1. Behavioural Patterns :

interface Observer {

void update(float temperature);

}

class WeatherStation {

private List<Observer> observers = new ArrayList<>();

private float temperature;

public void addObserver(Observer observer) {

observers.add(observer);

}

public void removeObserver(Observer observer) {

observers.remove(observer);

}

public void setTemperature(float temperature) {

this.temperature = temperature;

notifyObservers();

}

private void notifyObservers() {

for (Observer observer : observers) {

observer.update(temperature);

}

}

}

class PhoneDisplay implements Observer {

@Override

public void update(float temperature) {

System.out.println("Phone Display: Temperature updated to " + temperature);

}

}

WeatherStation weatherStation = new WeatherStation();

Observer phoneDisplay = new PhoneDisplay();

weatherStation.addObserver(phoneDisplay);

weatherStation.setTemperature(25.0f);

// Java Example - Strategy Pattern

interface PaymentStrategy {

void pay(int amount);

}

class CreditCardPayment implements PaymentStrategy {

@Override

public void pay(int amount) {

System.out.println("Paid $" + amount + " using Credit Card.");

}

}

class PayPalPayment implements PaymentStrategy {

@Override

public void pay(int amount) {

System.out.println("Paid $" + amount + " using PayPal.");

}

}

class PaymentContext {

private PaymentStrategy strategy;

public void setPaymentStrategy(PaymentStrategy strategy) {

this.strategy = strategy;

}

public void pay(int amount) {

strategy.pay(amount);

}

}

// Usage:

PaymentContext paymentContext = new PaymentContext();

paymentContext.setPaymentStrategy(new CreditCardPayment());

paymentContext.pay(100);

paymentContext.setPaymentStrategy(new PayPalPayment());

paymentContext.pay(200);

// Java Example - Singleton Pattern

class ConfigManager {

private static ConfigManager instance;

private ConfigManager() {}

public static synchronized ConfigManager getInstance() {

if (instance == null) {

instance = new ConfigManager();

}

return instance;

}

public void loadConfig() {

System.out.println("Config Loaded.");

}

}

// Usage:

ConfigManager configManager = ConfigManager.getInstance();

configManager.loadConfig();

// Java Example - Factory Pattern

interface Shape {

void draw();

}

class Circle implements Shape {

@Override

public void draw() {

System.out.println("Drawing a Circle");

}

}

class Square implements Shape {

@Override

public void draw() {

System.out.println("Drawing a Square");

}

}

class ShapeFactory {

public Shape getShape(String shapeType) {

if (shapeType == null) {

return null;

}

if (shapeType.equalsIgnoreCase("CIRCLE")) {

return new Circle();

} else if (shapeType.equalsIgnoreCase("SQUARE")) {

return new Square();

}

return null;

}

}

// Usage:

ShapeFactory shapeFactory = new ShapeFactory();

Shape shape1 = shapeFactory.getShape("CIRCLE");

shape1.draw();



// Java Example - Adapter Pattern

interface MediaPlayer {

void play(String audioType, String fileName);

}

class AudioPlayer implements MediaPlayer {

MediaAdapter mediaAdapter;

@Override

public void play(String audioType, String fileName) {

if (audioType.equalsIgnoreCase("mp3")) {

System.out.println("Playing mp3 file: " + fileName);

} else if (audioType.equalsIgnoreCase("vlc") || audioType.equalsIgnoreCase("mp4")) {

mediaAdapter = new MediaAdapter(audioType);

mediaAdapter.play(audioType, fileName);

}

}

}

class MediaAdapter implements MediaPlayer {

AdvancedMediaPlayer advancedMediaPlayer;

public MediaAdapter(String audioType) {

if (audioType.equalsIgnoreCase("vlc")) {

advancedMediaPlayer = new VlcPlayer();

} else if (audioType.equalsIgnoreCase("mp4")) {

advancedMediaPlayer = new Mp4Player();

}

}

@Override

public void play(String audioType, String fileName) {

if (audioType.equalsIgnoreCase("vlc")) {

advancedMediaPlayer.playVlc(fileName);

} else if (audioType.equalsIgnoreCase("mp4")) {

advancedMediaPlayer.playMp4(fileName);

}

}

}

interface AdvancedMediaPlayer {

void playVlc(String fileName);

void playMp4(String fileName);

}

class VlcPlayer implements AdvancedMediaPlayer {

@Override

public void playVlc(String fileName) {

System.out.println("Playing vlc file: " + fileName);

}

@Override

public void playMp4(String fileName) {}

}

class Mp4Player implements AdvancedMediaPlayer {

@Override

public void playVlc(String fileName) {}

@Override

public void playMp4(String fileName) {

System.out.println("Playing mp4 file: " + fileName);

}

}

// Usage:

MediaPlayer audioPlayer = new AudioPlayer();

audioPlayer.play("mp3", "song.mp3");

audioPlayer.play("mp4", "movie.mp4");

// Java Example - Composite Pattern

interface FileComponent {

void showDetails();

}

class FileLeaf implements FileComponent {

private String name;

public FileLeaf(String name) {

this.name = name;

}

@Override

public void showDetails() {

System.out.println("File: " + name);

}

}

class DirectoryComposite implements FileComponent {

private String name;

private List<FileComponent> components = new ArrayList<>();

public DirectoryComposite(String name) {

this.name = name;

}

public void addComponent(FileComponent component) {

components.add(component);

}

@Override

public void showDetails() {

System.out.println("Directory: " + name);

for (FileComponent component : components) {

component.showDetails();

}

}

}

// Usage:

DirectoryComposite mainDirectory = new DirectoryComposite("Root");

FileLeaf file1 = new FileLeaf("file1.txt");

FileLeaf file2 = new FileLeaf("file2.txt");

DirectoryComposite subDirectory = new DirectoryComposite("Subfolder");

subDirectory.addComponent(file1);

mainDirectory.addComponent(subDirectory);

mainDirectory.addComponent(file2);

mainDirectory.showDetails();

MINI PROJECT:

using System;

using System.Collections.Generic;

public enum Direction {

North, South, East, West

}

public class Rover {

public int X { get; private set; }

public int Y { get; private set; }

public Direction Facing { get; private set; }

public Rover(int x, int y, Direction facing) {

X = x;

Y = y;

Facing = facing;

}

public void MoveForward() {

switch (Facing) {

case Direction.North:

Y++;

break;

case Direction.South:

Y--;

break;

case Direction.East:

X++;

break;

case Direction.West:

X--;

break;

}

Console.WriteLine($"Rover moved to position ({X}, {Y}), facing {Facing}");

}

public interface ICommand {

void Execute();

}

public class MoveCommand : ICommand {

private Rover \_rover;

public MoveCommand(Rover rover) {

\_rover = rover;

}

public void Execute() {

\_rover.MoveForward();

}

}

public class TurnLeftCommand : ICommand {

private Rover \_rover;

public TurnLeftCommand(Rover rover) {

\_rover = rover;

}

public void Execute() {

\_rover.TurnLeft();

}

}

public class TurnRightCommand : ICommand {

private Rover \_rover;

public TurnRightCommand(Rover rover) {

\_rover = rover;

}

public void Execute() {

\_rover.TurnRight();

}

}

public void TurnLeft() {

Facing = Facing switch {

Direction.North => Direction.West,

Direction.West => Direction.South,

Direction.South => Direction.East,

Direction.East => Direction.North,

\_ => Facing

};

Console.WriteLine($"Rover turned left. Now facing {Facing}");

}

public void TurnRight() {

Facing = Facing switch {

Direction.North => Direction.East,

Direction.East => Direction.South,

Direction.South => Direction.West,

Direction.West => Direction.North,

\_ => Facing

};

Console.WriteLine($"Rover turned right. Now facing {Facing}");

}

}

public class CommandInvoker {

private readonly List<ICommand> \_commands = new List<ICommand>();

public void AddCommand(ICommand command) {

\_commands.Add(command);

}

public void ExecuteCommands() {

foreach (var command in \_commands) {

command.Execute();

}

\_commands.Clear();

}

}

public class Grid {

private int Width;

private int Height;

private HashSet<(int, int)> Obstacles = new HashSet<(int, int)>();

public Grid(int width, int height) {

Width = width;

Height = height;

}

public void AddObstacle(int x, int y) {

Obstacles.Add((x, y));

Console.WriteLine($"Obstacle added at ({x}, {y})");

}

public bool IsObstacle(int x, int y) {

return Obstacles.Contains((x, y));

}

public bool IsWithinBounds(int x, int y) {

return x >= 0 && x < Width && y >= 0 && y < Height;

}

}

public class Program {

public static void Main(string[] args) {

// Initialize the grid and rover

Grid grid = new Grid(10, 10);

grid.AddObstacle(2, 2);

grid.AddObstacle(3, 5);

Rover rover = new Rover(0, 0, Direction.North);

CommandInvoker invoker = new CommandInvoker();

// Define some commands

invoker.AddCommand(new MoveCommand(rover));

invoker.AddCommand(new TurnRightCommand(rover));

invoker.AddCommand(new MoveCommand(rover));

invoker.AddCommand(new TurnLeftCommand(rover));

invoker.AddCommand(new MoveCommand(rover));

// Execute commands

invoker.ExecuteCommands();

// After moving, check if there's an obstacle

if (grid.IsObstacle(rover.X, rover.Y)) {

Console.WriteLine($"Obstacle detected at ({rover.X}, {rover.Y})! Rover cannot move further.");

} else if (!grid.IsWithinBounds(rover.X, rover.Y)) {

Console.WriteLine($"Rover is out of bounds at ({rover.X}, {rover.Y})! Returning to last valid position.");

} else {

Console.WriteLine($"Rover final position: ({rover.X}, {rover.Y}), facing {rover.Facing}");

}

}

}

Output:

Rover moved to position (0, 1), facing North

Rover turned right. Now facing East

Rover moved to position (1, 1), facing East

Rover turned left. Now facing North

Rover moved to position (1, 2), facing North

Rover final position: (1, 2), facing North