**Two use cases to demonstrate two behavioural design pattern.**

**1.Sorting Stratergies**

import java.util.Arrays;

interface SortStrategy {

void sort(int[] array, int size);

}

class BubbleSort implements SortStrategy {

@Override

public void sort(int[] array, int size) {

for (int i = 0; i < size - 1; i++) {

for (int j = 0; j < size - i - 1; j++) {

if (array[j] > array[j + 1]) {

int temp = array[j];

array[j] = array[j + 1];

array[j + 1] = temp;

}

}

}

}

}

class QuickSort {

public void sort(int[] array, int low, int high) {

if (low < high) {

int pivot = array[high];

int i = (low - 1);

for (int j = low; j < high; j++) {

if (array[j] < pivot) {

i++;

int temp = array[i];

array[i] = array[j];

array[j] = temp;

}

}

int temp = array[i + 1];

array[i + 1] = array[high];

array[high] = temp;

int pi = i + 1;

sort(array, low, pi - 1);

sort(array, pi + 1, high);

}

}

}

public class Main {

public static void sortArray(SortStrategy strategy, int[] array, int size) {

strategy.sort(array, size);

}

public static void main(String[] args) {

int[] array = {64, 25, 12, 22, 11};

int size = array.length;

System.out.println("Original array: " + Arrays.toString(array));

System.out.println("Sorting using Bubble Sort:");

sortArray(new BubbleSort(), array, size);

System.out.println(Arrays.toString(array));

// Reset array

int[] array2 = {64, 25, 12, 22, 11};

size = array2.length;

System.out.println("Sorting using Quick Sort:");

QuickSort quickSort = new QuickSort();

quickSort.sort(array2, 0, size - 1);

System.out.println(Arrays.toString(array2));

}

}

**2. Event Notification System**

import java.util.ArrayList;

import java.util.List;

interface Observer {

void update(String message);

}

class Subject {

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

public void addObserver(Observer observer) {

if (observers.size() < 10) {

observers.add(observer);

}

}

public void notifyObservers(String message) {

for (Observer observer : observers) {

observer.update(message);

}

}

}

class ExampleObserver implements Observer {

@Override

public void update(String message) {

System.out.println("Observer notified with message: " + message);

}

}

public class Main {

public static void main(String[] args) {

Subject subject = new Subject();

Observer observer1 = new ExampleObserver();

Observer observer2 = new ExampleObserver();

subject.addObserver(observer1);

subject.addObserver(observer2);

System.out.println("Notifying observers...");

subject.notifyObservers("Event occurred!");

}

}