DAY 5

1) 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 {
         throw new IllegalArgumentException("Marks must be between 0 and 100.");
      }
   }
   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("Harshu", 1, 110);
      s1.displayDetails();
   }
}
```

Output:

Name: Harshu Roll No: 101 Marks: 110

2) Rectangle

```
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;
   }
   public void setHeight(double height) {
      if (height > 0) this.height = height;
   }
   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: 0.0 Area: 0.0 Perimeter: 20.0

3) Bank Account

```
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("1458745890", "Harshu", 5000);
      acc.deposit(2000);
      acc.withdraw(1000);
      System.out.println(acc);
      System.out.println("Last Transaction: " + acc.getLastTransaction());
   }
}
```

Output:

Account: ***5890, Holder: Harshu, Balance: 6000.0

Last Transaction: Withdrew: 1000.0

4) 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) 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) { 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("Mobile")
            .withCode("Mo1001")
            .withPrice(65000)
            .withCategory("Electronics")
            .build();
        System.out.println("Product: " + p.getName() + ", Price: " + p.getPrice());
    }
}
```

Output:
Product: Mobile, Price: 65000.0

6) Reverse CharSequence

```
class BackwardSequence implements CharSequence {
    private String reversed;
    public BackwardSequence(String input) {
        StringBuilder sb = new StringBuilder();
        for (int i = input.length() - 1; i >= 0; i--) {
            sb.append(input.charAt(i));
        }
        this.reversed = sb.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("harshu");
        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:
uhsrah
Length: 6
CharAt(1): l
SubSequence(1,4): hsr

7) 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(“Hello”,"Harshu", "Shetty");
        List<String> upper = words.stream().map(String::toUpperCase).collect(Collectors.toList());
        System.out.println(upper);
    }
}
```

Output:
[HELLO,HARSHU,SHETTY]

8) Aggregate Operations

```
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).reduce(0.0, (a, b) -> a + b);
        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

9) Calculate Factorial

```
public class MainFactorial {
    public static void main(String[] args) {
        int num = 5;
        int fact = 1;
        for (int i = 1; i <= num; i++) {
            fact *= i;
        }
        System.out.println(fact);
    }
}
```

Output:

120

10) Create Similar Lambdas for Max/Min

```
import java.util.function.BinaryOperator;
public class MainMaxMinLambda {
    public static void main(String[] args) {
        BinaryOperator<Integer> max = Math::max;
        BinaryOperator<Integer> min = (a, b) -> a < b ? a : b;
        System.out.println(max.apply(5, 89));
        System.out.println(min.apply(-5, 9));
    }
}
```

Output:

89

-5

11) 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) -> b.length() - a.length()).collect(Collectors.toList());
        List<String> alphabetical = words.stream().sorted().collect(Collectors.toList());
        System.out.println(byLength);
        System.out.println(alphabetical);
    }
}
```

Output:

[banana, grape, apple, kiwi]

[apple, banana, grape, kiwi]

12) Check If a String Is Empty

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

Output:

true

false

13) Filter Even or Odd Numbers

```
import java.util.Arrays;
import java.util.List;
import java.util.Map;
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);
    Map<Boolean, List<Integer>> partitioned = numbers.stream().collect(Collectors.partitioningBy(n -> n % 2 == 0));
    System.out.println(partitioned.get(true));
    System.out.println(partitioned.get(false));
  }
}
```

Output:

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

14) 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() {
    if (!clients.isEmpty()) {
      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: 2025-08-10T22:08:08.571400
ClientB time: 2025-08-10T22:08:08.571400

15) Extended Interface Hierarchy

```
interface BaseVehicle {
  void start();
}
interface AdvancedVehicle extends BaseVehicle {
  void stop();
```

```
    boolean refuel(int amount);
}
class Car implements AdvancedVehicle {
    int fuel;
    boolean isStarted = false;
    public Car(int fuel) {
        this.fuel = fuel;
    }
    public void start() {
        if (fuel > 0 && !isStarted) {
            System.out.println("Car started");
            isStarted = true;
        } else if (fuel <= 0) {
            System.out.println("No fuel");
        } else {
            System.out.println("Car is already running");
        }
    }
    public void stop() {
        System.out.println("Car stopped");
        isStarted = false;
    }
    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.start();
        car.stop();
        car.refuel(20);
        car.start();
    }
}
```

Output:

Car started
Car is already running
Car stopped
Car started

16) Default and Static Methods

```
import java.util.Arrays;
import java.util.stream.IntStream;
interface Polygon {
    double getArea();
    default double getPerimeter(int... sides) {
        return IntStream.of(sides).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 Main {
    public static void main(String[] args) {
        RectangleShape r = new RectangleShape(5, 5);
        TriangleShape t = new TriangleShape(3, 2);
        System.out.println("Rectangle area: " + r.getArea());
        System.out.println("Triangle area: " + t.getArea());
        System.out.println("Perimeter of rectangle: " + r.getPerimeter(4, 2, 3, 5));
        System.out.println(Polygon.shapeInfo());
    }
}
```

Output:

Rectangle area: 25.0
Triangle area: 5.0
Perimeter of rectangle: 14.0
Polygons have multiple sides

17) Sum of Two Integers

```
import java.util.function.BinaryOperator;
interface SumCalculator {
    int sum(int a, int b);
}
public class Sum {
    public static void main(String[] args) {
        SumCalculator calc = Integer::sum;
        System.out.println(calc.sum(5, 7));
    }
}
```

Output:

12