Programmiermethodik
3. Klausur

9. 1. 2014

Name	
Matrikelnummer	

Aufgabe	**mögliche Punkte**	**erreichte Punkte**
1	20	
2	16	
3	45	
4	19	
5	20	
Gesamt	**120**	

Aufgabe 1) Objekt-Orientierung und Vererbung (20 Punkte)

1. Welche Zugriffsattribute gibt es in Java, und wie unterscheiden Sie sich? Erläutern Sie anhand eines Beispiels, warum es sinnvoll ist, Zugriffsattribute einzusetzen. **6 Punkte**

   Lösung

2. Erklären Sie die Begriffe Kohäsion und Kopplung. Welche Qualitäten sollte ein gutes Programm haben? **6 Punkte**

   Lösung

3. Betrachten Sie folgende Problemstellung: Sie möchten eine Klasse `Point2D` implementieren, die Punkte in einem zweidimensionalen Koordinatensystem repräsentiert. Um Objekte anzulegen werden kartesische Koordinaten `(double x, double y)` oder Polarkoordinaten `(double r, double phi)` übergeben.

   Erklären Sie, warum es nicht möglich ist, einfach zwei Konstruktoren `Point2D(double x, double y)` und `Point2D(double r, double phi)` zu verwenden. Nennen Sie zwei *sinnvolle* Möglichkeiten, dieses Problem zu lösen. **8 Punkte**

   Lösung

Aufgabe 2) Schnittstellen und Abstrakte Klassen (16 Punkte)

1. Erklären Sie die Unterschiede zwischen einer abstrakten Klasse und einem Interface in Java, sowie die Unterschiede zwischen einer abstrakten und einer konkreten Klasse.

   Skizzieren Sie einen Anwendungsfall, in dem es jeweils besser ist ein Interface bzw. eine abstrakte Klasse zu verwenden. **10 Punkte**

2. Beantworten Sie folgende Fragen kurz in Stichworten.

   (a) Wenn ein Interface von einer abstrakten Klasse erbt, müssen die Methoden implementiert werden? Wenn nicht, warum?

   (b) Kann ein Interface mehrere Interfaces implementieren? Wenn nicht, warum?

   (c) Muss ein Interface einen Konstruktor implementieren? Wenn nicht, warum?

   (d) Kompiliert ein Interface, wenn es keine Methoden und Konstanten enthält? Wenn nicht, warum?

   6 Punkte

Lösung

Aufgabe 3) Ausnahmen, Vererbung, Delegation (45 Punkte)

Betrachten Sie das folgenden vereinfachte `Map`-Interface.

```
public interface Map<K, V> {
    void put(K key, V value);
    V get(K key);
}
```

Bitte beachten Sie in der folgenden Aufgabenstellung, dass viele der Aufgaben unabhängig voneinander gelöst werden können! Lediglich 2(b) und 3(a) hängen von 1 ab.

1. Entwickeln Sie eine Listen-basierte Implementierung des `Map`-Interfaces in einer Klasse `ListMap`, so dass das folgende Test-Programm ohne Fehler compiliert und ausgeführt wird. **20 Punkte**

```
1  public class Test {
2      public static void main(String[] args) {
3          Map<Integer, String> map = new ListMap<>();
4          map.put(2, "zwei");
5          map.put(3, "III");
6          map.put(2, "II");
7          String s = map.get(2);
8          System.out.println(s);
9      }
10 }
```

   - Sie können davon ausgehen, dass sämtliche Schlüssel nicht `null` sind.
   - Falls ein Schlüssel nicht vorhanden ist, sollte `get` den Wert `null` liefern.
   - Zu jedem Schlüssel darf es maximal einen Eintrag in der Liste geben. Ein `put` überschreibt die alte Zuordnung, falls diese existiert.
   - Berücksichtigen Sie das Prinzip der Datenkapselung.

2. Es soll nun sichergestellt werden, dass nur Schlüssel ungleich `null` verwendet werden: Falls `get` oder `put` mit `key = null` aufgerufen werden, sollte keine `NullPointerException` auftreten, sondern eine speziellere Ausnahme geworfen werden. **17 Punkte**

   (a) Definieren Sie eine Ausnahme `KeyNullException`.

   (b) Erstellen Sie eine Klasse `SafeKeyMapInheritance`, die die oben gewünschten Eigenschaften hat, die auf der Klasse `ListMap` basiert, und die dabei das Prinzip der Vererbung verwendet.
   Müssen Sie hierzu die Datenkapselung aufweichen?

   (c) Erstellen Sie eine Klasse `SafeKeyMapDelegate` mit gleicher Funktionalität, die aber das Prinzip der Delegation verwendet (und `ListMap` als Delegat verwendet).

   (d) Man sollte nun im Test-Programm den Aufruf `new ListMap...` in Zeile 3 durch `new SafeKeyMap...` ersetzen können. Müssen Sie dazu noch weitere Änderungen durchführen? Wenn ja, welche?

3. Nun soll unterschieden werden können, ob ein `null` beim Aufruf von `get(key)` bedeutet, dass es keinen Eintrag zu `key` in der `Map` gibt, oder ob der zu `key` gehörige Wert eben `null` ist. Im ersten Fall soll eine `NoSuchKeyException` geworfen werden, wobei Sie davon ausgehen können, dass es die Klasse `NoSuchKeyException` schon gibt und das diese eine Unterklasse von `RuntimeException` ist. **8 Punkte**

(a) Erstellen Sie, wenn möglich, eine Klasse `SafeGetMapInheritance`, die die oben gewünschten Eigenschaften hat, die auf der Klasse `ListMap` basiert, und die dabei das Prinzip der Vererbung verwendet. Wenn es nicht möglich ist, diese Implementierung so durchzuführen, erläutern Sie kurz den Grund.

(b) Lösen Sie dieselbe Aufgabenstellung wie in (b) in einer Klasse `SafeGetMapDelegate`, wobei nun jedoch das Prinzip der Delegation anstelle der Vererbung verwendet werden soll.

Lösung

Aufgabe 4) Generische Programmierung (19 Punkte)

1. Betrachten Sie nachfolgendes Codefragement. Welche Aussage(n) trifft/treffen zu? **4 Punkte**

```
List<  Integer  > list = new ArrayList();
list.add(89);
int i = list.get(0);
System.out.println(i);
```

   (a) Das Codefragment kompiliert ohne Warnung und liefert als Ergebnis 89.

   (b) Das Codefragment kompiliert mit Warnung und liefert als Ergebnis 89.

   (c) Das Codefragment kompiliert nicht.

   (d) Das Codefragment führt zu einer Runtime Exception.

2. Was ist/sind (ein) zulässige(r) generische(r) Parameter in Java? **3 Punkte**

   (a) Klassen

   (b) Primitive Typen

   (c) Interfaces

   (d) Literale

3. Ist es in Java erlaubt, eine `ArrayList` ohne entsprechenden Typparameter anzulegen? **3 Punkte**

4. Was ist die Ausgabe des nachfolgenden Codefragments? **4 Punkte**

```
public static void before() {
    Set<String> set = new TreeSet<String>();
    set.add("2");
    set.add(3);
    set.add("1");
    Iterator it = set.iterator();
    while (it.hasNext())
        System.out.print(it.next() + " ");
}
```

5. Betrachten Sie nachfolgendes Codefragment. Erlaubt Java die Verwendung solch *generischer* Exceptions? Begründen sie Ihre Antwort! **5 Punkte**

```
try{
    ...
}catch(MyException<Byte>){
    ...
}catch(MyException<Integer>){
    ...
}
```

Lösung

Aufgabe 5) Modellierung (20 Punkte)

1. Es soll ein Prozessmanagementtool implementiert werden (In Form eines gerichteten Graphen). Ein Prozess besteht aus mehreren Arbeitsschritten. Ein Arbeitsschritt kann entweder eine einzelne Aktivität sein, oder wieder selber ein (Unter-)Prozess. Prozesse und Aktivitäten haben einen Namen. Aktivitäten haben dazu noch eine Dauer, sowie Kosten. Um einzelne Arbeitsschritte verbinden zu können, braucht es zwischen den Arbeitsschritten Übergänge. Da die Übergänge auch Verzweigungen oder Synchronisationen sein können, gibt es hier 4 Typen (STRAIGHT, AND, OR, SYNC). In dem Tool gibt es ein Nachrichtencenter, das den Benutzer informiert, wenn eine Aktivität abgeschlossen ist. Das Nachrichtencenter gibt es genau einmal. Weiters informieren nicht alle Aktivitäten das Nachrichtencenter, sondern nur die, die eine Benutzereingabe erfordern. **15 Punkte**
2. Erläutern Sie, welche Designpattern Sie verwendet haben, und wie Sie diese angewendet haben. **5 Punkte**

Lösung