HATÉKONY JAVA PROGRAMOZÁS

2. forduló

MSCI

A kategória támogatója: MSCI

RENDELKEZÉSRE ÁLLÓ IDŐ:

10:00

Ismertető a feladathoz

Fontos információk

Ha kifutsz az adott feladatlap kitöltésére rendelkezésre álló időből, a felület **automatikusan megpróbálja beküldeni** az addig megadott válaszokat

A kérdésekre **mindig van helyes válasz**! Ha csak egy helyes válasz van az adott kérdésre, radio button-os választási lehetőségeket fogsz látni.

Olyan kérdés viszont nincs, amelyre az összes válasz helyes!

Egyéb információkat a versenyszabályzatban találsz!

Felhasznált idő: 02:07/10:00 **Elért pontszám:** 0/5

1. feladat 0/1 pont

Mivel egészítsük ki a kódrészletet, hogy az a lehető leggyorsabban lefusson?

```
public List<Integer> generate() {
        List<Integer> test = ...;
        for (int i = 0; i < 1_000_000; i++) {
                test.add(i);
        }
        return test;
}
```

Válasz

- [ ] new CopyOnWriteArrayList<>()
- [x] new LinkedList<>()
- [ ] new Vector<>()
- [ ] new List<>()

Magyarázat

A CopyOnWriteArrayList lassú egyszálú környezetben.

A List nem példányosítható.

A Vector helyett az ArrayList használata javasolt.

A felsorolt lehetőségek közül a LinkedList a leghatékonyabb.

2. feladat 0/1 pont

Mivel egészítsük ki a kódrészletet, hogy az a lehető leggyorsabban lefusson?

```
public List<Integer> generate() {
        List<Integer> test = ...;
        for (int i = 0; i < 1_000_000; i++) {
                test.add(0, i);
        }
        return test;
}
```

Válasz

- [ ] new ArrayList<>(1_000_000)
- [x] new LinkedList<>()
- [ ] new Vector<>()
- [ ] new ArrayList<>()

Magyarázat

Az első helyre való beszúrás csak a LinkedList esetén hatékony.

3. feladat 0/1 pont

A beépített HashSet osztály egy HashMap segítségével valósítja meg a működését. Melyik egy valós különbség a két osztály között?

```
public class HashSet<E>
        extends AbstractSet<E>
        implements Set<E>, Cloneable, java.io.Serializable
{
        ...
        private transient HashMap<E,Object> map;

        // Dummy value to associate with an Object in the backing Map
        private static final Object PRESENT = new Object();

        /**
         * Constructs a new, empty set; the backing {@code HashMap} instance has
         * default initial capacity (16) and load factor (0.75).
         */
        public HashSet() {
                map = new HashMap<>();
        }
        ...
}
```

Válasz

- A HashSet szálbiztos, de a HashMap nem
- A HashMap egy kulcsot többször is el tud tárolni, a HashSet egy értéket csak egyszer tárol
- A HashSet sorosítható, de a HashMap nem
- A HashMap a kulcshoz értékeket rendel, míg a HashSet nem

Magyarázat

A HashSet egy halmazt képez, a HashMap kulcs-érték párokat reprezentál.

4. feladat 0/1 pont

Az alább látható Double Brace Initialization mintával létrehozhatunk egy kollekciót előre definiált elemekkel. Hogyan magyarázható a működése?

```
List<Integer> list = new ArrayList<Integer>() {{
    add(1);
```

```
        add(2);
    }};
```

Válasz

- A Double Brace Initialization része a Java nyelv specifikációjának (abban külön fejezet szól erről) és eredetileg pont ilyen kollekciók létrehozására alkották meg
- Egy névtelen ArrayList leszármazottat (anonymous inner class) hoz létre és a konstruktor előtt lefutó inicializációs blokkban adja hozzá az elemeket
- Egy lambda kifejezéssel kiegészíti az ArrayList konstruktorát, mely a konstruktor után automatikusan lefutva hozzá tudja adni a kívánt számokat
- Ez a minta csak speciális környezetben működik (pl. Lombok vagy IntelliJ IDEA használata), nem része a Java nyelvnek

Magyarázat

A leszármazott névtelen osztály inicializációs blokkja a leszármazott konstruktora előtt fut le, de mindez csak az ősosztály létrehozása után történik meg, tehát hozzá lehet adni elemeket.

5. feladat 0/1 pont

Melyik egy hatékony megvalósítása egy immutable (megváltoztathatatlan) és üres kollekció visszaadásának?

Válasz

-
```
List<Integer> getImmutableAndEmptyList() {
        return new ArrayList<>();
}
```

-
```
private static final List<Integer> EMPTY = new ArrayList<>();
List<Integer> getImmutableAndEmptyList() {
        return EMPTY;
}
```

-
```
private static final List<Integer> EMPTY = Collections.synchronizedList(new ArrayList<>());
List<Integer> getImmutableAndEmptyList() {
        return EMPTY;
}
```

```
List<Integer> getImmutableAndEmptyList() {
        return List.of();
}
```

Magyarázat

Ez az egyedüli válaszlehetőség, amelyik valóban immutable kollekciót használ. A többi kollekció módosítható a visszaadás után.