Day5\_Assignment\_Answers:

1. package day\_5\_Assignments;

/\*Student with Grade Validation & Configuration

Ensure marks are always valid and immutable once set.

Create a Student class with private fields: name, rollNumber, and marks.

Use a constructor to initialize all values and enforce marks to be between 0 and 100; invalid values reset to 0.

Provide getter methods, but no setter for marks (immutable after object creation).

Add displayDetails() to print all fields.

In future versions, you might allow updating marks only via a special inputMarks(int newMarks)

method that has stricter logic (e.g. cannot reduce marks). Design accordingly.\*/

class Student{

private String name;

private int roll;

private double marks;

Student(String name,int roll,double marks){

this.name=name;

this.roll=roll;

if(marks>=0 && marks<=100) {

this.marks=marks;

}

else {

this.marks=0;

}

}

public String getName() {

return name;

}

public int getRoll() {

return roll;

}

public double getMarks() {

return marks;

}

public void setName(String name) {

this.name=name;

}

public void setRoll(int roll) {

this.roll=roll;

}

void display\_details() {

System.***out***.println("Name : "+name);

System.***out***.println("Roll : "+roll);

System.***out***.println("Marks : "+marks);

}

void update\_marks(double newMarks) {

if(this.marks<newMarks) {

System.***out***.println("Updated marks upon verification");

this.marks=newMarks;

}

else {

System.***err***.println("Not possible to update marks as newmarks are less than previous marks");

}

}

}

public class Student\_Details {

public static void main(String[] args) {

// **TODO** Auto-generated method stub

Student s=new Student("Swapna",501,98.5);

s.display\_details();

s.setName("Swapna");

s.setRoll(501);//we cant set marks

s.update\_marks(99);

}

}

Output:

Name : Swapna

Roll : 501

Marks : 98.5

Updated marks upon verification

2. public class Rectangle {

private double width;

private double height;

private static final double DEFAULT\_SIZE = 1.0;

public Rectangle(double width, double height) {

setWidth(width);

setHeight(height);

}

public void setWidth(double width) {

if (width > 0) {

this.width = width;

} else {

throw new IllegalArgumentException(

"Width must be positive. Given: " + width);

}

}

public void setHeight(double height) {

if (height > 0) {

this.height = height;

} else {

throw new IllegalArgumentException(

"Height must be positive. Given: " + height);

}

}

public double getWidth() {

return width;

}

public double getHeight() {

return height;

}

public double getArea() {

return width \* height;

}

public double getPerimeter() {

return 2 \* (width + height);

}

public void displayDetails() {

System.out.printf(

"Rectangle [width=%.2f, height=%.2f, area=%.2f, perimeter=%.2f]%n",

width, height, getArea(), getPerimeter());

}

public static void main(String[] args) {

try {

Rectangle r = new Rectangle(5.0, 3.0);

r.displayDetails();

Rectangle invalid = new Rectangle(-2.0, 0.0);

invalid.displayDetails(); // Won't reach here

} catch (IllegalArgumentException e) {

System.err.println("Error creating rectangle: " + e.getMessage());

}

}

}

Output:

Error creating rectangle: Width must be positive. Given: -2.0

3. 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.

public class BackwardSequence implements CharSequence {

private final String original;

public BackwardSequence(String original) {

this.original = original != null ? original : "";

}

@Override

public int length() {

return original.length();

}

@Override

public char charAt(int index) {

if (index < 0 || index >= length()) {

throw new IndexOutOfBoundsException(

"Index: " + index + ", Length: " + length());

}

return original.charAt(length() - 1 - index);

}

@Override

public CharSequence subSequence(int start, int end) {

if (start < 0 || end < start || end > length()) {

throw new IndexOutOfBoundsException(

"start " + start + ", end " + end + ", length " + length());

}

StringBuilder sb = new StringBuilder();

for (int i = start; i < end; i++) {

sb.append(charAt(i));

}

return sb.toString();

}

@Override

public String toString() {

return new StringBuilder(original).reverse().toString();

}

public static void main(String[] args) {

BackwardSequence bs = new BackwardSequence("hello");

System.out.println("Original: \"hello\"");

System.out.println("BackwardSequence.toString(): " + bs.toString());

System.out.println("length(): " + bs.length());

System.out.println("charAt(0): " + bs.charAt(0));

System.out.println("charAt(4): " + bs.charAt(4));

System.out.println("subSequence(1, 4): " + bs.subSequence(1, 4));

try {

System.out.println(bs.charAt(5));

} catch (IndexOutOfBoundsException e) {

System.out.println("Expected exception: " + e);

}

}

}

Output:

Original: "hello"

BackwardSequence.toString(): olleh

length(): 5

charAt(0): o

charAt(4): h

subSequence(1, 4): lle

Expected exception: java.lang.IndexOutOfBoundsException: Index: 5, Length: 5

4.

public interface Movable {

void moveUp();

void moveDown();

void moveLeft();

void moveRight();

}

public class MovablePoint implements Movable {

private 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;

}

@Override public void moveUp() { y -= ySpeed; }

@Override public void moveDown() { y += ySpeed; }

@Override public void moveLeft() { x -= xSpeed; }

@Override public void moveRight() { x += xSpeed; }

@Override

public String toString() {

return String.format("Point at (%d, %d), speed=(%d, %d)", x, y, xSpeed, ySpeed);

}

}

public class MovableCircle implements Movable {

private int radius;

private MovablePoint center;

public MovableCircle(int radius, MovablePoint center) {

this.radius = radius;

this.center = center;

}

@Override public void moveUp() { center.moveUp(); }

@Override public void moveDown() { center.moveDown(); }

@Override public void moveLeft() { center.moveLeft(); }

@Override public void moveRight() { center.moveRight(); }

@Override

public String toString() {

return String.format("Circle radius=%d, center=%s", radius, center);

}

public class MovableRectangle implements Movable {

private MovablePoint topLeft;

private MovablePoint bottomRight;

public MovableRectangle(MovablePoint p1, MovablePoint p2) {

if (p1 == null || p2 == null) {

throw new IllegalArgumentException("Points cannot be null");

}

if (p1.xSpeed != p2.xSpeed || p1.ySpeed != p2.ySpeed) {

throw new IllegalArgumentException("Both points must have equal speed");

}

topLeft = p1;

bottomRight = p2;

}

@Override public void moveUp() { topLeft.moveUp(); bottomRight.moveUp(); }

@Override public void moveDown() { topLeft.moveDown(); bottomRight.moveDown(); }

@Override public void moveLeft() { topLeft.moveLeft(); bottomRight.moveLeft(); }

@Override public void moveRight() { topLeft.moveRight(); bottomRight.moveRight(); }

@Override

public String toString() {

return String.format("Rectangle [topLeft=%s, bottomRight=%s]", topLeft, bottomRight);

}

}

public class TestMovableShapes {

public static void main(String[] args) {

MovablePoint p1 = new MovablePoint(0, 0, 2, 3);

MovablePoint p2 = new MovablePoint(10, 10, 2, 3);

MovableCircle circle = new MovableCircle(5, p1);

MovableRectangle rect = new MovableRectangle(p1, p2);

System.out.println("Initial States:");

System.out.println(circle);

System.out.println(rect);

circle.moveRight();

circle.moveDown();

System.out.println("\nAfter moving circle:");

System.out.println(circle);

rect.moveLeft();

rect.moveUp();

System.out.println("\nAfter moving rectangle:");

System.out.println(rect);

circle.moveUp();

circle.moveLeft();

rect.moveDown();

rect.moveRight();

System.out.println("\nAfter additional moves:");

System.out.println(circle);

System.out.println(rect);

}

}

Output:

Initial States:

Circle radius=5, center=Point at (0, 0), speed=(2, 3)

Rectangle [topLeft=Point at (0, 0), speed=(2, 3), bottomRight=Point at (10, 10), speed=(2, 3)]

After moving circle:

Circle radius=5, center=Point at (2, 3), speed=(2, 3)

After moving rectangle:

Rectangle [topLeft=Point at ( -2, -3), bottomRight=Point at ( 8, 7)]

After additional moves:

Circle radius=5, center=Point at (0, 0)

Rectangle [topLeft=Point at ( 0, 0), bottomRight=Point at ( 10, 10)]

Output:

Initial States:

Circle radius=5, center=Point at (0, 0), speed=(2, 3)

Rectangle [topLeft=Point at (0, 0), speed=(2, 3), bottomRight=Point at (10, 10), speed=(2, 3)]

After moving circle:

Circle radius=5, center=Point at (2, 3), speed=(2, 3)

After moving rectangle:

Rectangle [topLeft=Point at ( -2, -3), bottomRight=Point at ( 8, 7)]

After additional moves:

Circle radius=5, center=Point at (0, 0)

Rectangle [topLeft=Point at ( 0, 0), bottomRight=Point at ( 10, 10)]

5. 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.

a) public class PrinterDemo {

public static void main(String[] args) {

Printer p;

p = new LaserPrinter();

p.print("Document printed by LaserPrinter.");

p = new InkjetPrinter();

p.print("Document printed by InkjetPrinter.");

}

interface Printer {

void print(String document);

}

static class LaserPrinter implements Printer {

@Override

public void print(String document) {

System.out.println("LaserPrinter is printing...");

System.out.println(document);

}

}

static class InkjetPrinter implements Printer {

@Override

public void print(String document) {

System.out.println("InkjetPrinter is spraying ink...");

System.out.println(document);

}

}

}

Output:

LaserPrinter is printing...

Document printed by LaserPrinter.

InkjetPrinter is spraying ink...

Document printed by InkjetPrinter.

6.Lambda Expressions:

package day\_5\_Assignments;

interface SumCalculator {

int sum(int a, int b);

}

public class LambdaSum {

public static void main(String[] args) {

SumCalculator calculator = (a, b) -> a + b;

int sum1 = calculator.sum(7, 6);

System.***out***.println("Sum of 7 and 6: " + sum1);

int sum2 = calculator.sum(15, -35);

System.***out***.println("Sum of 15 and -35: " + sum2);

}

}

Output:

Sum of 7 and 6: 13

Sum of 15 and -35: -20

7.Even or odd

package day\_5\_Assignments;

public class LambdaEven {

public static void main(String[] args) {

CheckEvenOdd check = (num) -> num % 2 == 0;

int number = 7;

System.***out***.println(number + " is " + (check.isEven(number) ? "even" : "odd"));

}

*@FunctionalInterface*

interface CheckEvenOdd {

boolean isEven(int num);

}

}

Output:7 is odd