Lernerfolgskontrolle

8 Aufgaben

100 Punkte

Version 3.0

Aufgabe 1: (12 Punkte, je 2 Punkte)

Wahr oder falsch? Markieren Sie jede der Aussage eindeutig mit “Wahr” oder “Falsch”.

1. Ein MVC-Programm besteht aus Model, View und Computer.
2. Innere Klassen können auf die privaten Variablen der äußeren Klasse zugreifen.
3. Die Größe einer `ArrayList` kann verändert werden.
4. Ein HTTP-Server kann in Java ganz einfach mit der Klasse `Socket` implementiert werden.
5. Enums können Methoden enthalten.
6. Um einen neuen Thread zu starten müssen wir auf jeden Fall eine Klasse schreiben, die von der Klasse `Thread` erbt.

Aufgabe 2: (14 Punkte, je 2 Punkte)

Vervollständigen Sie diesen Java-Code, der eine Zeile an den Server sendet und die Antwort auf der Konsole ausgibt.

```
try (Socket _________________ = new Socket("localhost", 1114);

   _________________ out = new PrintStream(socket.getOutputStream());

   Scanner in = new _________________(socket.getInputStream())) {

   _________________.println("Hallo Server!");

   String line = _________________.nextLine();

   System.out.println("Der Server sagt: " + _________________);

} catch (IOException _________________) {
   e.printStackTrace();
}
```

Aufgabe 3: (10 Punkte)

Auf der folgenden Seite ist ein Swing-Programm abgebildet. Markieren Sie die mit ?____? markierten Stellen, ob sie zur GUI (View) oder zum Spiel (Model) gehören, oder ob sie der Verbindung zwischen beiden dienen (Controller).

```
public class Ratespiel extends JFrame {

   private static final String SECRET = "secret";     ?_____________?

   private JTextField textEins = new JTextField(15);

   private JTextField textZwei = new JTextField(15);  ?_____________?

   private int fehler;  ?_____________?

   public Ratespiel() {
      setTitle("Ratespiel");
      setDefaultCloseOperation(EXIT_ON_CLOSE);
      setSize(250, 100);
      setLayout(new FlowLayout());

      add(new JLabel("Eingabe:"));
      textEins.addActionListener(this::enter);
      add(textEins);

      add(new JLabel("Ausgabe:"));   ?_____________?
      add(textZwei);

      fehler = 0;      ?_____________?
   }

   private void enter(ActionEvent event) {
      String text = textEins.getText();  ?_____________?
      String ausgabe;
      if (text.equals(SECRET)) {         ?_____________?
        ausgabe = "Richtig!";
      } else {
         fehler++;
         if (fehler < 3) {            ?_____________?
            ausgabe = "Falsch!";
         } else {
            ausgabe = "Verloren!";       ?_____________?
         }
      }
      textZwei.setText(ausgabe);         ?_____________?
   }

   public static void main(String[] args) {
      new Ratespiel().setVisible(true);
   }
}
```

Aufgabe 4: (10 Punkte)

Schreiben Sie Klasse `RatespielModel`, die das Model abbildet, das sich aus der Analyse in der vorherigen Aufgabe ergibt. Die Klasse wird eine Methode `play` enthalten, die immer aufgerufen wird, wenn der Spieler ein Wort eingegeben hat.

Aufgabe 5: (21 Punkte)

Es geht um die drei Klassen `Schule`, `Klasse` und `Schueler`. Alle drei haben eine Variable `name` vom Typ `String`. Welche durch den Konstruktor initialisiert wird. Eine `Schule` hat mehrere `Klassen` und mehrere `Schueler`. Jeder `Schueler` ist einer bestimmten `Klasse` zugeordnet. Jeder `Schueler` hat eine Methode `getName()` und `getKlassenName()`, die einen `String` mit dem Namen seiner Klasse zurück gibt. Schreiben Sie die beiden Klassen `Klasse` und `Schueler`. Wie sie aus der Klasse `Schule` unten schon sehen können, wird `Schueler` eine Innere Klasse von `Klasse` sein.

```
public class Schule {

  private String name;

  private Klasse[] klassen;

  private Klasse.Schueler[] schueler;

  public Schule(String name) {
    this.name = name;

    klassen = new Klasse[] {
      new Klasse("1a"),
      new Klasse("1b"),
      new Klasse("1c"),
    };

    schueler = new Klasse.Schueler[] {
      klassen[0].new Schueler("Martin"),
      klassen[0].new Schueler("Stefan"),
      klassen[1].new Schueler("Tetje"),
      klassen[2].new Schueler("Cordula"),
    };
  }

  public static void main(String[] args) {
    Schule schule = new Schule("Grundschule");

    for (Klasse.Schueler s : schule.schueler) {
      System.out.println(s.getName()+" in "+s.getKlassenName());
    }
  }
}
```

Platz für Aufgabe 5:

Aufgabe 6: (9 Punkte)

Gegeben ist folgende Tabelle `Person` in einer Datenbank:

Vorname	Nachname	E-Mail
Bob	Collins	bob.collins@yahoo.com
Rebecca	Cabeca	rebecca.cabeca@gmail.com
Anthony	Clark	anthony.clark@gmail.com

Die Methode `getPersonen()` gibt die Tabelle in Form einer `List<Person>` zurück.
Welche Ausgabe hat das folgende Programm?

```
Collection<Person> personen = getPersonen();
for (Person p : personen) {
  String text = p.getEMail() + ": "
            + p.getNachname() + ", "
            + p.getVorname();
  System.out.println(text);
}
```

Welche Ausgabe hat dieses Programm?

Aufgabe 7: (14 Punkte)

Ergänzen Sie folgende deklaration eines Enums so, dass es eine Methode **`int`** `getPunkte()` enthält, welche die Anzahl der Punkte auf der jeweiligen Würfelseite als **`int`** zurück gibt.

```
public enum Wuerfel {

    EINS(_________________), ZWEI(_________________), DREI(_________________),

    VIER(_________________), FUENF(_________________), SECHS(_________________);




}
```

Aufgabe 8: (10 Punkte)

Auf der folgenden Seite sehen Sie die Test-Klasse `ZahlenTest`, welche die Klasse `Zahlen` testet. Vervollständigen Sie die Klasse Zahlen, damit sie den Test besteht. Schreiben Sie eine Lösung, die einen Anspruch darauf hat vollständig zu sein, also auch andere Tests bestehen würde.

Die Aufgabe der Methode `zahlen` ist es die Ziffern, die sie als Parameter bekommt, als Liste von Wörter zurück zu geben. Der Parameter `z`**`iffern`** verhält sich wie ein **`int`**`[]`.

Das Interface `Map` definiert zwei Methode, die hier von Bedeutung sind.

`put(key, value)` speichert den Wert `value` unter dem Schlüssel `key`. Die Methode `get(key)` gibt den unter dem Schlüssel `key` gespeicherten Wert wieder zurück.

Beim Interface `List` ist nur die Methode `add(value)` für die Aufgabe interessant. Sie hängt den Wert `value` am Ende der Liste an.

```
public class ZahlenTest {
    @Test
    public void test735() {
        List<String> actual = new Zahlen().zahlen(7, 3, 5);
        assertEquals("[Sieben, Drei, Fünf]", actual.toString());
    }

    @Test
    public void test123() {
        List<String> actual = new Zahlen().zahlen(1, 2, 3);
        assertEquals("[Eins, Zwei, Drei]", actual.toString());
    }
}


public class Zahlen {

    private Map<Integer, String> zahlen;

    public Zahlen() {
        zahlen = new HashMap<>();
        zahlen.put(1, "Eins"); zahlen.put(2, "Zwei");
        zahlen.put(3, "Drei"); zahlen.put(4, "Vier");
        zahlen.put(5, "Fünf"); zahlen.put(6, "Sechs");
        zahlen.put(7, "Sieben"); zahlen.put(8, "Acht");
        zahlen.put(9, "Neun"); zahlen.put(0, "Zehn");
    }

    public List<String> zahlen(int... ziffern) {









    }
}
```