University of Zurich[UZH]

s.e.a.l. software evolution & architecture lab

Informatik I – EProg HS12

Zwischentest II

Allgemeine Hinweise:

- Jede Aufgabe muss vollständig bearbeitet werden, damit Sie die volle Punktzahl erhalten. Insgesamt können **60 Punkte** erreicht werden.
- Die Bearbeitungszeit beträgt **60 Minuten**.
- Unterlagen sind grundsätzlich **keine** erlaubt. Als Ausnahme gilt, falls Sie nicht deutscher Muttersprache sind, ein entsprechendes Wörterbuch.
- Schreiben Sie Ihren **Namen** und Ihre **Matrikelnummer** bitte in die dafür vorgesehenen Kästchen am Ende **jeder** Seite.
- Die Verwendung unerlaubter Hilfsmittel oder das Abschreiben von eine(r/m) Komiliton(in / en) hat die sofortige Abgabe und das Nichtbestehen der Veranstaltung zur Folge. Zudem ist mit einem Disziplinarverfahren zu rechnen.
- Eine englische Version dieses Zwischentestes ist auf Nachfrage verfügbar. Bei Differenzen zwischen der deutschen und der englischen Version dieses Zwischentestes ist die deutsche Version massgebend.

Ich bestätige mit meiner Unterschrift, dass ich:

- die obenstehenden Hinweise gelesen und verstanden habe.
- die Prüfung unter zumutbaren Bedingungen geschrieben habe.

Unterschrift: _____ Datum: _____

Punkte
/60

1 Aufgabe: Multiple Choice — 10 Punkte

Kreuzen Sie Zutreffendes an. Zu jeder Frage können mehrere Antworten richtig sein. Bitte beachten Sie, dass Ihnen für jedes falsch gesetzte Kreuz innerhalb einer Teilaufgabe gleich viele Punkte abgezogen werden, wie Sie für ein korrektes Ankreuzen erhalten. Negative Punktzahlen ergeben null Punkte für die betreffende Frage.

1. Richtige Aussagen sind anzukreuzen: **2 Punkte**
   - ☐ Ein String ist kein primitiver Datentyp.
   - ☐ *Immutable* bezieht sich darauf, dass eine Variable vom Typ String nie `null` sein kann.
   - ☐ Mit dem Plus-Operator(+) können Strings konkateniert werden.
   - ☐ Bei jeder Operation auf einem String wird der gespeicherte String verändert.

   Lösung:

   - ☒ Ein String ist kein primitiver Datentyp.
   - ☐ Immutable bezieht sich darauf, dass eine Variable vom Typ String nie `null` sein kann.
   - ☒ Mit dem Plus-Operator(+) können Strings konkateniert werden.
   - ☐ Bei jeder Operation auf einem String wird der gespeicherte String verändert.

2. Richtige Aussagen sind anzukreuzen: **2 Punkte**
   - ☐ ”X implements Y” ist nur dann korrekt, wenn X ein Klasse ist und Y ein Interface.
   - ☐ ”X implements Y” ist korrekt, wenn X und Y entweder beides Klassen oder beides Interfaces sind.
   - ☐ In Java kann eine Klasse direkt von mehreren anderen Klassen erben.
   - ☐ In Java kann eine Klasse direkt mehrere Interfaces implementieren.

   Lösung:

   - ☒ ”X implements Y” ist nur dann korrekt, wenn X ein Klasse ist und Y ein Interface.
   - ☐ ”X implements Y” ist korrekt, wenn X und Y entweder beides Klassen oder beides Interfaces sind.
   - ☐ In Java kann eine Klasse direkt von mehreren anderen Klassen erben.
   - ☒ In Java kann eine Klasse direkt mehrere Interfaces implementieren.

3. Richtige Aussagen sind anzukreuzen: **2 Punkte**
   - ☐ Der Modifikator `final` beeinflusst die Vererbungshierarchie nicht.
   - ☐ Eine Subklasse kann auf `private` wie auch auf `public` Variablen der Superklasse zugreifen.
   - ☐ In Java kann eine Superklasse in mehreren Subklassen erweitert werden.
   - ☐ Eine Subklasse besitzt automatisch die Methoden der Superklasse, sofern diese `protected` oder `public` definiert sind.

   Lösung:

   - ☐ Der Modifikator `final` beeinflusst die Vererbungshierarchie nicht.
   - ☐ Eine Subklasse kann auf `private` wie auch auf `public` Variablen der Superklasse zugreifen.
   - ☒ In Java kann eine Superklasse in mehreren Subklassen erweitert werden.
   - ☒ Eine Subklasse besitzt automatisch die Methoden der Superklasse, sofern diese `protected` oder `public` definiert sind.

4. Richtige Aussagen sind anzukreuzen: **2 Punkte**
   - ☐ Wenn eine Klasse `abstract` ist, darf sie nicht als Superklasse verwendet werden.
   - ☐ Eine abstrakte Klasse darf auch Implementierungen enthalten, ein Interface hingegen nicht.
   - ☐ Sobald eine Klasse als `abstract` definiert wird, müssen auch alle Methoden dieser Klasse als `abstract` definiert werden.
   - ☐ Eine abstrakte Klasse kann einen oder mehrere Konstruktor(en) definieren.

   Lösung:

   - ☐ Wenn eine Klasse `abstract` ist, darf sie nicht als Superklasse verwendet werden.
   - ☒ Eine abstrakte Klasse darf auch Implementierungen enthalten, ein Interface hingegen nicht.
   - ☐ Sobald eine Klasse als `abstract` definiert wird, müssen auch alle Methoden dieser Klasse als `abstract` definiert werden.
   - ☒ Eine abstrakte Klasse kann einen oder mehrere Konstruktor(en) definieren.

5. Richtige Aussagen sind anzukreuzen: **2 Punkte**
   - ☐ Ein Array ist ein primitiver Datentyp.
   - ☐ Um auf den letzten Wert in einem Array *a* zuzugreifen kann der Index *(a.length - 1)* verwendet werden.
   - ☐ Ein Array besitzt eine feste Grösse.
   - ☐ Um auf Werte in einem Array vom Typ `double` zuzugreifen, muss der Index ebenfalls vom Typ `double` sein.

   Lösung:

   - ☐ Ein Array ist ein primitiver Datentyp.
   - ☒ Um auf den letzten Wert in einem Array *a* zuzugreifen kann der Index *(a.length - 1)* verwendet werden.
   - ☒ Ein Array besitzt eine feste Grösse.
   - ☐ Um auf Werte in einem Array vom Typ `double` zuzugreifen, muss der Index ebenfalls vom Typ `double` sein.

2 Aufgabe: Statischer und dynamischer Typ (6 Punkte)

Gegeben seien folgendes Interface und folgende Klasse:

```java
public interface Z {
   public void sayMyName();
}
```

```java
public class X implements Z{

   public static void print(String string) {
      System.out.println(string);
   }

   public void sayMyName() {
      System.out.println("My name is Nobody.");
   }
}
```

Kreuzen Sie jeweils an, welche der folgenden Statements in einem TestDriver in der main-Methode verwendet werden könnten, ohne einen Syntaxfehler zu verursachen oder eine Exception zur Laufzeit zu werfen. Schreiben Sie allfälligen Konsolen-Output der Statements daneben.

```java
X a = new Z();
a.print("Hi!");
```

☐ Zulässig ☐ Unzulässig Output:

```java
X b = new X();
X.print("Hello World");
```

☐ Zulässig ☐ Unzulässig Output:

```java
Z c = new X();
c.sayMyName();
```

☐ Zulässig ☐ Unzulässig Output:

Lösung:

☐ Zulässig	☒ Unzulässig // 1P	Output: // 1P
☒ Zulässig // 1P	☐ Unzulässig	Output: "Hello World" // 1P
☒ Zulässig // 1P	☐ Unzulässig	Output: "My name is Nobody." // 1P

3 Aufgabe: Rekursion (4 Punkte)

Gegeben sei folgende Methode:

```
1 public static int foo(int m, int n){
2    if(n == 0)
3       return m;
4    else if(n > m)
5       return foo(n, m);
6    else
7       return foo(n, m%n);
8 }
```

1. Was liefert die Methode für `foo(6, 21)` zurück? (2 Punkte)

2. Was berechnet die Methode? (2 Punkte)

Lösung:

1. Die Funktion `foo(6,21)` liefert *3* zurück.
2. Die Funktion berechnet den *grössten gemeinsamen Teiler (ggT)*.

4 Aufgabe: Implementierung (40 Punkte)

4.1 Aufgabenstellung

Bilden Sie den nachfolgenden Sachverhalt auf Interfaces, (wo angebracht abstrakte) Klassen, Methoden, Attribute und Objekte ab. Befolgen Sie dazu unbedingt auch die nachfolgenden Implementierungshinweise.

4.2 Problemstellung

Ein soziales Netzwerk besteht aus vielen Personen. Ihre Freunde (*Friend*) sind dabei entweder Familienmitglieder (*FamilyMember*) oder Mitarbeiter (*CoWorker*). Jeder Ihrer Freunde hat einen Namen (*name*) und eine Telefonnummer (*number*). Mit Ihren Familienmitgliedern sind sie besser bekannt und deshalb wissen Sie auch ihr Alter (*age*). Neben Ihren Freunden gibt es im sozialen Netzwerk auch prominente Persönlichkeiten (*Celebrity*), wie beispielsweise Steve Jobs, von denen nur der Name (*name*) bekannt ist.

Im sozialen Netzwerk gibt es die Möglichkeit, Fan von einer bekannten Persönlichkeit zu werden, um von Zeit zu Zeit eine neue Statusmeldung von dieser Person zu erhalten. Eine bekannte Persönlichkeit kann eine unbegrenzte Menge an Fans haben.

Implementierungshinweise/-vorschriften

- Gehen Sie bei der Implementierung von einer abstrakten Klasse *Friend* aus, und lassen Sie die Klassen *FamilyMember* und *CoWorker* von dieser abstrakten Klasse erben. Gleichzeitig sollen diese beide Klassen das folgende Interface implementieren:

```
public interface Fan {
  public void receiveStatusUpdate(Celebrity celebrity, String status);
}
```

- Wenn sich ein Freund bei einer bekannten Persönlichkeit als Fan anmeldet, soll letztere eine kurze Information der folgenden Art auf den Bildschirm ausgeben: *"Ich habe jetzt einen Fan mehr!"*.
- Wenn ein Freund kein Fan mehr ist, soll diese Person künftig keine Status-Updates von der bekannten Persönlichkeit mehr erhalten. Zusätzlich soll folgende Nachricht auf der Konsole ausgegeben werden: *"Ich habe jetzt einen Fan weniger!"*.
- Wenn ein Mitarbeiter einen Status-Update erhält und die berühmte Persönlichkeit insgesamt mehr als einen Fan hat, so wählt der Mitarbeiter zufällig zwei andere Fans aus und teilt den Status-Update mit ihnen. Dies macht er aber nur, wenn nicht er selbst der zufällig ausgewählte Fan ist. Zusätzlich wird auf dem Bildschirm folgende Meldung ausgegeben: *[NAME] hat eine neue Statusnachricht von [NAME] bekommen!"*
- Wenn ein Familienmitglied einen Status-Update erhält, wird das weitere Verhalten aufgrund des Alters des Familienmitglieds entschieden. Wenn das Familienmitglied jünger als 10 Jahre ist, oder älter als 40, dann wird folgende Meldung auf den Bildschirm ausgegeben: *[NAME] hat eine neue Statusnachricht von [NAME] bekommen!"*. Andernfalls beschliesst das Familienmitglied nicht auf den Status-Update zu reagieren..
- Implementieren Sie einen *TestDriver* und erzeugen Sie in diesem ein *FamilyMember*-Objekt, ein *CoWorker*-Objekt, sowie ein *Celebrity*-Objekt. Lassen Sie anschliessend das Familienmitglied und den Mitarbeiter Fan der berühmten Persönlichkeit werden. Anschliessend soll die Berühmtheit 100 Status-Updates verschicken.

- Überschreiben Sie die *toString*-Methode in der Klasse *Friend*, so dass diese den Namen der Person zurückgibt.
- Beachten Sie folgenden Auszug aus der Java API zur Methode *nextInt* der Klasse *Random*:

`public int` **`nextInt`**`(int n)`

Returns a pseudorandom, uniformly distributed `int` value between 0 (inclusive) and the specified value (exclusive), drawn from this random number generator's sequence. The general contract of `nextInt` is that one `int` value in the specified range is pseudorandomly generated and returned.

Parameters:
`n` - the bound on the random number to be returned. Must be positive.

Returns:
the next pseudorandom, uniformly distributed `int` value between `0` (inclusive) and `n` (exclusive) from this random number generator's sequence

Throws:
`IllegalArgumentException` - if n is not positive

Since:
1.2

Hier ist Platz für Ihre Lösung zu Aufgabe 4

Hier ist Platz für Ihre Lösung zu Aufgabe 4

Lösung:

```java
import java.util.Random;

public class CoWorker extends Friend implements Fan {

   public CoWorker(String name, String number) {
      super(name, number);
   }

   private static final Random RANDOM = new Random();

   @Override
   public void receiveStatusUpdate(Celebrity vip, String status) {
      System.out.println(getName() + " received a new status from " + vip.
          getName() + ": " + status);

      Fan[] fans = vip.getFans();
      if (fans.length >= 2) {
         for (int i = 0; i < 2; i++) {
            Fan fan = fans[RANDOM.nextInt(2)];
            if (!fan.equals(this)) {
               fan.receiveStatusUpdate(vip, status);
            }
         }
      }
   }

}
```

```java
public class FamilyMember extends Friend implements Fan {

   private int age;

   public FamilyMember(String name, String number, int age) {
      super(name, number);
      this.age = age;
   }

   @Override
   public void receiveStatusUpdate(Celebrity vip, String status) {
      if (age < 10 || age > 40) {
         System.out.println(getName() + " received a new status from " +
             vip.getName() + ": " + status);
      }
   }

```

```
}
```

```
public interface Fan {

   public void receiveStatusUpdate(Celebrity vip, String status);

}
```

```
public abstract class Friend {

   private String name;
   private String number;

   public Friend(String name, String number) {
      this.name = name;
      this.number = number;
   }

   @Override
   public String toString() {
      return "Friend: " + name;
   }

   public String getName() {
      return name;
   }

}
```

```
public class SocialNetwork {

   public static void main(String[] args) {
      CoWorker hans = new CoWorker("Hans", "123456789");
      FamilyMember hugo = new FamilyMember("Hugo", "123456789", 50);

      Celebrity steve = new Celebrity("Steve Jobs");
      steve.addFan(hans);
      steve.addFan(hugo);

      for (int i = 0; i < 100; i++) {
         steve.updateStatus("Test status: " + i);
      }
   }

}
```

```
public class Celebrity {

```

```
3     private String name;
4
5     private Fan[] fans = new Fan[0];
6
7     public Celebrity(String name) {
8        this.name = name;
9     }
10
11    public void removeFan(Fan fan) {
12       Fan[] temp = new Fan[fans.length - 1];
13
14       int fanCounter = 0;
15       for (Fan f : fans) {
16          if (!fan.equals(f)) {
17             temp[fanCounter++] = f;
18          }
19       }
20       fans = temp;
21       System.out.println("Ich habe jetzt einen Fan weniger");
22    }
23
24    public void addFan(Fan fan) {
25       Fan[] temp = new Fan[fans.length + 1];
26       int fanCounter = 0;
27       for (Fan f : fans) {
28          temp[fanCounter++] = f;
29       }
30       temp[fans.length] = fan;
31       fans = temp;
32       System.out.println("Ich habe jetzt einen Fan mehr!");
33    }
34
35    public void updateStatus(String status) {
36       for (Fan fan : fans) {
37          fan.receiveStatusUpdate(this, status);
38       }
39    }
40
41    public Fan[] getFans() {
42       return fans;
43    }
44
45    public String getName() {
46       return name;
47    }
48
49 }
```