TERVEZÉSI MINTÁK

7. forduló

IBM

A kategória támogatója: IBM

Ismertető a feladathoz

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

1. feladat 0/1 pont

Matematikai kifejezéseket szeretnénk modellezni és kiértékelni. Mely minták alkalmasak erre?

Válasz

- (●) Interpreter
  Ez a válasz helyes, de nem jelölted meg.
- ( ) Evaluator
- ( ) Command
- ( ) Composite
- ( ) Egyik sem

Magyarázat

Az Interpreter célja egy konkrét nyelv reprezentációja és annak értelmezése. Ez a nyelv akár a matematika is lehet.

Evaluator minta nem létezik.

A Composite korlátozottan alkalmas lehetne a modellezésre, de a kiértékelésre nem.

A Command célja kérések létrehozásának, tárolásának és végrehajtásának függetlenítése időben és helyben, így ennek nem sok köze van a problémához.

2. feladat 0/1 pont

Egy adatszerkezetben tárolt elemekhez szeretnénk szekvenciális hozzáférést adni anélkül, hogy felfednénk az adatszerkezet implementációját. Mely minták alkalmasak erre?

Válasz

- (●) Iterator
  Ez a válasz helyes, de nem jelölted meg.
- ( ) Observer
- ( ) State
- ( ) Object Loop
- ( ) Egyik sem

Magyarázat

Erre találták ki az Iterator mintát.

Az Observer célja egy objektum változásaira való feliratkozás.

A State állapottól függően változó viselkedés modellezésére való.

Object Loop nem létezik, csupán az Object Pool kreatív átnevezése.

3. feladat 0/1 pont

Egy autógyár különböző karosszériaváltozatokhoz (kombi, lépcsőshátú) gyárt alkatrészeket (csomagtartó, lökhárító). A különböző karosszériaváltozatokhoz tartozó alkatrészek nem kompatibilisek (kombi csomagtartó a lépcsőshátú lökhárítóval). Szeretnénk modellezni egy autó gyártási folyamatát, aminek során ki akarjuk kényszeríteni, hogy az alkatrészek ugyanahhoz a karosszériaváltozathoz készüljenek el. Mely minta alkalmas erre?

Válasz

- (●) Abstract Factory
  Ez a válasz helyes, de nem jelölted meg.
- ( ) Bridge
- ( ) Prototype
- ( ) Singleton
- ( ) Egyik sem

Magyarázat

Az Abstract Factory-nak pontosan ez a célja: egy összetartozó objektumcsalád elemeinek példányosítása annak kikényszerítésével, hogy ne a másik családból jöjjenek létre az objektumok.

A többi minta ilyet nem tud kikényszeríteni.

4. feladat 0/1 pont

Adottak az alábbi osztályok:

```
class NumberTransformer {
    final NumberTransformer nextTransformer;

    NumberTransformer(NumberTransformer nextTransformer) {
        this.nextTransformer = nextTransformer;
    }
```

```
    String transform(int number) {
        return nextTransformer.transform(number);
    }

    static NumberTransformer createFizzBuzzTransformer() {
        ToStringTransformer toStringTransformer = new ToStringTransformer(null);
        BuzzTransformer buzzTransformer = new BuzzTransformer(toStringTransformer);
        FizzTransformer fizzTransformer = new FizzTransformer(buzzTransformer);
        return new FizzBuzzTransformer(fizzTransformer);
    }
}

class FizzBuzzTransformer extends NumberTransformer {
    FizzBuzzTransformer(NumberTransformer nextTransformer) {
        super(nextTransformer);
    }

    @Override
    String transform(int number) {
        if (number % 15 == 0) {
            return "FizzBuzz";
        }

        return super.transform(number);
    }
}

class FizzTransformer extends NumberTransformer {
    FizzTransformer(NumberTransformer nextTransformer) {
        super(nextTransformer);
    }

    @Override
    String transform(int number) {
        if (number % 3 == 0) {
            return "Fizz";
        }

        return super.transform(number);
    }
}

class BuzzTransformer extends NumberTransformer {
    BuzzTransformer(NumberTransformer nextTransformer) {
        super(nextTransformer);
    }

    @Override
    String transform(int number) {
        if (number % 5 == 0) {
            return "Buzz";
        }

        return super.transform(number);
    }
}

class ToStringTransformer extends NumberTransformer {
    ToStringTransformer(NumberTransformer nextTransformer) {
        super(nextTransformer);
    }

    @Override
```

```
    String transform(int number) {
        return String.valueOf(number);
    }
}
```

A következő kódrészlet mutat egy példát a használatra:

```
NumberTransformer numberTransformer = NumberTransformer.createFizzBuzzTransformer();
System.out.println(numberTransformer.transform(3)); // prints Fizz
```

Mely minta megvalósítása ez?

Válasz

- (●) Chain of Responsibility
  Ez a válasz helyes, de nem jelölted meg.
- ( ) Template Method
- ( ) Strategy
- ( ) Interpreter
- ( ) Egyik sem

Magyarázat

Ez egy klasszikus példa a Chain of Responsibility minta megvalósítására. A NumberTransformer definiál egy ősosztályt, ami azonnal delegálja a kérést a lánc következő elemének. Az egyes leszármazottak egy-egy konkrét eset kezelését valósítják meg (ha képesek rá). A végén a NumberTransformer.createFizzBuzzTransformer() állítja össze a láncot, ami a teljes logikát magába foglalja (prioritással együtt).