Begonnen am	Thursday, 25. June 2020, 09:35
Status	Beendet
Beendet am	Thursday, 25. June 2020, 10:05
Verbrauchte Zeit	29 Minuten 56 Sekunden
Bewertung	**226,67** von 300,00 (**76**%)

Information

Bitte beachten Sie:

- 30 Minuten Zeit
- maximal **30 Punkte**. Anmerkung: Die Punkte im Test sind mit **10 skaliert**. Die Bewertungen der Fragen ergeben insgesamt 300 Punkte, die dann durch durch 10 dividiert werden.
- Bitte Fragen genau lesen
- Bei Antworttexten bitte kurze und präzise Antworten geben! Nur die gestellten Fragen beantworten. Antworten, die für die Fragestellung nicht relevant sind, führen zu Punkteabzügen.
- Bei Multiple-Choice-Fragen sind mehrere Antworten möglich. Falsche und fehlende Ankreuzungen führen zu Punkteabzügen.

Frage 1

Richtig

Erreichte Punkte 25,00 von 25,00

Kreuzen Sie an, in welchem Bereich ein mit dem Sichtbarkeitsattribut **protected** markiertes Programmelement sichtbar ist:

Wählen Sie eine oder mehrere Antworten:

- [ ] in Klassen in fremden Packages
- [x] in Klassen im gleichen Package ✔
- [x] in Unterklassen in fremden Packages ✔
- [x] in der Klasse selbst ✔
- [x] in Unterklassen im gleichen Package ✔

Die Antwort ist richtig.

Die richtigen Antworten sind: in der Klasse selbst, in Klassen im gleichen Package , in Unterklassen im gleichen Package , in Unterklassen in fremden Packages

Frage 2

Vollständig

Erreichte Punkte 15,00 von 20,00

Was bedeutet das Attribut final bei Methoden?

final bedeutet, dass die Methode in keiner Klasse überschrieben werden darf. Eine final Methode darf nicht verändert werden

Kommentar:
darf nicht verändert werden ??

Frage 3

Teilweise richtig

Erreichte Punkte 16,67 von 25,00

Bei einer Methode, die eine Methode aus der Basisklasse überschreibt darf:

Wählen Sie eine oder mehrere Antworten:

- [ ] die Methode mehr Exceptions werfen
- [ ] der Rückgabewert allgemeiner sein
- [x] der Rückgabewert spezieller sein ✔
- [ ] die Sichtbarkeit erweitert werden
- [x] die Methode weniger Exceptions werfen ✔

Die Antwort ist teilweise richtig.

Sie haben 2 richtig ausgewählt.
Die richtigen Antworten sind: die Sichtbarkeit erweitert werden, der Rückgabewert spezieller sein, die Methode weniger Exceptions werfen

Frage 4

Richtig

Erreichte Punkte 20,00 von 20,00

Welche Antwort ist richtig?

Wählen Sie eine oder mehrere Antworten:

- [ ] Der dynamische Typ einer Variablen muss allgemeiner als ihr statischer Typ sein!
- [x] Der dynamische Typ einer Variablen kann spezieller als ihr statischer Typ sein! ✔
- [ ] Der dynamische Typ einer Variablen muss gleich ihrem statischen Typ sein!
- [ ] Der dynamische Typ einer Variablen muss spezieller als ihr statischer Typ sein!

Die Antwort ist richtig.

Die richtige Antwort lautet: Der dynamische Typ einer Variablen kann spezieller als ihr statischer Typ sein!

Frage 5

Vollständig

Erreichte Punkte 30,00 von 30,00

Erklären Sie den Unterschied zwischen

- statischen Member-Klassen
- nicht-statischen Member-Klassen

auf Methoden der statischen MemberKlasse wird über die definierende Klasse zugegriffen-> SomeClass.SomeMethode

statische MemberKlassen haben nur zugriff auf statische Variablen der äußeren Klasse

bei den nicht statischen MemberKlassen wird auf die innere Methoden und Variablen über ein Objekt der inneren Klasse zugegriffen

zb.

```
public class A{
  int x = 0;
 B b = new B();
  public int methodA(){
     return  x + b.y;
   }
 class B{
      int y = 0;
     }
}
```

Kommentar:

Frage 6

Falsch

Erreichte Punkte 0,00 von 20,00

Die folgende Operation *copyTo* kopiert die Elemente der Liste *from* in die Liste *to*. Bei der Typangabe für den Parameter *from* wird ein Wildcard verwendet.

```
public static <T> void copyTo(List<?  super T  ✖ >
from, List<T> to) {
  for (T e : from) {
    to.add(e);
  }
}
```

Wie lautet der Typconstraint?

Frage 7

Richtig

Erreichte Punkte 20,00 von 20,00

Was zeichnet Funktionale Interfaces aus?

Wählen Sie eine oder mehrere Antworten:

- [ ] Haben nur Default-Methoden
- [ ] Haben nur eine Methode
- [x] Haben nur eine abstrakte Methode ✔
- [ ] Haben nur statische Methoden

Die Antwort ist richtig.

Die richtige Antwort lautet: Haben nur eine abstrakte Methode

Frage 8

Vollständig

Erreichte Punkte 40,00 von 40,00

Erklären Sie in einer kurzen Beschreibung die Prinzipien von JUnit-Tests.

Junit test sind Schnittstellen Tests und im Sinne von BlackBoxen. Man schaut also nur ob das Ergebnis passt, nicht aber ob die Implementierung korrekt ist. Das innere Strukturverhalten der zu testenden Komponenten muss nicht verstanden werden(da es sich um BlackBox testing handelt)
Es werden einzelne Komponenten eines Systems getestet zb Klassen oder Methoden.
In der Test klasse gibt es verschiedene Testmethoden die mit Annotation(zb @BeforeEach/ @Test etc) gekennzeichent sind.
Bei den eigentlichen Test methoden (@Test) wird mit Hilfe von

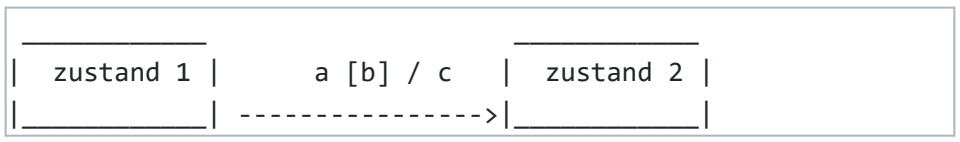
Kommentar:

Frage 9

Falsch

Erreichte Punkte 0,00 von 20,00

Bei UML Zustandsdiagrammen werden Kanten in folgender Form mit drei Teilen *a*, *b* und *c* beschriftet:



Für was stehen die drei Teile?

geht von a ✖ [über b ✖] / in c ✖

Frage 10

Vollständig

Erreichte Punkte 30,00 von 30,00

Erklären Sie was mit den folgenden UML Diagrammen modelliert wird:

- UML Sequenzdiagrammen
- UML Zustandsdiagrammen

Beim Sequenzdiagramm werden die Nachrichten, die daran beteiligten Objekte und die zeitliche Nachrichtenfolge modelliert. Beim Zustandsdiagramm werden die Zustände der Objekte modelliert und was zu einer Zustandsänderung führt.

Kommentar:

Frage 11

Vollständig

Erreichte Punkte 15,00 von 20,00

Was wird beim Moduldeskriptor *module-info.java* mit folgenden beiden Anweisungen spezifiziert?

- exports
- requires

exports zeigt an dass dieses Modul exportiert wird und damit anderen Modulen zur Verfügung gestellt wird.

requires zeigt an dass dieses Modul ein bestimmen Modul braucht, um zu funktionieren

Kommentar:

exports: exportierte Packages

Frage 12

Vollständig

Erreichte Punkte 15,00 von 30,00

Gegeben sei eine generische Klasse *List* mit Methoden *get* und *add* wie folgt:

```
public class List<E> {
  public E get(int i) { ... }
  public void add(E e) { ... }
}
```

Des Weiteren seien Klasse *Person* und Subklasse *Student* gegeben:

```
class Person {...}
class Students extends Person {...}
```

Von dieser Klasse wird eine Instanz mit Elementtyp *Student* erzeugt:

```
List<Student> students = new List<Student>();
```

Begründen Sie, warum eine kovariante Zuweisung wie folgt nicht erlaubt ist?

```
List<Person> persons = students;
```

Eine kovariante Zuweisung ist nicht erlaubt, weil der Rückgabewert dadurch nicht mehr garantiert wäre.
bei Person p = persons.get(0);
könnte es in Folge zu Fehlern kommen da der Rückgabewert nicht mehr Person ist.
um kovariante Zuweisungen in Java zu ermöglichen gibt es Upper Bounded WIldcards(<?extends Person) damit wäre so eine Zuweisung erlaubt

Kommentar:

Begründung falsch, Wildcard richtig

Begründung: weil schreibender Zugriff persons.add nicht typsicher ist.

◄ Klausur 1: Praktischer Teil

Direkt zu:

00.Inhalt ►