Beadandó: Tervezési Minták az Objektumorientált Programozásban

Bevezetés

Az objektumorientált programozás (OO) paradigmája számos előnnyel jár a modern szoftvertervezés számára, és a tervezési minták egyfajta iránytűt nyújtanak a hatékony és rugalmas kód kialakításához. Ebben a beadandóban részletesen vizsgáljuk meg az MVC (Model-View-Controller) mintát, és néhány másik kulcsfontosságú tervezési mintát, konkrét példákkal szemléltetve az alkalmazásukat a való életből vett helyzetekben.

1. MVC Minta: Model-View-Controller

Az MVC minta segítségével egy alkalmazás felépítése és működése különféle komponensekre oszlik.

- Model (Modell): Képzeljük el egy online áruház alkalmazását, ahol a Model a termékek kezeléséért és az adatbázis interakciójáért felel. Például a Model gondoskodik a termékek listázásáról, hozzáadásáról és eltávolításáról.

Példa: import java.util.ArrayList;

import java.util.List;

// Termék osztály a Model részeként

class Product {

private String name;

private double price;

public Product(String name, double price) {

this.name = name;

this.price = price;

}

public String getName() {

return name;

}

public double getPrice() {

return price;

}

}

// Áruház osztály a Model részeként

class Store {

private List<Product> products = new ArrayList<>();

public void addProduct(Product product) {

products.add(product);

}

public List<Product> getProducts() {

return new ArrayList<>(products); // Védett másolatot ad vissza a termékekről

}

}

- View (Nézet): A View a felhasználói felületért felelős komponens, és a felhasználó által látott weboldalt vagy alkalmazást alkotja. Az áruház esetében a View a termékek megjelenítését és a kosár tartalmának kijelzését vállalhatja.

Példa:

// Az Áruház Nézet osztály

class StoreView {

public void displayProducts(List<Product> products) {

System.out.println("Available products:");

for (Product product : products) {

System.out.println(product.getName() + " - $" + product.getPrice());

}

System.out.println();

}

}

// A Pénztár Nézet osztály

class CheckoutView {

public void displayTotal(double total) {

System.out.println("Total: $" + total);

}

}

- Controller (Vezérlő): A Controller irányítja a felhasználói interakciókat és az eseményeket. Ha a felhasználó hozzáad egy terméket a kosarához, a Controller fogja kezelni ezt az eseményt és értesíti a Modelt és a Viewt.

Példa:

// Az Áruház Vezérlő osztály

class StoreController {

private Store model;

private StoreView storeView;

private CheckoutView checkoutView;

public StoreController(Store model, StoreView storeView, CheckoutView checkoutView) {

this.model = model;

this.storeView = storeView;

this.checkoutView = checkoutView;

}

public void displayProducts() {

List<Product> products = model.getProducts();

storeView.displayProducts(products);

}

public void checkout() {

List<Product> products = model.getProducts();

double total = calculateTotal(products);

checkoutView.displayTotal(total);

}

private double calculateTotal(List<Product> products) {

double total = 0;

for (Product product : products) {

total += product.getPrice();

}

return total;

}

}

2. További Tervezési Minták

Most tekintsünk át néhány másik kulcsfontosságú tervezési mintát és lássunk példákat a való életből.

- Singleton Minta: Vegyük példának egy naplózó rendszert, ahol csak egyetlen naplózó objektumra van szükség. A Singleton minta biztosítja, hogy az alkalmazás egész terjedelmében egyetlen naplózó példány legyen, például az alkalmazás konfigurációjának tartalmazására.

Példa a mintára:

public class Logger {

private static Logger instance;

private Logger() {

// Privát konstruktor

}

public static Logger getInstance() {

if (instance == null) {

instance = new Logger();

}

return instance;

}

public void logMessage(String message) {

System.out.println("Log: " + message);

}

}

- Observer Minta: Képzeljük el egy hírportált, ahol a felhasználók értesítést kapnak az új cikkekről. Az Observer minta lehetővé teszi, hogy a felhasználók figyeljék a hírportált, és értesítve legyenek a frissítésekről.

Példa a mintára:

public interface Observer {

void update(String news);

}

public class User implements Observer {

private String name;

public User(String name) {

this.name = name;

}

@Override

public void update(String news) {

System.out.println(name + " received news: " + news);

}

}

public class NewsPortal {

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

public void addObserver(Observer observer) {

observers.add(observer);

}

public void publishNews(String news) {

for (Observer observer : observers) {

observer.update(news);

}

}

}

- Strategy Minta: Tekintsük át egy fizetési rendszer esetét, ahol többféle fizetési mód közül választhat a felhasználó. A Strategy minta lehetővé teszi, hogy a különböző fizetési módokat egyszerűen cseréljük ki.

Példa a mintára:

public interface PaymentStrategy {

void pay(int amount);

}

public class CreditCardPayment implements PaymentStrategy {

@Override

public void pay(int amount) {

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

}

}

public class PayPalPayment implements PaymentStrategy {

@Override

public void pay(int amount) {

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

}

}

public class ShoppingCart {

private PaymentStrategy paymentStrategy;

public void setPaymentStrategy(PaymentStrategy paymentStrategy) {

this.paymentStrategy = paymentStrategy;

}

public void checkout(int amount) {

paymentStrategy.pay(amount);

}

}

Záró Megjegyzések

Ez a beadandó részletesebben bemutatja az MVC minta és néhány másik kulcsfontosságú tervezési minta alkalmazását a való életből vett példákon keresztül. A tervezési minták nemcsak kódstruktúrális elveket képviselnek, hanem hatékony eszközök a szoftvertervezési problémák kezelésére és a kód karbantarthatóságának javítására.

Referenciák

- Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1994). Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley.

- Fowler, M. (2003). Patterns of Enterprise Application Architecture. Addison-Wesley.