

Programmiermethodik
3. Klausur **Lösung**

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 private (nur gleiche Klasse), default (gleiches Paket), protected (alle Subklassen), public (alle Klassen); je 1 Punkt

Beispiel: mit Zugriffsattributen kann zwischen internen/externen Variablen/Methoden unterschieden werden. -> Invarianten können garantiert werden, Überprüfungen fallen weg, etc. 2 Punkte

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

Lösung Kohäsion: Zusammenhalt einer Klasse. Eine Klasse für eine Aufgabe, bzw. auch eine Methode für eine Aufgabe. 2 Punkte Koppelung: Wie stark sind die Abhängigkeiten zwischen Klassen? 2 Punkte Guter Code: Hohe Kohäsion, Lose Koppelung. 2 Punkte

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 Die Konstruktoren hätten die gleiche Signatur, was liefert dann ein Aufruf `new Point2D(1,1)`? 2 Punkte Möglichkeit 1: Ein Objekt `Point2D` anlegen, und erst dann mit Settern die Werte entsprechend setzen (bzw. mit Methoden `setCoordinatesPolar`/`setCoordinatesCartesian` o.ä.) Möglichkeit 2: Mit Hilfe einer statischen Factory Method, die ein Objekt vom Typ `Point2D` zurückgibt. Beispiel `Point2D.cartesian(x, y)` oder `Point2D.polar(r, phi)` Möglichkeit 3: boolescher Parameter zum Konstruktor hinzufügen, der entscheidet, ob die Doubles polare oder kartesische Koordinaten sind. je 3 Punkte

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

1. Abstrakte Klassen können Attribute haben und Methoden implementieren, während Interfaces nur Konstante und abstrakte Methoden definieren können.

Abstrakte Klassen werden mit dem Schlüsselwort `abstract` markiert, abstrakte Klassen können abstrakte Methoden beinhalten, von Interfaces geerbte Methoden müssen nicht implementiert werden, man kann nicht direkt von einer abstrakten Klasse eine Instanz erzeugen (nur von konkreten Unterklassen der abstrakten Klasse)

Interfaces werden dort benötigt, wo sonst Mehrfachvererbung gebraucht wird, während man abstrakte Klassen besser dort einsetzt, wo viele Subklassen die gleiche Funktionalität implementieren und somit Code Reuse sinnvoll ist.

2. (a) Nein. Interfaces können nicht von abstrakten Klassen erben.
(b) Ja.
(c) Nein. Es geht sogar nicht, da es keine Instanzen eines Interfaces gibt.
(d) Ja.

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 dass 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 (a) in einer Klasse `SafeGetMapDelegate`, wobei nun jedoch das Prinzip der Delegation anstelle der Vererbung verwendet werden soll.

Lösung

1.

```

public class ListMap<K, V> implements Map<K, V> {

    private Node head = null;

    public void put(K key, V value) {
        Node n = this.find(this.head, key);
        if (n == null) {
            this.head = new Node(key, value, this.head);
        } else {
            n.value = value;
        }
    }

    public V get(K key) {
        Node n = this.find(this.head, key);
        if (n == null) {
            return null;
        } else {
            return n.value;
        }
    }

    private Node find(Node node, K key) {
        while (node != null) {
            if (node.key.equals(key)) {
                return node;
            }
        }
        return null;
    }

    private class Node {
        K key;
        V value;
        Node next;

        private Node(K key, V value, Node next) {
            this.key = key;
            this.value = value;
            this.next = next;
        }
    }
}

```

2. (a)

```

public class KeyNullException extends RuntimeException {}

```

(b)

```

public class SafeKeyMapInheritance<K, V> extends ListMap<K, V> {

    protected Node find(Node n, K key) {
        if (key == null) {
            throw new KeyNullException();
        } else {

```

```

        return super.find(n, key);
    }
}

```

Damit man `find` weiterverwenden und überschreiben kann, muss man für `Node` und `find` den Zugriff von `private` auf `protected` abschwächen.

(c)

```

public class SafeKeyMapDelegate<K, V> implements Map<K, V> {

    private final Map<K, V> map;

    public SafeKeyMapDelegate() {
        this.map = new ListMap<>();
    }

    public void put(K key, V value) {
        if (key == null) {
            throw new KeyNullException();
        } else {
            this.map.put(key, value);
        }
    }

    public V get(K key) {
        if (key == null) {
            throw new KeyNullException();
        }
        return this.map.get(key);
    }
}

```

(d) Da wir unsere Ausnahme als `RuntimeException` definiert haben, brauchen wir das Test-Programm und auch das Interface nicht ändern.

3. (a)

```

public class SafeGetMapInheritance<K, V> extends ListMap<K, V> {

    public V get(K key) {
        Node n = this.find(this.head, key);
        if (n == null) {
            throw new NoSuchKeyException();
        } else {
            return n.value;
        }
    }
}

```

Wie in Aufgabe 2(b) muss man den Zugriff von `private` auf `protected` abschwächen, hier insbesondere für `head` (und wie zuvor auch für `Node`).

(b) Diese Aufgabe ist so nicht mittels Delegation lösbar, wenn man davon ausgeht, dass man das öffentliche Interface von `ListMap` nicht erweitert. Da man bei der Delegation nur das Ergebnis von `get` abfragen kann, ist es an dieser Stelle nicht mehr entscheidbar, welcher der Fälle vorliegt.

Aufgabe 4) Generische Programmierung (19 Punkte)

1. Betrachten Sie nachfolgendes Codefragment. 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 solcher *generischer* Exceptions? Begründen sie Ihre Antwort! **5 Punkte**

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


Lösung

1. (b)
2. (a) + (c)
3. Ja
4. Kompilierfehler
5. Nein, Java erlaubt es nicht. Begründung: JVM könnte, dank Type-Erasure, zur Laufzeit nicht unterscheiden, ob es sich um eine `MyException<Byte>` oder `MyException<Integer>` handelt. Für die JVM sind beides einfach nur Instanzen von `MyException`.

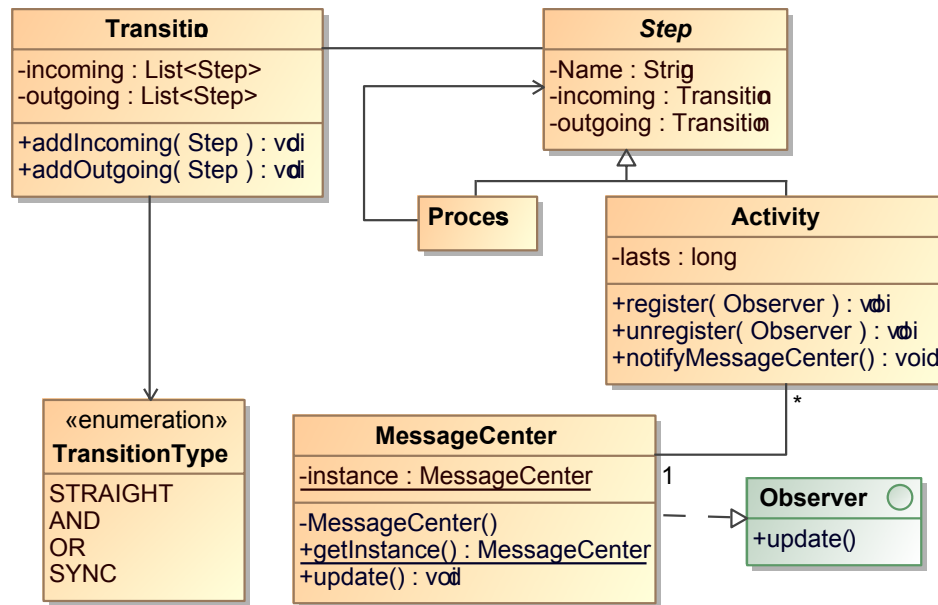


Abbildung 1: Lösung

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

1. Siehe Abbildung 1
2. Identifizierte Pattern:
 - (a) Composite Pattern Pattern: Das unterschiedliche Aussehen der Items
 - (b) Observer Pattern (Push): Nachrichtencenter
 - (c) Singleton Pattern: Nachrichtencenter