB implements TimeServer.Client {
    public void updateTime(LocalDateTime now) {
        System.out.println("ClientB time: " + now);
    }
}
public class MainTimeServer {
    public static void main(String[] args) {
        TimeServer server = new TimeServer();
        server.registerClient(new ClientA());
        server.registerClient(new ClientB());
        server.notifyClients();
    }
}
```

Output

```
ClientA time: 2025-08-09T14:35:12.345
ClientB time: 2025-08-09T14:35:12.345
```

---

6. Write a Program for Default and Static Methods in Interfaces

```
interface Polygon {
    double getArea();
    default double getPerimeter(int... sides) {
        double sum = 0;
        for (int s : sides) sum += s;
        return sum;
    }
    static String shapeInfo() {
        return "Polygons have multiple sides";
    }
}
class RectangleShape implements Polygon {
    double width, height;
    public RectangleShape(double width, double height) {
        this.width = width;
        this.height = height;
    }
    public double getArea() {
        return width * height;
    }
}
class TriangleShape implements Polygon {
    double base, height;
    public TriangleShape(double base, double height) {
        this.base = base;
        this.height = height;
    }
    public double getArea() {
        return 0.5 * base * height;
    }
}
public class MainPolygon {
    public static void main(String[] args) {
        RectangleShape r = new RectangleShape(4,5);
        TriangleShape t = new TriangleShape(3,6);
        System.out.println("Rectangle area: " + r.getArea());
        System.out.println("Triangle area: " + t.getArea());
        System.out.println("Perimeter of rectangle: " + r.getPerimeter(4,5,4,5));
        System.out.println(Polygon.shapeInfo());
    }
}
```

Output:

```
Rectangle area: 20.0
Triangle area: 9.0
Perimeter of rectangle: 18.0
Polygons have multiple sides
```

---

1. Write a Program for Sum of Two Integers

```
interface SumCalculator {
    int sum(int a, int b);
}
public class MainSumLambda {
    public static void main(String[] args) {
        SumCalculator calc = (a, b) -> a + b;
        System.out.println(calc.sum(5, 7));
    }
}
```

Output:

```
12
```

---

2. Write a Program for Check If a String Is Empty

```
import java.util.function.Predicate;
public class MainIsEmpty {
    public static void main(String[] args) {
        Predicate<String> isEmpty = s -> s.isEmpty();
        System.out.println(isEmpty.test(""));
        System.out.println(isEmpty.test("hello"));
    }
}
```

Output:

```
true
false
```

---

3. Write a Program for Filter Even or Odd Numbers

```
import java.util.Arrays;
import java.util.List;
import java.util.stream.Collectors;
public class MainFilterEvenOdd {
    public static void main(String[] args) {
        List<Integer> numbers = Arrays.asList(1, 2, 3, 4, 5, 6);
        List<Integer> evens = numbers.stream().filter(n -> n % 2 == 0).collect(Collectors.toList());
        List<Integer> odds = numbers.stream().filter(n -> n % 2 != 0).collect(Collectors.toList());
        System.out.println(evens);
        System.out.println(odds);
    }
}
```

Output:

```
[2, 4, 6]
[1, 3, 5]
```

---

4. Write a Program for Convert Strings to Uppercase/Lowercase

```
import java.util.Arrays;
import java.util.List;
import java.util.stream.Collectors;

public class MainCaseConvert {
    public static void main(String[] args) {
        List<String> words = Arrays.asList("java", "lambda", "code");
        List<String> upper = words.stream().map(s -> s.toUpperCase()).collect(Collectors.toList());
        List<String> lower = words.stream().map(s -> s.toLowerCase()).collect(Collectors.toList());
        System.out.println(upper);
        System.out.println(lower);
    }
}
```

Output:

```
[JAVA, LAMBDA, CODE]
[java, lambda, code]
```

---

5. Write a Program for Strings by Length or Alphabetically

```
import java.util.Arrays;
import java.util.List;
import java.util.stream.Collectors;
public class MainSortStrings {
    public static void main(String[] args) {
        List<String> words = Arrays.asList("banana", "apple", "kiwi", "grape");
        List<String> byLength = words.stream().sorted((a, b) -> a.length() - b.length()).collect(Collectors.toList());
        List<String> alphabetical = words.stream().sorted().collect(Collectors.toList());
        System.out.println(byLength);
        System.out.println(alphabetical);
    }
}
```

Output:

```
[kiwi, grape, apple, banana]
[apple, banana, grape, kiwi]
```

---

6. Write a Program for Aggregate Operations (Sum, Max, Average) on Double Arrays

```
import java.util.Arrays;
public class MainAggregate {
    public static void main(String[] args) {
        double[] nums = {1.5, 2.5, 3.5};
        double sum = Arrays.stream(nums).sum();
        double max = Arrays.stream(nums).max().getAsDouble();
        double avg = Arrays.stream(nums).average().getAsDouble();
        System.out.println(sum);
        System.out.println(max);
        System.out.println(avg);
    }
}
```

Output:

```
7.5
3.5
2.5
```

---

7. Write a Program for Create Similar Lambdas for Max/Min

```
import java.util.Arrays;
import java.util.List;
import java.util.function.BinaryOperator;
public class MainMaxMinLambda {
    public static void main(String[] args) {
        BinaryOperator<Integer> max = (a, b) -> a > b ? a : b;
        BinaryOperator<Integer> min = (a, b) -> a < b ? a : b;
        System.out.println(max.apply(5, 9));
        System.out.println(min.apply(5, 9));
    }
}
```

Output:

```
9
5
```

---

8. Write a Program for Calculate Factorial

```
import java.util.stream.IntStream;
public class MainFactorial {
    public static void main(String[] args) {
        int num = 5;
        int fact = IntStream.rangeClosed(1, num).reduce(1, (a, b) -> a * b);
        System.out.println(fact);
    }
}
```

Output:120.