Encapsulation

1.package encapsulation;

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; // default for invalid marks

}

}

public String getName() { return name; }

public int getRollNumber() { return rollNumber; }

public int getMarks() { return marks; }

public void inputMarks(int newMarks) {

if (newMarks >= 0 && newMarks <= 100 && newMarks >= this.marks) {

this.marks = newMarks; // only allow increase

} else {

System.out.println("Invalid or lower marks, update rejected!");

}

}

public void displayDetails() {

System.out.println("Student: " + name + ", Roll: " + rollNumber + ", Marks: " + marks);

}

}

2.package encapsulation;

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; // default

}

public void setHeight(double height) {

if (height > 0) this.height = height;

else this.height = 1; // default

}

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.println("Rectangle [Width=" + width + ", Height=" + height +

", Area=" + getArea() + ", Perimeter=" + getPerimeter() + "]");

}

}

3.package encapsulation;

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);

} else {

System.out.println("Invalid deposit amount.");

}

}

public boolean withdraw(double amount) {

if (amount > 0 && amount <= balance) {

balance -= amount;

transactionHistory.add("Withdrawn: " + amount);

return true;

}

System.out.println("Insufficient funds or invalid withdrawal.");

return false;

}

public double getBalance() {

return balance;

}

public String getLastTransaction() {

if (transactionHistory.isEmpty()) return "No transactions yet.";

return transactionHistory.get(transactionHistory.size() - 1);

}

@Override

public String toString() {

String maskedAcc = "\*\*\*\*" + accountNumber.substring(accountNumber.length() - 4);

return "Account Holder: " + accountHolder + ", Account: " + maskedAcc + ", Balance: " + balance;

}

}

4.package encapsulation

class Locker {

private String lockerId;

private boolean isLocked;

private String passcode;

public Locker(String lockerId, String passcode) {

this.lockerId = lockerId;

this.passcode = passcode;

this.isLocked = true;

}

private class SecurityManager {

private boolean verify(String code) {

return passcode.equals(code);

}

}

public void lock() {

isLocked = true;

System.out.println("Locker " + lockerId + " is now locked.");

}

public boolean unlock(String code) {

SecurityManager sm = new SecurityManager();

if (sm.verify(code)) {

isLocked = false;

System.out.println("Locker " + lockerId + " unlocked successfully.");

return true;

}

System.out.println("Incorrect passcode. Locker remains locked.");

return false;

}

public boolean isLocked() {

return isLocked;

}

}

5.package encapsulation;

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; }

@Override

public String toString() {

return "Product [Name=" + name + ", Code=" + code +

", Price=" + price + ", Category=" + 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) throw new IllegalArgumentException("Price cannot be negative.");

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);

}

}

}

Interface

1.package interface;

class BackwardSequence implements CharSequence {

private final String original;

private final String reversed;

public BackwardSequence(String str) {

this.original = str;

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

}

@Override

public int length() {

return reversed.length();

}

@Override

public char charAt(int index) {

return reversed.charAt(index);

}

@Override

public CharSequence subSequence(int start, int end) {

return reversed.subSequence(start, end);

}

@Override

public String toString() {

return reversed;

}

public static void main(String[] args) {

BackwardSequence bs = new BackwardSequence("hello");

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

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

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

System.out.println("CharAt(1): " + bs.charAt(1));

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

}

}

2.package interface;

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;

}

@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 "Point(" + x + "," + y + ")";

}

}

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 "Circle(radius=" + radius + ", center=" + center + ")";

}

}

class MovableRectangle implements Movable {

private MovablePoint topLeft;

private MovablePoint bottomRight;

public MovableRectangle(MovablePoint topLeft, MovablePoint bottomRight) {

if (topLeft.xSpeed != bottomRight.xSpeed || topLeft.ySpeed != bottomRight.ySpeed) {

throw new IllegalArgumentException("Speeds must match for rectangle corners!");

}

this.topLeft = topLeft;

this.bottomRight = bottomRight;

}

@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 "Rectangle(TopLeft=" + topLeft + ", BottomRight=" + bottomRight + ")";

}

}

class ShapeTest {

public static void main(String[] args) {

MovablePoint p = new MovablePoint(0, 0, 2, 2);

MovableCircle c = new MovableCircle(5, p);

MovableRectangle r = new MovableRectangle(new MovablePoint(0, 0, 1, 1),

new MovablePoint(2, 2, 1, 1));

System.out.println(c);

c.moveRight();

System.out.println("After moveRight: " + c);

System.out.println(r);

r.moveUp();

System.out.println("After moveUp: " + r);

}

}

Lamda Expression

1.package lamda;

interface SumCalculator {

int sum(int a, int b);

}

public class LambdaExamples {

public static void main(String[] args) {

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

System.out.println("Sum: " + add.sum(10, 20)); // Output: 30

}

}

2. import java.util.function.Predicate;

public class StringCheck {

public static void main(String[] args) {

Predicate<String> isEmpty = s -> s.isEmpty();

System.out.println(isEmpty.test(""));

System.out.println(isEmpty.test("Hello"));

}

}

3. package lamda;

import java.util.function.Predicate;

public class StringCheck {

public static void main(String[] args) {

Predicate<String> isEmpty = s -> s.isEmpty();

System.out.println(isEmpty.test("")); // true

System.out.println(isEmpty.test("Hello")); // false

}

}

4. package lamda;

import java.util.\*;

import java.util.stream.Collectors;

public class EvenOddFilter {

public static void main(String[] args) {

List<Integer> numbers = Arrays.asList(1,2,3,4,5,6,7,8,9,10);

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("Even Numbers: " + evens);

System.out.println("Odd Numbers: " + odds);

}

}

5.package lamda;

import java.util.\*;

import java.util.stream.Collectors;

public class StringCase {

public static void main(String[] args) {

List<String> words = Arrays.asList("java", "lambda", "expression");

List<String> upper = words.stream()

.map(String::toUpperCase)

.collect(Collectors.toList());

List<String> lower = words.stream()

.map(String::toLowerCase)

.collect(Collectors.toList());

System.out.println("Uppercase: " + upper);

System.out.println("Lowercase: " + lower);

}

}

6.package lamda;

import java.util.\*;

public class SortStrings {

public static void main(String[] args) {

List<String> words = Arrays.asList("apple", "banana", "kiwi", "pear");

words.sort((s1, s2) -> Integer.compare(s1.length(), s2.length()));

System.out.println("Sorted by length: " + words);

words.sort(String::compareTo);

System.out.println("Sorted alphabetically: " + words);

}

}

7.package lamda;

import java.util.\*;

import java.util.stream.\*;

public class AggregateOps {

public static void main(String[] args) {

double[] arr = {10.5, 20.2, 5.8, 40.0};

double sum = Arrays.stream(arr).sum();

double max = Arrays.stream(arr).max().getAsDouble();

double avg = Arrays.stream(arr).average().getAsDouble();

System.out.println("Sum: " + sum);

System.out.println("Max: " + max);

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

}

}

8.package lamda;

import java.util.function.BinaryOperator;

public class MaxMinLambda {

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: " + max.apply(10, 20));

System.out.println("Min: " + min.apply(10, 20));

}

}

9.package lamda;

import java.util.function.IntFunction;

public class FactorialLambda {

public static void main(String[] args) {

IntFunction<Integer> factorial = n -> {

int result = 1;

for (int i = 1; i <= n; i++) result \*= i;

return result;

};

System.out.println("Factorial of 5: " + factorial.apply(5)); // Output: 120

}

}