TERVEZÉSI MINTÁK

2. forduló

IBM

A kategória támogatója: IBM

Ismertető a feladathoz

Útmutató:

- A **radio button-os kérdésekre** egy helyes válasz van.
- **Ha lejár a feladatlap ideje, a rendszer AUTOMATIKUSAN** beküldi azt az addig megjelölt válaszokkal.
- Az **adatbekérős feladatokra NEM jár részpontszám**, csak a feleletválasztósakra.
- **Badge-ket** a 4.forduló után kapsz majd először.
- Az **adatbekérős kérdéseknél** igyekeztünk minden variációt megadni (kisbetű, nagybetű, szóköz), de ha mégis eltérést tapasztalsz a megoldásokban, kérjük, jelezd felénk!

+1: Azért szólunk, hogy senkit ne a végén érjen meglepetés: a játék nem tipp-mix és csapatkategória sincs! Természetesen akinek nem inge...

Jó versenyzést kívánunk!

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

1. feladat 0/1 pont

Mi a SOLID betűszó feloldása?

Válasz

- ○ Single Responsibility Principle
  Open/Closed Principle
  Liskov Substitution Principle
  Interface Segregation Principle
  Demeter's Law
- ○ Singleton Is an Antipattern
  Object-Orientation
  Loose Coupling
  Inheritance Over Composition
  Depend on Abstractions
- ○ Separate Business Logic

Occam's Razor

Least Astonishment Rule

Inversion of Control

Don't Repeat Yourself

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

Magyarázat

A helyes feloldás:

Single Responsibility Principle

Open/Closed Principle

Liskov Substitution Principle

Interface Segregation Principle

Dependency Inversion Principle

2. feladat 0/1 pont

Adott a következő kódváz, amivel a Singleton tervezési mintát szeretnénk szálbiztosan megvalósítani.

```
class Singleton {
    // 1

    /* 2 */ Singleton() {
    }

    public static Singleton getInstance() {
        // 3
    }

    // üzleti metódusok
}
```

Milyen láthatóságú lehet a konstruktor, azaz mi kerüljön a /* 2 */ helyére?

Válasz

- [x] private
  Ez a válasz helyes, de nem jelölted meg.
- [ ] package private (nincs láthatósági módosító)
- [ ] protected
- [ ] public

Magyarázat

Mivel meg akarunk akadályozni minden külső példányosítást, így a konstruktornak kötelezően privátnak kell lennie.

3. feladat 0/1 pont

Adott a következő kódváz, amivel a Singleton tervezési mintát szeretnénk szálbiztosan megvalósítani.

```
class Singleton {
    // 1

    /* 2 */ Singleton() {
    }

    public static Singleton getInstance() {
        // 3
    }

    // üzleti metódusok
}
```

Hogyan kezeljük a példányt és az ahhoz való hozzáférést?

Válaszok

- [x] Az // 1 helyére:

```
private static final Singleton INSTANCE = new Singleton();
```

És a // 3 helyére:

```
return INSTANCE;
```

Ez a válasz helyes, de nem jelölted meg.

- [ ] Az // 1 helyére:

```
private static Singleton INSTANCE;
```

És a // 3 helyére:

```
if (INSTANCE == null) {
    INSTANCE = new Singleton();
}
return INSTANCE;
```

- [x] Az // 1 helyére:

```
private static Singleton INSTANCE;
```

És a // 3 helyére:

```
synchronized (Singleton.class) {
    if (INSTANCE == null) {
        INSTANCE = new Singleton();
    }
}
return INSTANCE;
```

Ez a válasz helyes, de nem jelölted meg.

- [ ] Az // 1 helyére:

```
private static Singleton INSTANCE;
```

És a // 3 helyére:

```
if (INSTANCE == null) {
    synchronized (Singleton.class) {
        INSTANCE = new Singleton();
    }
}
return INSTANCE;
```

- [ ] Egyik sem

Magyarázat

A példány létrehozásának szálbiztosnak kell lennie, különben előfordulhat, hogy több példány jön létre.

Az inicializátor esetén a classloader gondoskodik a szálbiztosságról, így az a megoldás jó.

Amikor kívül van a szinkronizáció, nem fordulhat elő, hogy egyszerre két szál párhuzamosan azt érzékelje, hogy még nincs inicializálva a példány, így itt sem jöhet létre több példány.

A másik két megoldás esetén nem garantált a kölcsönös kizárás, így azok nem helyes implementációk.

4. feladat 0/1 pont

Egy komplex alrendszer műveleteit szeretnénk kiajánlani anélkül, hogy felfednénk az alrendszer implementációját. Mely minták alkalmasak erre?

Válaszok

- [x] Façade
  Ez a válasz helyes, de nem jelölted meg.
- [x] Bridge
  Ez a válasz helyes, de nem jelölted meg.
- [ ] Mediator
- [ ] Interpreter
- [ ] Egyik sem

Magyarázat

Az Interpreter nyelvek reprezentációjára alkalmas.

A Mediator objektumok együttműködését teszi lehetővé úgy, hogy megmaradjon köztük a laza csatolás.

A Façade egy egyszerűsített interface-t nyújt több komponens elfedésével.

A Bridge az absztrakciót elválasztja az implementációtól, ezzel elfedve az utóbbit.

Ezek alapján a két utóbbi lehet alkalmas a feladatra.