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

Bevezetés:

A tervezési minták olyan megoldások, melyek segítenek a gondok megoldásában szoftverfejlesztőknek. Ezek a minták nemcsak fontos szerepet töltenek be a kód felépítésében, hanem növelik is fejlesztési folyamat hatékonyságát. A címben látható MVC (modell nézet vezérlő) mintát és sok más tervezési minta áttekintjük, hogy megértsük hogyan lehet őket alkalmazni az objektum orientált programozásban.

Mi az az MVC minta?

Az MVC egy ismert tervezési minta, amit főleg webalkalmazásokban és asztali programokban használnak, és célja, hogy a szoftvert három külön részre osztja:

Model: Ez a réteg felelős az app adatainak kezeléséért Itt tároljuk a biznisz logikát, adatbázis kapcsolatokat és minden olyan funkciót, ami közvetlenül az adatokkal foglalkozik

View (Nézet): A nézet az adatok megjelenítéséért felelős felhasználók számára. A nézet a felhasználói felületet alakítja ki, legyen szó weboldalakról, desktop alkalmazásokról vagy mobil applikációkról. A cél, hogy a felhasználó számára könnyen érthető és használható felületet biztosítsunk.

Controller (Vezérlő): A vezérlő a felhasználói interakciókat kezeli, például gombnyomásokat vagy adatbevitelt, és frissíti a modellt vagy a nézetet. Itt dől el, hogy milyen válaszokat ad a rendszer a felhasználói igényekre.

Miért hasznos az MVC?

Az MVC előnye, hogy világosan elkülöníti a különböző felelősségeket, elősegítve a tiszta kódot és a karbantartást. Az MVC segítségével a felhasználók akár párhuzamosan is dolgozhatnak különböző rétegeken, anélkül, hogy zavarnák egymást, ami nagyon fontos nagyobb projektek esetén. Továbbá az MVC minta széles körben alkalmazható különböző objektum-orientált programozási nyelvekben, mint például Java, C#, vagy Python, amelyek mind népszerűek a webfejlesztésben.

Egyéb tervezési minták:

Az MVC mellett számos más tervezési minta is jelen van az objektumorientált programozásban, ezek lehetővé teszik a tiszta, karbantartható és bővíthető kód létrehozását.

Nézzünk meg néhány tervezési mintát!

Singleton minta:

A Singleton minta biztosítja azt, hogy egy osztályból csak egy példány jöjjön létre, és ez az egyetlen példány bárhonnan elérhető legyen. Ez különösen hasznos lehet olyankor, amikor egy erőforrás, például egy konfigurációs beállítás vagy adatbázis kapcsolat, mindenhol ugyanannak az osztálynak egy konkrét objektumát használja.

Példa használatra:

java kód:

```
public class Singleton {
private static Singleton instance;
private Singleton() {}
public static Singleton getInstance() {
if (instance == null) {
instance = new Singleton ();}
return instance;}}
```

A *Singleton* minta azt biztosítja, hogy többszörös példányokat ne hozzunk létre, így csökkentjük az erőforrások használatát.

Observer minta:

Az *Observer* minta lehetővé teszi, hogy az összes regisztrált megfigyelőt, figyelmeztesse egy objektum (az "alany"), ha egy bizonyos esemény történik. Ez akkor lehet hasznos, ha az alany állapotának változását más objektumoknak is tudomásul kell venniük, anélkül, hogy közvetlen kapcsolat lenne közöttük.

Példa használatra:

java kód:

```
import java.util.ArrayList;
import java.util.List;
interface Observer {
void update(String message);}
class Subject {
private List<Observer> observers = new ArrayList<>();
public void attach(Observer observer) {
observers.add(observer);
}
public void notifyObservers(String message) {
for (Observer observer : observers) {
observer.update(message);}}}
```

Ez a minta nagyon hasznos lehet, ha egy esemény több más komponensre is hatással van, de nem akarjuk közvetlenül összekapcsolni őket.

Factory minta:

A Factory minta abban segít, hogy ne kelljen közvetlenül létrehozni a programban az objektumokat. Egy külön osztály (factory) felelős az objektumok létrehozásáért, így elválasztva a létrehozási logikát a felhasználási logikától.

Példa használatra:

java kód:

```
abstract class Animal {
abstract void makeSound();}
class Dog extends Animal {
void makeSound() {
System.out.println("Bark");}}
class Cat extends Animal {
void makeSound() {
System.out.println("Meow");}}
class AnimalFactory {
public static Animal createAnimal(String type) {
if (type.equals("Dog")) {
return new Dog();
} else if (type.equals("Cat")) {
return new Cat();}
return null;}}
```

A *Factory* mintával a kódunk rugalmasabbá válik, mivel bármikor bővíthetjük új típusokkal anélkül, hogy a fő kódbázist módosítani kellene.

Strategy minta:

A *Strategy* minta lehetővé teszi, hogy egy algoritmust futtassunk anélkül, hogy a felhasználó tudná, hogy hogy van implementálva. Helyezzük az algoritmusokat különböző osztályokba és a program futása közben válasszuk ki, hogy melyiket használjuk.

Példa használatra:

java kód

```
interface Strategy {
int execute(int a, int b);}
class AddStrategy implements Strategy {
public int execute(int a, int b) {
return a + b;}}
class SubtractStrategy implements Strategy {
public int execute(int a, int b) {
return a - b;}}
class Context {
private Strategy strategy;
public Context(Strategy strategy) {
this.strategy = strategy;}
public int executeStrategy(int a, int b) {
return strategy.execute(a, b);}}
```

A mintában lévő kódsor segít abban, hogy egyszerűen válthassunk különböző algoritmusok között, anélkül, hogy a kódunk bonyolulttá válna.

Összegzés

A tervezési minták, mint az MVC, Singleton, Observer, Factory és Strategy, segítenek a szoftverfejlesztésben a kód strukturálásában, a karbantarthatóság javításában és a rugalmasság növelésében. Az MVC minta különösen nagy figyelmet kap a webfejlesztésben, míg a többi minta különböző problémákra kínál megoldást. Ha a mintákat megfelelően alkalmazzuk, akkor jelentősen növelhetjük szoftvereink teljesítményét, fenntarthatóságát és olvashatóságát.

Forrás: Modell-nézet-vezérlő – Wikipédia; Webadatbázis-programozás, internet

Készítette: Barta Csongor (SAOD6J)

PTI-BSC