Tervezési minták egy OO programozási nyelvben. MVC, mint modell-nézet-vezérlő minta és néhány másik tervezési minta

A szoftverfejlesztésnél a tervezési minták fontos eszközök, amelyek segítenek a fejlesztőknek strukturált és hatékony kódot írni. Ezek a minták bevált megoldásokat kínálnak gyakran előforduló problémákra, ezzel növelve a kód újrafelhasználhatóságát és karbantarthatóságát. A tervezési minták alapelvei a moduláris tervezést, az egyszerűségre törekvést és a rugalmasságot hangsúlyozzák. Három népszerű tervezési minta a Model-View-Controller (MVC), a Singleton és az Observer, amelyek mindegyike sajátos kihívásokra nyújt hatékony megoldást.

MVC

Az MVC egy architekturális tervezési minta, ami különálló komponensekre osztja a szoftveralkalmazást: modell, nézet és vezérlő. Ez a szétválasztás elősegíti a kód tisztaságát és karbantarthatóságát.

A modell felelős az adatok kezeléséért és az üzleti logika megvalósításáért. Például egy webshopban a modell kezeli a termékek, ügyfelek és rendelések adatait, valamint biztosítja az üzleti szabályok betartását.

Példa:

Egy webshopban a modell felel a termékek, vásárlók és rendelések kezeléséért.

Részletkód a modellről a Java-ban:

```
public class Product {
    private String name;
    private double price;

    public Product(String name, double price) {
        this.name = name;
        this.price = price;
    }
}
```

A nézet a felhasználói interfészért felelős, ahol az adatok vizuálisan megjelennek a felhasználó számára. Ez a komponens kizárólag az adatmegjelenítésre fókuszál, az üzleti logikától mentesen. HTML és CSS például gyakran használatosak a nézetek megvalósításában webes alkalmazásokban.

Részletkód a nézetről a CSS-ban:

```
<html>
    <body>
        <h1>Product List</h1>
        <ul>
            <!-- Dinamikus terméklista -->
        </ul>
    </body>
</html>
```

A vezérlő kezeli a felhasználói bemeneteket, kommunikál a modellel, és kiválasztja a megfelelő nézetet az adatok megjelenítésére. Például, amikor egy felhasználó bejelentkezik egy alkalmazásba, a vezérlő validálja a bevitt adatokat, és megfelelően frissíti a nézetet az eredmény függvényében.

Ez egy példa a vezérlőről. A felhasználó kattint egy "Vásárlás" gombra, a vezérlő hozzáadja a kosárhoz a terméket.

Részletkód a vezérlőről a Java-ban:

```
public class ProductController {
    private Product model;

    public ProductController(Product model) {
        this.model = model;
    }

    public void updateView(ProductView view) {
        view.displayProduct(model.getName(), model.getPrice());
    }
}
```

Az MVC minta segít a kód strukturálásában és egyszerűbbé teszi a különböző komponensek fejlesztését és karbantartását.

Az MVC minta azért jó, mert könnyen szét lehet választani a kódot és emiatt könnyen tesztelhető és újrafelhasználható, de kis projekteknél felesleges, mert csak bonyolultabbá teheti az alkalmazás felépítését.

Singleton

A Singleton minta célja, hogy egy osztályból csak egyetlen példány létezzen az alkalmazás futása során, és ez az egy példány globálisan hozzáférhető legyen.

A Singleton osztály privát konstruktort használ, hogy megakadályozza több példány létrejöttét. Az egyedi példányhoz egy statikus metóduson keresztül férhetünk hozzá, amely ellenőrzi, hogy létezik-e már a példány, és ha nem, akkor létrehozza.

Részletkód a Singletonról ami megmutat egy naplózási rendszert, ahol az összes naplóbejegyzést egy központi objektum kezeli.

```
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 log(String message) {
        System.out.println(message);
    }
}

// Felhasználás:
Logger logger = Logger.getInstance();
logger.log("Ez egy naplóüzenet.");
```

A Singleton mintát gyakran használják konfigurációs objektumok, naplózási rendszerek és kapcsolatkezelők esetén. Ez a minta hatékonyabb erőforráskezelést tesz lehetővé és biztosítja, hogy az alkalmazásban csak egyetlen példány létezzen egy adott osztályból, de ez a minta megnehezíti a tesztelést, mert a globális állapot befolyásolhatja a tesztek eredményét.

Observer

Az Observer minta lehetővé teszi, hogy egy objektum (alany) értesítse a megfigyelő objektumokat az állapotváltozásairól. Ez a minta különösen hasznos olyan alkalmazásokban, ahol az eseményekre adott válaszok fontosak.

A subject (alany) nyilvántartja a megfigyelők listáját, és értesíti őket, amikor az állapota megváltozik.

Az Observer (megfigyelő) az a komponens, amely figyeli az alany állapotát és reagál a változásokra.

Ez egy példa a megfigyelőről. Egy valós idejű árfolyam-megjelenítő frissíti a grafikonokat, amikor az árfolyam változik.

Részletkód az megfigyelőről a Java-ban:

```
import java.util.ArrayList;
import java.util.List;

// Alany (Subject)
class Stock {
    private List<Observer> observers = new ArrayList<>();
    private double price;

    public void addObserver(Observer observer) {
        observers.add(observer);
    }

    public void setPrice(double price) {
        this.price = price;
        notifyObservers();
    }

    private void notifyObservers() {
        for (Observer observer : observers) {
            observer.update(price);
        }
    }
}

// Megfigyelő (Observer)
interface Observer {
    void update(double price);
}

// Konkrét megfigyelő
class StockDisplay implements Observer {
    public void update(double price) {
        System.out.println("Updated stock price: " + price);
    }
}

// Felhasználás
public class Main {
    public static void main(String[] args) {
        Stock stock = new Stock();
        StockDisplay display = new StockDisplay();

        stock.addObserver(display);
        stock.setPrice(100.0);
    }
}
```

Az Observer minta gyakran használt grafikus felhasználói felületeken, valós idejű adatfrissítéseknél és eseményvezérelt rendszerekben. Ez a minta felhasználói felületek frissítését végzi valós időben, például amikor egy chatalkalmazásban új üzenet érkezik, vagy valós idejű adatmegjelenítési rendszerek. Ez a minta lehetővé teszi a rugalmas és skálázható alkalmazások fejlesztését, ahol a komponensek lazán kapcsolódnak egymáshoz. A hátránya ennek a mintának az az, hogy nagy számú megfigyelő esetén a gyakori értesítések teljesítménybeli problémákat okozhatnak.

Készítette:

Bányai Bence

FQT8A1