JAVA 11

5. forduló

A kategória támogatója: IBM

RENDELKEZÉSRE ÁLLÓ IDŐ:

15:00

Ismertető a feladathoz

Felhasznált idő: 02:06/15:00 **Elért pontszám:** 0/35

1. feladat 0/5 pont

Melyik esetekben lesz az A és B osztály között **kompozíciós (composition)** kapcsolat?

Válasz

- ○
```
class A extends B {}
```
- ○
```
class A implements B {}
```
- ○
```
class A {
  B b;
}
```
- ◯
```
class A {
  class B {}
}
```

```
class A {
  static class B {}
}
```

```
class A {
  void foo(B b) {}
}
```

```
class A {
  B foo() { return new B(); }
}
```

Magyarázat

Kompozíció esetén a tartalmazott nem tud létezni a tartalmazó nélkül, ez pedig csak a nem statikus belső osztállyal valósítható meg.

2. feladat 0/5 pont

Melyik esetekben lesz az A és B osztály között **aggregációs (aggregation)** kapcsolat?

Válasz

```
class A extends B {}
```

```
class A implements B {}
```

```
class A {
  B b;
}
```

```
class A {
  class B {}
}
```

```
class A {
  static class B {}
}
```

```
class A {
  void foo(B b) {}
}
```

```
class A {
  B foo() { return new B(); }
}
```

Magyarázat

Asszociáció és aggregáció esetén tartós kapcsolat van az osztályok példányai között, ami tagváltozóban tárolt referencián valósítható meg. A különbség logikai, azaz hogy rész-egész viszonyban vannak-e. Pl. ha A az autó, B a kerék, az aggregáció, ha A egy ember, B pedig egy ismerőse, az asszociáció.

3. feladat 0/5 pont

Melyik esetekben lesz az A és B osztály között **asszociációs (association)** kapcsolat?

Válasz

```
class A extends B {}
```

```
class A implements B {}
```

```
class A {
  B b;
}
```

```
class A {
  class B {}
}
```

```
class A {
  static class B {}
}
```

```
class A {
  void foo(B b) {}
}
```

```
class A {
  B foo() { return new B(); }
}
```

Magyarázat

Asszociáció és aggregáció esetén tartós kapcsolat van az osztályok példányai között, ami tagváltozóban tárolt referencián valósítható meg. A különbség logikai, azaz hogy rész-egész viszonyban vannak-e. Pl. ha A az autó, B a kerék, az aggregáció, ha A egy ember, B pedig egy ismerőse, az asszociáció.

4. feladat 0/0 pont

Melyik esetekben lesz az A és B osztály között **függési (dependency)** kapcsolat?

Válaszok

```
class A extends B {}
```

```
class A implements B {}
```

- [ ]
```
class A {
  B b;
}
```

- [ ]
```
class A {
  class B {}
}
```

- [ ]
```
class A {
  static class B {}
}
```

- [x]
```
class A {
  void foo(B b) {}
}
```

- [x]
```
class A {
  B foo() { return new B(); }
}
```

Magyarázat

Függés esetén időleges a kapcsolat, például paraméterátadás vagy factory metódus segítségével.

Frissítés (2021.11.19.): Egy technikai hiba miatt kimaradt a feladatok szövegéből, hogy az UML definíciói szerinti kapcsolatokról van szó. A feladatot ezért 0 pontosra állítottuk.

5. feladat 0/10 pont

Adott az alábbi osztály:

```
package com.ibm;
class Outer {
  class Nested {}

  Nested createNested() {
    return new Nested();
  }
}
```

És adott a felhasználásának a helye:

```
package com.ibm;

import com.ibm.Outer;
import com.ibm.Outer.Nested;

class NestedDependency {
  void useNested() {
    Outer o = new Outer();
    Nested n = /* 1 */;
  }
}
```

Az alábbiak közül mely értékadások **nem váltanak ki fordítási hibát** az /* 1 */ helyére illesztve?

Válaszok

- [ ] `new Nested()`
- [ ] `new Outer.Nested()`
- [x] `o.createNested()`
- [ ] `new Outer().Nested()`
- [x] `new Outer().new Nested()`
- [ ] `new o.Nested()`

- [x] `o.new Nested()`
- [ ] `Outer.createNested()`
- [x] `new Outer().createNested()`

Magyarázat

A Nested egy nem statikus belső osztály, ezért a tartalmazó osztály referenciája nélkül nem lehet példányosítani. Így a következők hamisak:

```
new Nested()
new Outer.Nested()
```

Az alábbinál a Nested nevű metódust hívnák meg, ami nem létezik:

```
new Outer().Nested()
```

Az alábbinál egy nemlétező metódushívás és példányosítás van kombinálva, ami sok sebből vérzik:

```
new o.Nested()
```

A createNested() metódus nem statikus, így a következő helytelen:

```
Outer.createNested()
```

A többi helyes – bár nem feltétlenül szép.

6. feladat 0/5 pont

Adott a következő interface:

```
public interface Super {
  void operation();
}
```

Az alábbiak közül mely interface deklarációk **nem váltanak ki fordítási hibát**, ha az importok rendben vannak?

Válaszok

- [x]
```
interface Child extends Super {}
```

- [x]
```
interface Child extends Super {
  void operation();
}
```

- [ ]
```
interface Child extends Super {
  default void operation();
}
```

- [x]
```
interface Child extends Super {
  default void operation() {}
}
```

Magyarázat

Nem adtunk implementációt az operation metódushoz annak ellenére, hogy default-ként jelöltük meg.

7. feladat 0/5 pont

Adott a következő két interface:

```
interface Operation1 {
  default void operation() {
    System.out.println("1");
  }
}

interface Operation2 {
  default void operation() {
    System.out.println("2");
  }
}
```

Az alábbi osztálydeklarációk közül melyek **nem váltanak ki fordítási hibát**, feltéve hogy nincs láthatósági probléma?

Válaszok

- [x]
```
class ConcreteOperation implements Operation1 {}
```

- [x]
```
class ConcreteOperation implements Operation2 {}
```

- [ ]
```
class ConcreteOperation implements Operation1, Operation2 {}
```

- [x]
```
class ConcreteOperation implements Operation1, Operation2 {
  @Override
  public void operation() {
    Operation1.super.operation();
  }
}
```

- [x]
```
class ConcreteOperation implements Operation1, Operation2 {
  @Override
  public void operation() {
    System.out.println("1");
  }
}
```

Magyarázat

A hibás osztály két azonos szignatúrájú, de különböző helyen implementált default metódust is örököl. A fordító nem tudja eldönteni, hogy melyiket használja, ezért hibát jelez.