Interface

1. Reverse CharSequence: Custom BackwardSequence

•Create a class BackwardSequence that implements java.lang.CharSequence.

•Internally store a String and implement all required methods: length(), charAt(), subSequence(), and toString().

•The sequence should be the reverse of the stored string (e.g., new BackwardSequence("hello") yields "olleh").

•Write a main() method to test each method.

ANSWER

```
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; }
}
```

2. Moveable Shapes Simulation

•Define an interface Movable with methods: moveUp(), moveDown(), moveLeft(), moveRight().
•Implement classes: oMovablePoint(x, y, xSpeed, ySpeed) implements Movable
oMovableCircle(radius, center: MovablePoint) oMovableRectangle(topLeft: MovablePoint, bottomRight: MovablePoint) (ensuring both points have same speed)

•Provide toString() to display positions.

•In main(), create a few objects and call move methods to simulate motion.

ANSWER

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

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

   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 + ")"; }
}
```

3. Contract Programming: Printer Switch •Declare an interface Printer with method void print(String document).

•Implement two classes: LaserPrinter and InkjetPrinter, each providing unique behavior.

•In the client code, declare Printer p;, switch implementations at runtime, and test printing.

ANSWER

```
interface Printer { void print(String document); }


class LaserPrinter implements Printer {
   public void print(String document) { System.out.println("Laser: " + document); }
}
class InkjetPrinter implements Printer {
   public void print(String document) { System.out.println("Inkjet: " + document); }
}
```

4. Extended Interface Hierarchy

- Define interface BaseVehicle with method void start().
- Define interface AdvancedVehicle that extends BaseVehicle, adding method void stop() and boolean refuel(int amount).
- Implement Car to satisfy both interfaces; include a constructor initializing fuel level.
- In Main, manipulate the object via both interface types.

ANSWER

```
interface BaseVehicle { void start(); }
interface AdvancedVehicle extends BaseVehicle {
   void stop();
   boolean refuel(int amount);
}


class Car implements AdvancedVehicle {
   private int fuel;
   Car(int fuel) { this.fuel = fuel; }
   public void start() { System.out.println("Car started"); }
   public void stop() { System.out.println("Car stopped"); }
```

```
    public boolean refuel(int amount) { fuel += amount; return true; }
}
```

5. Nested Interface for Callback Handling

- Create a class TimeServer which declares a public static nested interface named Client with void updateTime(LocalDateTime now).
- The server class should have method registerClient(Client client) and notifyClients() to pass current time.
- Implement at least two classes implementing Client, registering them, and simulate notifications

ANSWER

```
import java.time.LocalDateTime;
import java.util.*;

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);
    }
}
```

6. Default and Static Methods in Interfaces

•Declare interface Polygon with: odouble getArea() odefault method default double getPerimeter(int... sides) that computes sum of sides oa static helper static String shapeInfo() returning a description string

•Implement classes Rectangle and Triangle, providing appropriate getArea().

•In Main, call getPerimeter(...) and Polygon.shapeInfo().

ANSWER

```
interface Polygon {
   double getArea();
   default double getPerimeter(int... sides) {
      int sum = 0; for (int s : sides) sum += s; return sum;
   }
   static String shapeInfo() { return "Polygon: has area and perimeter"; }
}
```