JAVA PROGRAMOZÁS

5. forduló

A kategória támogatója: Click Clock by BCS

Ismertető a feladatlaphoz

> Kezdj neki minél hamarabb, mert a feladatot a forduló záró időpontjáig lehet beküldeni, nem addig lehet elkezdeni!
>
> **A feladatot ajánljuk nem mobilon/tableten játszani!**

Sok sikert!

A fordulóban elhangzó kérdések mindegyike Java 17 alapokon értelmezendő. Ha valamelyik kérdés JDK-ban használt eszközre kérdez rá, ott mindig az OpenJDK-t vegyük alapul.

Az említett verzió a következő linken tölthető le: https://jdk.java.net/java-se-ri/17

1. feladat 0 pont

Melyik SOLID elv sérül a következő kódrészletben?

```java
import java.math.BigDecimal;

interface Account {

  void depositMoney(BigDecimal amount);

  void transferToPartner(String partnerIdentifier,
                         BigDecimal amount);

}

class Bank {

  public static void main(String[] args) {
    Account account = new MainAccount("Darth Vader");
    account.depositMoney(BigDecimal.TEN);
    account.transferToPartner("Anakin Skywalker", BigDecimal.TEN);

    account = new SavingsAccount("Darth Vader");
    account.depositMoney(BigDecimal.TEN);
    account.transferToPartner("Anakin Skywalker", BigDecimal.TEN);
  }

}

class MainAccount implements Account {

  private final String owner;

  MainAccount(String owner) {
    this.owner = owner;
  }

  @Override
  public void depositMoney(BigDecimal amount) {
    System.out.println("Depositing money:[" + amount + "] to main account");
  }

  @Override
  public void transferToPartner(String partnerIdentifier, BigDecimal amount) {
    System.out.println("Transferring [" + amount + "] amount" +
      " of money to partner:[" + partnerIdentifier + "]");
  }
}

class SavingsAccount implements Account {

  private final String owner;

  SavingsAccount(String owner) {
    this.owner = owner;
  }

  @Override
  public void depositMoney(BigDecimal amount) {
    System.out.println("Depositing money:[" + amount + "] to savings account");
  }

  @Override
  public void transferToPartner(String partnerIdentifier, BigDecimal amount) {
    throw new UnsupportedOperationException("Savings account is not able to
transfer money to partners.");
  }
}

```

Válasz

- SRP - Single-responsibility principle
- OCP - Open–closed principle
- LSP - Liskov Substitution Principle
- ISP - Interface segregation principle
- DIP - Dependency inversion principle

2. feladat 2 pont

A következő végrehajtó (executor) osztályok közül melyik dolgozik "munka lopási" (work-steal) algoritmussal?

Válasz

- ScheduledThreadPoolExecutor
- ThreadPoolExecutor
- MultiTaskProcessor
- ForkJoinPool

3. feladat 2 pont

Mi fog történni, ha a 'test.txt' fájl nem létezik a következő kód lefutása után?

```java
import java.io.IOException;
import java.nio.file.Files;
import java.nio.file.Paths;

class FileTask {

  public static void main(String[] args) throws IOException {
    Files.delete(Paths.get("test.txt"));
  }

}

```

Válasz

- A fájl létre lesz hozva a Paths.get() használata miatt.
- Semmi nem fog történni.
- Egy NoSuchFileException kivétel fog dobódni.
- A kód a szülő könyvtárat fogja törölni.

4. feladat 3 pont

Melyik kódrészletet kell írnunk a kérdőjeles helyre, ha meg szeretnénk tudni, hogy mennyi a 30 év feletti emberek átlagfizetése városonként?

```java
import java.util.List;

record Person(
  String name,
  int age,
  String city,
  double salary) {
}

class StreamUsage {

  public static void main(String[] args) {
    var people = List.of(
      new Person("Zsolt", 35, "Szeged", 750_000),
      new Person("Alíz", 25, "Szeged", 500_000),
      new Person("Barnabás", 27, "Debrecen", 600_000),
      new Person("Elemér", 33, "Budapest", 700_000),
      new Person("Gábor", 36, "Szeged", 800_000),
      new Person("Péter", 40, "Budapest", 900_000),
      new Person("Fanni", 42, "Debrecen", 1_000_000),
      new Person("Zsófia", 45, "Szeged", 950_000),
      new Person("Ágoston", 37, "Győr", 875_000),
      new Person("Adalbert", 32, "Nyíregyháza", 700_000)
    );
    var averageSalaries = /* ? */
    System.out.println(averageSalaries);

  }

}


```

Válasz

```
people.stream()
    .filter(p -> p.age() >= 30)
    .collect(Collectors.toMap(
        Person::city,
        Person::salary
    ));
```

```
people.stream()
    .filter(p -> p.age() >= 30)
    .collect(Collectors.groupingBy(Person::city));
```

```
people.stream()
    .filter(p -> p.age() >= 30)
    .collect(Collectors.groupingBy(
        Person::city,
        Collectors.averagingDouble(Person::salary)
    ));
```

```
people.stream()
      .filter(p -> p.age() >= 30)
      .mapToDouble(Person::salary)
      .boxed()
      .collect(Collectors.toMap(
        p -> p.city(),
        Function.identity()
      ));
```

5. feladat 3 pont

A következő Garbage Collector-ok közül melyek nem generációsak?

Válaszok

- [ ] ZGC
- [ ] Parallel GC
- [ ] G1 GC
- [ ] Shenandoah
- [ ] Serial GC

6. feladat 3 pont

Milyen GC-t fog választani a JVM abban az esetben, ha az alkalmazást futtató eszköz (gép, hardware, szerver) 1GB memóriával és 2 magos CPU-val rendelkezik?

Válasz

- ZGC
- SerialGC
- EpsilonGC
- G1GC

7. feladat 2 pont

Melyik Java kapcsolóval engedélyezhetjük az Epsilon nevű Garbage Collector-t (GC) ?

Válasz

- -XX:+UseEpsilonGC
- -XX:+UnlockExperimentalVMOptions -XX:+UseEpsilonGC
- -XX:+UseEpsilon
- -XX:+UnlockExperimentalVMOptions -XX:+UseEpsilon
- Egyik sem

8. feladat 4 pont

Milyen JIT-es optimalizáción eshet át a következő kódrészlet?

```java
public class InterestCalculator {

  public static void main(String[] args) {
    var maximumPrincipal = Long.valueOf(args[0]);
    var principalStepSize = Long.valueOf(args[1]);
    var rate = Double.valueOf(args[2]);
    var timeInYears = Long.valueOf(args[3]);

    var app = new InterestCalculator();
    for (var principal = 1; principal <= maximumPrincipal; principal += principalStepSize) {
      var interest = app.calculate(principal, rate, timeInYears);
      System.out.println("""
        The interest for %d years with
        a rate of %.2f %% for a %d HUF is: %.2f
        """.formatted(timeInYears, rate, principal, interest));
    }
  }

  double calculate(double principal, double rate, double time) {
    return new InvoiceCalculator(principal, rate, time)
      .calculateInterest();
  }

  static class InvoiceCalculator {
    private double principal;
    private double rate;
    private double time;

    public InvoiceCalculator(double principal, double rate, double time) {
      this.principal = principal;
      this.rate = rate;
      this.time = time;
    }

    double calculateInterest() {
      return principal * Math.pow((1 + rate / 100), time);
    }

  }

}
```

Válasz

- ( ) Constant inlining
- ( ) Escape Analysis
- ( ) Branch prediction
- ( ) Loop unrolling

9. feladat 2 pont

Mi történik, ha egy már előzőleg a shutdown() metódus használatával leállított ExecutorService példányon újabb feladatot szeretnénk rögzíteni?

Válasz

- A feladat sikeresen elindul.
- A feladat bekerül a futtató által birtokolt sorba és ha újra elindul a szolgáltatás elindítja.
- Újraindítja a leállított szolgáltatást és elindítja a feladatot.
- A hívás `RejectedExecutionException` kivételt dob.

Megoldások beküldése