Tervezési minták egy objektumorientált programozási nyelvben
(Java példákkal)

Bevezetés
Az objektumorientált programozás (OOP) a modern szoftverfejlesztés egyik legfontosabb megközelítése. Lényege, hogy a programokat egymással együttműködő objektumok alkotják, amelyek adatokat és a hozzájuk kapcsolódó műveleteket egyaránt tartalmazzák. Az objektumorientált gondolkodás számos előnyt kínál, például a kód újrafelhasználhatóságát, a könnyebb karbantarthatóságot és a jobb moduláris felépítést.

A gyakorlatban azonban sokszor ugyanazok a tervezési problémák ismétlődnek meg: hogyan kezeljük az objektumok közti kapcsolatokat, hogyan biztosítsuk az egy példányos erőforrásokat, vagy miként válasszunk le egymásról különböző rétegeket?
Ezekre a kérdésekre adnak választ a tervezési minták (design patterns).

Az alábbiakban néhány elterjedt tervezési mintát mutatok be Java nyelven, köztük az MVC (Model–View–Controller) architekturális mintát, valamint a Singleton, Observer és Strategy mintákat.

1. MVC – Model–View–Controller minta
Az MVC minta az egyik legismertebb architekturális tervezési minta, amely az alkalmazás különböző részeit funkcionálisan elkülöníti. Célja, hogy a logika (Model), a felhasználói felület (View) és az irányítás (Controller) ne keveredjenek egymással, így az alkalmazás átláthatóbb, bővíthetőbb és tesztelhetőbb legyen.

Java példa:
```
class Counter {
   private int value = 0;
   public void increment() { value++; }
   public int getValue() { return value; }
}

class CounterView {
   public void show(int value) {
      System.out.println("Számláló értéke: " + value);
   }
}

class CounterController {
   private Counter model;
   private CounterView view;
```

```
    public CounterController(Counter model, CounterView view) {
        this.model = model;
        this.view = view;
    }

    public void increase() {
        model.increment();
        view.show(model.getValue());
    }
}

public class MVCExample {
    public static void main(String[] args) {
        Counter model = new Counter();
        CounterView view = new CounterView();
        CounterController controller = new CounterController(model, view);

        controller.increase();
        controller.increase();
    }
}
```

2. Singleton minta

A Singleton minta célja, hogy egy osztályból a program futása során csak egyetlen példány létezzen. Ez hasznos például adatbázis-kapcsolat, konfigurációs beállítások vagy naplózás esetén.

Java példa:

```
public class Database {
    private static Database instance;

    private Database() {
        System.out.println("Adatbázis kapcsolat létrehozva.");
    }

    public static Database getInstance() {
        if (instance == null) {
            instance = new Database();
        }
        return instance;
    }
```

```
    public void connect() {
        System.out.println("Csatlakozás az adatbázishoz...");
    }
}
```

3. Observer minta

Az Observer (megfigyelő) minta lényege, hogy egy objektum (Subject) értesíti a hozzá kapcsolt megfigyelő objektumokat (Observer-eket) minden változásról. Ezt gyakran használják GUI rendszerekben, eseménykezelésben vagy valós idejű adatszinkronizálásnál.

Java példa:

```
import java.util.*;

interface Observer {
    void update(int value);
}

class Subject {
    private List<Observer> observers = new ArrayList<>();
    private int state;

    public void attach(Observer o) {
        observers.add(o);
    }

    public void setState(int value) {
        state = value;
        for (Observer o : observers) {
            o.update(state);
        }
    }
}

class Display implements Observer {
    public void update(int value) {
        System.out.println("Megfigyelő: új érték -> " + value);
    }
}

public class ObserverDemo {
    public static void main(String[] args) {
        Subject subject = new Subject();
```

```
        subject.attach(new Display());
        subject.setState(10);
        subject.setState(20);
    }
}
```

4. Strategy minta

A Strategy minta segítségével egy objektum különböző algoritmusokat használhat ugyanarra a feladatra, anélkül hogy a fő programkódot módosítani kellene.

Java példa:

```
interface PaymentStrategy {
    void pay(int amount);
}

class CardPayment implements PaymentStrategy {
    public void pay(int amount) {
        System.out.println(amount + " Ft fizetve bankkártyával.");
    }
}

class CashPayment implements PaymentStrategy {
    public void pay(int amount) {
        System.out.println(amount + " Ft fizetve készpénzzel.");
    }
}

public class StrategyDemo {
    public static void main(String[] args) {
        PaymentStrategy strategy = new CardPayment();
        strategy.pay(5000);

        strategy = new CashPayment();
        strategy.pay(2000);
    }
}
```

Összegzés

A tervezési minták alapvető szerepet játszanak az objektumorientált rendszerek felépítésében. Segítenek egységes gondolkodásmódot kialakítani a fejlesztők között, és elősegítik a kód újrafelhasználását, rugalmasságát és karbantarthatóságát.

A bemutatott példák – MVC, Singleton, Observer és Strategy – jól mutatják, hogy a minták nem bonyolultak, viszont jelentősen javíthatják a program szerkezetét.
Ezek a minták a Java nyelvben könnyen alkalmazhatók, és számos modern keretrendszer (például Spring vagy JavaFX) is ezekre a koncepciókra épül.