TERVEZÉSI MINTÁK

6. forduló

IBM

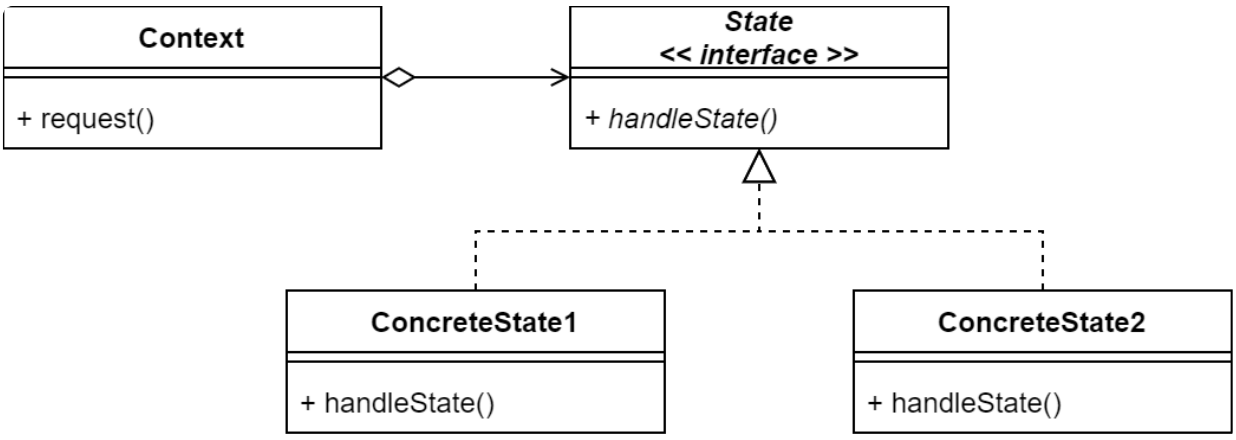
A kategória támogatója: IBM

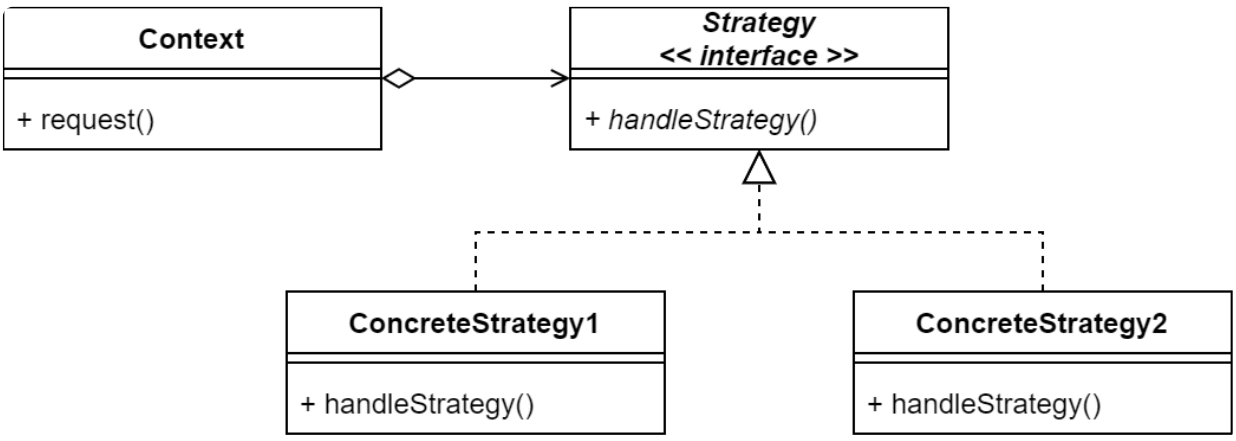
Ismertető a feladathoz

Felhasznált idő: 05:00/05:00 **Elért pontszám:** 0/4

1. feladat 0/1 pont

A State és Strategy minták osztálydiagramja gyakorlatilag ugyanúgy néz ki:





Mi a különbség a minták között?

Válasz

- (●) A különbség csupán logikai: a State esetén a választott implementációtól az függ, hogy mit csinál az alkalmazás, míg Strategy esetén ugyanazt csinálja, csak más algoritmussal
  Ez a válasz helyes, de nem jelölted meg.
- ( ) Semmi: a State a régi, míg a Strategy a minta modern neve, amit azért választottak, hogy egyértelműbb legyen a célja
- ( ) Az osztálydiagrammok egyezése csupán véletlen, két teljesen más mintáról van szó. A State az alkalmazás belső állapotának mentésére és visszatöltésére való, míg a Strategy segítségével különböző stratégiákat tudunk definiálni egy probléma megoldására.
- ( ) Semmi: ugyanazt a mintát jelöli mindkettő

Magyarázat

A megoldás magáért beszél.

2. feladat 0/1 pont

Egy okosotthonban a különböző eszközök közötti interakcióra szabályokat szeretnénk alkotni. Például, ha túl meleg van, automatikusan leengedjük a redőnyt és bekapcsoljuk a klímát. Mely minták alkalmasak erre?

Válaszok

- [x] Observer
  Ez a válasz helyes, de nem jelölted meg.
- [x] Mediator
  Ez a válasz helyes, de nem jelölted meg.
- [ ] Adapter
- [ ] Chain of Responsibility
- [ ] Egyik sem

Magyarázat

Az Observer és a Mediator is több különböző objektum együttműködését célozza úgy, hogy azok lazán legyenek csatolva. A különbség az, hogy míg Mediator esetén van központi vezérlő, az Observer esetén nincs.

Az Adapternek és a Chain of Responsiblity-nek nem célja több résztvevő együttműködésének összehangolása.

3. feladat 0/1 pont

Adottak az alábbi osztályok:

```
interface Shape {
    double calculate(Calculator calculator);
}

class Circle implements Shape {
    double radius;

    double getRadius() {
        return radius;
    }
```

```
    @Override
    public double calculate(Calculator calculator) {
        return calculator.calculateForCircle(this);
    }
}

class Square implements Shape {
    double side;

    double getSide() {
        return side;
    }

    @Override
    public double calculate(Calculator calculator) {
        return calculator.calculateForSquare(this);
    }
}

interface Calculator {
    double calculateForSquare(Square square);
    double calculateForCircle(Circle circle);
}

class AreaCalculator implements Calculator {
    @Override
    public double calculateForSquare(Square square) {
        return Math.pow(square.getSide(), 2);
    }

    @Override
    public double calculateForCircle(Circle circle) {
        return Math.pow(circle.getRadius(), 2) * Math.PI;
    }
}

class CircumferenceCalculator implements Calculator {
    @Override
    public double calculateForSquare(Square square) {
        return square.getSide() * 4;
    }

    @Override
    public double calculateForCircle(Circle circle) {
        return 2 * circle.getRadius() * Math.PI;
    }
}
```

A következő kódrészlet mutat egy példát a használatra:

```
Shape shape = new Circle();
Calculator calculator = new AreaCalculator();
double circleArea = shape.calculate(calculator);
```

Mely minta megvalósítása ez?

Válasz

- Visitor

Ez a válasz helyes, de nem jelölted meg.

- Interpreter
- Bridge
- Observer
- Egyik sem

Magyarázat

Ez egy klasszikus példa a Visitor minta megvalósítására. A Calculator (Visitor) implementációk tartalmazzák egy konkrét tulajdonság kiszámolásának megvalósítását. Mivel minden osztályra külön metódust hoztunk létre, így castolás nélkül is típusbiztosan működik a megvalósítás. Ezt az egyes Shape (Element) implementációk oldják meg úgy, hogy a method overloading-ot kihasználva a fordítóra bízzák a megfelelő metódus meghívását.

4. feladat 0/1 pont

Ugyanazzal a létrehozási folyamattal több, különböző osztályt szeretnénk példányosítani. A folyamat során végrehajtott lépésekben használandó konfigurációt szeretnénk testre szabhatóvá tenni. Mely minták alkalmasak erre?

Válasz

- [x] Builder
  Ez a válasz helyes, de nem jelölted meg.
- [ ] Bridge
- [ ] Adapter
- [ ] Command
- [ ] Egyik sem

Magyarázat

A Buildernek pontosan ez a célja. A Bridge és az Adapter szerkezeti, a Command viselkedési minta, így az objektumok létrehozása egyiknek sem célja.

© 2023 Human Priority Kft.