3.A Proxy

interface SensitiveObject {

String getSecretKey();

}

class RealSensitiveObject implements SensitiveObject {

private String secretKey;

public RealSensitiveObject(String secretKey) {

this.secretKey = secretKey;

}

@Override

public String getSecretKey() {

return secretKey;

}

}

class SensitiveObjectProxy implements SensitiveObject {

private RealSensitiveObject realObject;

private String password;

public SensitiveObjectProxy(String secretKey, String password) {

this.realObject = new RealSensitiveObject(secretKey);

this.password = password;

}

@Override

public String getSecretKey() {

if (authenticate()) {

return realObject.getSecretKey();

} else {

throw new SecurityException("Access denied: Incorrect password");

}

}

private boolean authenticate() {

return "correctPassword".equals(password);

}

}

public class ProxyPatternExample {

public static void main(String[] args) {

SensitiveObject proxy = new SensitiveObjectProxy("superSecretKey123", "correctPassword");

try {

String secretKey = proxy.getSecretKey();

System.out.println("Secret Key: " + secretKey);

} catch (SecurityException e) {

System.out.println("Error: " + e.getMessage());

}

try {

SensitiveObject proxyWrongPassword = new SensitiveObjectProxy("superSecretKey123", "wrongPassword");

String secretKey = proxyWrongPassword.getSecretKey();

System.out.println("Secret Key: " + secretKey);

} catch (SecurityException e) {

System.out.println("Error: " + e.getMessage());

}

}

}

4.A Strategy

interface SortingStrategy {

void sort(int[] array);

}

class BubbleSort implements SortingStrategy {

@Override

public void sort(int[] array) {

int n = array.length;

boolean swapped;

do {

swapped = false;

for (int i = 1; i < n; i++) {

if (array[i - 1] > array[i]) {

int temp = array[i - 1];

array[i - 1] = array[i];

array[i] = temp;

swapped = true;

}

}

n--;

} while (swapped);

}

}

class QuickSort implements SortingStrategy {

@Override

public void sort(int[] array) {

quickSort(array, 0, array.length - 1);

}

private void quickSort(int[] array, int low, int high) {

if (low < high) {

int pi = partition(array, low, high);

quickSort(array, low, pi - 1);

quickSort(array, pi + 1, high);

}

}

private int partition(int[] array, int low, int high) {

int pivot = array[high];

int i = low - 1;

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

if (array[j] < pivot) {

i++;

// Swap elements

int temp = array[i];

array[i] = array[j];

array[j] = temp;

}

}

int temp = array[i + 1];

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

array[high] = temp;

return i + 1;

}

}

class Context {

private SortingStrategy strategy;

public void setStrategy(SortingStrategy strategy) {

this.strategy = strategy;

}

public void sortArray(int[] array) {

strategy.sort(array);

}

}

public class StrategyPatternExample {

public static void main(String[] args) {

int[] numbers = {5, 2, 8, 1, 6};

Context context = new Context();

context.setStrategy(new BubbleSort());

context.sortArray(numbers.clone());

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

printArray(numbers);

context.setStrategy(new QuickSort());

context.sortArray(numbers.clone());

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

printArray(numbers);

}

private static void printArray(int[] array) {

for (int num : array) {

System.out.print(num + " ");

}

System.out.println();

}

}