ALTFRAGEN

Samstag, 4. November 2023 14:51

Frage 1
Bisher nicht beantwortet
Erreichbare Punkte: 4,00
Frage markieren

4 Punkte (1 Punkt pro korrekte Antwort)

```
public class Decrementer extends Thread {
      private static int counter = 0;
      private final int N;

      public Decrementer(int N) { this.N = N;}

      public static int getCounter() { return counter; }

      public void run() {
            for (int i=0; i < N; i++) decrement();
      }

      private void decrement() { counter--; }
}
```

Wählen Sie für die Methode **decrement()** jene Varianten aus, sodass die private Klassenvariable **counter** gleichzeitig von mehreren Instanzen der Klasse **Decrementer** korrekt dekrementiert werden kann.

Wählen Sie für jede der unten angeführten Optionen, ob diese im oben beschriebenen Kontext **korrekt** oder **inkorrekt** ist.

```
private void decrement() {
   synchronized (getClass()) { counter--; }
}
```
Auswählen ...

```
private synchronized void decrement() { counter--; }
```
Auswählen ...

```
private void decrement() {
   synchronized (this) { counter--; }
}
```
Auswählen ...

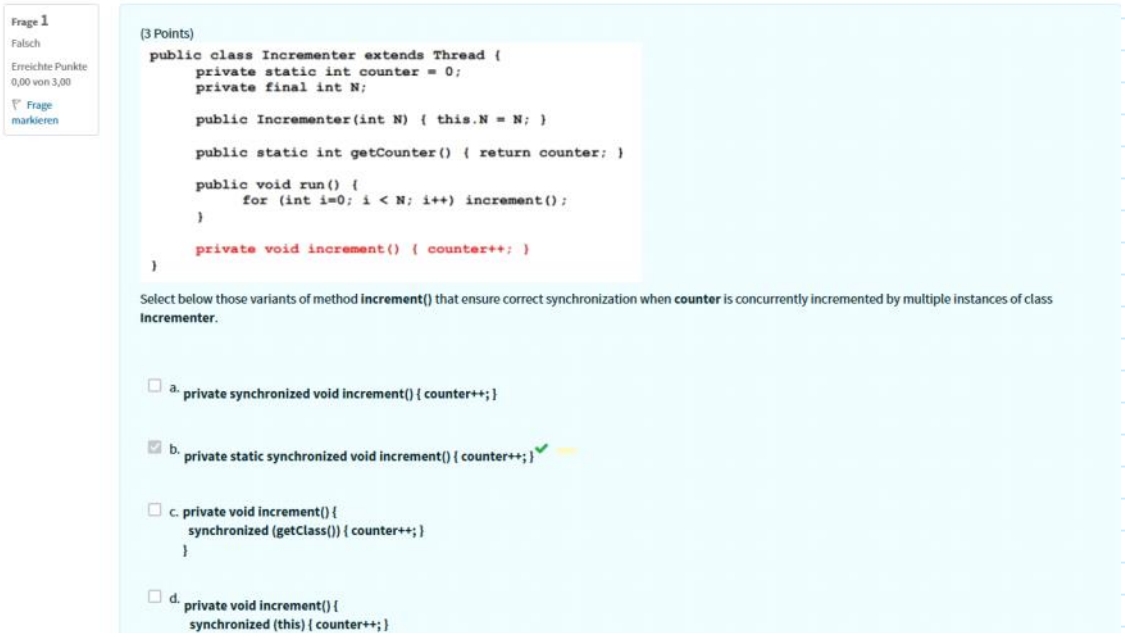
```
private static synchronized void decrement() { counter--; }
```
Auswählen ...

1) correct

2) Incorrect

3) Incorrect

4) Correct



Frage 1
Falsch
Erreichte Punkte 0,00 von 3,00
Frage markieren

(3 Points)

```
public class Incrementer extends Thread {
      private static int counter = 0;
      private final int N;

      public Incrementer(int N) { this.N = N; }

      public static int getCounter() { return counter; }

      public void run() {
            for (int i=0; i < N; i++) increment();
      }

      private void increment() { counter++; }
}
```

Select below those variants of method **increment()** that ensure correct synchronization when **counter** is concurrently incremented by multiple instances of class **Incrementer**.

a. `private synchronized void increment() { counter++; }`

b. `private static synchronized void increment() { counter++; }` ✔

c.
```
private void increment() {
   synchronized (getClass()) { counter++; }
}
```

d.
```
private void increment() {
   synchronized (this) { counter++; }
```

```
}
```

d.
```
private void increment() {
  synchronized (this) { counter++; }
}
```

(b and c correct)

Frage 2

Bisher nicht beantwortet

Erreichbare Punkte: 2,00

Frage markieren

2 Punkte

```
public class A {
   public static void sm() {System.out.print("1");}
   public void im() {System.out.print("2");}
}
public class B extends A {
   public static void sm() { System.out.print("3"); }
   public void im() { System.out.print("4"); }

   public static void main(String[] args) {
      B b = new B();
      A a = b;
      a.sm();
      a.im();
   }
}
```

Welche Ausgabe liefert das Programm?

Antwort:

Output: "14"

Static methods are resolved at compile-time and are not overridden.
When you call a.im();, it will call the instance method im() of class B because it overrides the method in class A.

Frage 3

Bisher nicht beantwortet

Erreichbare Punkte: 2,00

Frage markieren

2 Punkte

Was ist eine polymorphe Variable?

Polymorphism designates the ability of a variable to refer to objects of different types (instances of different classes)

Overridden methods are also referred to as polymorphic methods.

Variables for object types (reference variables) are polymorphic in Java. "Polymorphic" means something like "multifaceted". For object-oriented programming, this means that a variable can refer to different object types

4 Punkte

```
public abstract class Person {
      String name; ...
      public void ausgeben() { System.out.println("Person: " + name); }
}
public class Student extends Person {
      String name; ...
      public void ausgeben() { System.out.println("Student: " + name); }
}
public class BachelorStudent extends Person {
      String name; ...
      public void ausgeben() { System.out.println("Bachelor: " + name); }
}

public interface Utilities {
      public static void printAll(/* ..... */ list)  {
                  list.stream().forEach(e -> e.ausgeben());
      }
}
```

Der Typ des formalen Parameters list der Methode **printAll(...)** ist so zu wählen, dass ein Objekt vom Typ **List<T>** übergeben werden kann, wobei ***T*** der Typ **Person** oder der Typ einer beliebigen Unterklasse von **Person** sein kann.

Wählen Sie **jede** unten angegebene Option aus, die diesen Kriterien entspricht. Für diese Frage erhalten Sie nur Punkte, wenn **alle** Optionen, die korrekt sind, ausgewählt werden (all-or-nothing multiple choice).

Der Typ des formalen Parameters list der Methode **printAll(...)** ist so zu wählen, dass ein Objekt vom Typ **List<T>** übergeben werden kann, wobei ***T*** der Typ **Person** oder der Typ einer beliebigen Unterklasse von **Person** sein kann.

Wählen Sie **jede** unten angegebene Option aus, die diesen Kriterien entspricht. Für diese Frage erhalten Sie nur Punkte, wenn **alle** Optionen, die korrekt sind, ausgewählt werden (all-or-nothing multiple choice).

- ☐ a. List<? super Person>
- ☐ b. List<? extends Person>
- ☐ c. List<Object>
- ☐ d. List<Person>

b
(c wäre zu allgemein)

6 Punkte (0,5 Punkte pro korrekte Antwort)

Gegeben sind folgende Java-Klassen und Anweisungen:

```
class X {}
class Y extends X {}
class Z extends Y {}
...
X x = new X();
Y y = new Y();
Z z = new Z();

X[] ax = new X[10];
Y[] ay = new Y[20];

List<X> lx = new ArrayList<X>();
List<Y> ly = new ArrayList<Y>();
List<?> lw;
List<Object> lo;

List<? super X> lsX;
List<? super Y> lsY;
```

Geben Sie für jedes der folgende Codestücke an, ob es sich um korrektes Java handelt.

- lsY = lsX; — Auswahlen ...
- lx = ly; — Auswählen ...
- lo = lw; — Auswählen ...
- y = x; — Auswählen ...
- x = z; — Auswählen ...

- lsY = lsX; — Korrekt
- lx = ly; — Inkorrekt
- lo = lw; — Inkorrekt
- y = x; — Inkorrekt
- x = z; — Korrekt
- lw = lx; — Korrekt

6 Punkte (1 Punkte pro markierter Stelle)

Zeigen Sie, wie durch Verwendung des Interface **Motor** mittels *Dependency Injection/Constructor Injection* die Abhängigkeit der Klasse **Fahrzeug** von der Klasse **BenzinMotor** aufgelöst werden kann. Die Klasse **SetupFahrzeug** soll dabei die *Dependency Injection* durchführen.

Zeigen Sie wie für **fzg1** die Klasse **BenzinMotor** und für **fzg2** die Klasse **DieselMotor** verwendet werden kann. Ändern/Ersetzen/Erweitern Sie dazu den Code an den markierten Stellen.

```
interface Motor { int getLeistung(); }

class Fahrzeug {
   BenzinMotor m = new BenzinMotor();                    // 1
                                                         // 2
}


class BenzinMotor                                        // 3
{
   public int getLeistung() { return 100; }
}

class DieselMotor                                        // 4
{
   public int getLeistung()  { return 88; }
}

public class SetupFahrzeug {
   public static void main(String[] args) {
      Fahrzeug fzg1 = new Fahrzeug();          // 5
      Fahrzeug fzg2 = new Fahrzeug();          // 6

      System.out.println("Server fzg1: " + fzg1.m.getLeistung());
      System.out.println("Server fzg2: " + fzg2.m.getLeistung());
}
```

```
interface Motor { int getLeistung(); }


class Fahrzeug {

    BenzinMotor m = new BenzinMotor();                    // 1
    private Motor motor;                                  // 2
    public Fahrzeug(Motor m) { this.motor=m; }
}   public int getLeistung() { return motor.getLeistung(); }


class BenzinMotor implements Motor                        // 3
{
    public int getLeistung() { return 100; }
}


class DieselMotor implements Motor                        // 4
{
    public int getLeistung()  { return 88; }
}


public class SetupFahrzeug {

    public static void main(String[] args) {
        Fahrzeug fzg1 = new Fahrzeug();  benzin            // 5
        Fahrzeug fzg2 = new Fahrzeug();  diesel            // 6
        Motor benzin = new BenzinMotor();
        Motor diesel = new DieselMotor();
        System.out.println("Server fzg1: " + fzg1.m.getLeistung());

        System.out.println("Server fzg2: " + fzg2.m.getLeistung());
}
```

```
public class ParameterTransfer {
   static void setRadius(double r, Circle c) {
      c.radius = r; r = 0; c = null;
   }
   public static void main(String[] args) {
      double maxr = 100.0;
      Circle c    = new Circle(1.0,1.0,1.0);
      setRadius(maxr, c);
      System.out.println("maxr = " + maxr);
      System.out.println("c.radius = " + c.radius);
   }
}
```

```
> java ParameterTransfer
maxr = 100.0
c.radius = 100.0
```

r=5, c.r=5

LAMDA EXPRESSIONS

```
public class LambdaExpressions {
   public static void main(String[] args) {
      List<Person> personList = /* deine Liste hier initialisieren */;

      // Lambda-Ausdruck für Personen älter als 25
      System.out.println("Personen älter als 25:");
      printPersons(personList, p -> p.getAge() > 25);

      // Lambda-Ausdruck für Studenten jünger als 30
      System.out.println("\nStudenten jünger als 30:");
      printPersons(personList, p -> p instanceof Student && p.getAge() < 30);

      // Lambda-Ausdruck für Arbeiter mit Gehalt über 2000
      System.out.println("\nArbeiter mit Gehalt über 2000:");
      printPersons(personList, p -> p instanceof Worker && ((Worker) p).getSalary() > 2000);
   }

   // Generische Methode zum Filtern und Ausgeben von Personen
   private static void filterAndPrint(List<Person> list, FilterCondition condition) {
      for (Person person : list) {
         if (condition.test(person)) {
            System.out.println(person);
         }
      }
   }

   // Funktionales Interface für Filterbedingungen
   @FunctionalInterface
   interface FilterCondition {
      boolean test(Person p);
   }
}

class Person {
   // Implementiere hier die Klasse Person
```

```
class Person {
   // Implementiere hier die Klasse Person
}

class Student extends Person {
   // Implementiere hier die Klasse Student
}

class Worker extends Person {
   // Implementiere hier die Klasse Worker
}
```