Prof. Dr. Thomas Breuel
Dipl.-Inf. Patrick Michel (AG ST)
Dipl.-Ing. Ludwig Schmidt-Hackenberg

TU Kaiserslautern
Fachbereich Informatik
AG Bildverstehen und Mustererkennung

Lösungshinweise/-vorschläge zum Übungsblatt 14: Software-Entwicklung 1 (WS 2010/11)

Die Hinweise und Vorschläge in diesem Dokument sollen der Lösungsfindung dienen und erheben demnach weder Anspruch auf Vollständigkeit noch Korrektheit. Sollten Sie Fehler finden, würden wir uns freuen, wenn Sie uns diese mitteilen. (Kontaktdaten finden Sie auf unserer Webpräsenz.)

Aufgabe 1 Attribute und Methoden

Betrachten Sie das folgende Codestück und beantworten Sie dazu die nachfolgenden Fragen:

```
1  class A {
2      int i = 1;
3      boolean typtest()  { return ( this instanceof A ); }
4      void f()           { g(); h(); }
5
6      public    void g() { System.out.println("g in A"); }
7      private   void h() { System.out.println("h in A"); }
8  }
9
10 class B extends A {
11     int i = 2;
12     boolean typtest()  { return super.typtest(); }
13
14     public    void g() { System.out.println("g in B"); }
15     protected void h() { System.out.println("h in B"); }
16 }
17
18 class C extends B {
19     int i = 3;
20
21     public    void g() { System.out.println("g in C"); }
22     public    void h() { System.out.println("h in C"); }
23 }
```

a) Wie viele objektlokale Variablen des Namens `i` besitzt ein Objekt der Klasse `C`?

Objekte der Klasse `C` besitzen drei objektlokale Variablen des Namens `i`: das erste wird in `A` deklariert, das zweite in `B` und das dritte in `C`.

b) Implementieren Sie eine Methode `sum_i` in `C`, die die Summe der in diesen objektlokalen Variablen gespeicherten Werte zurückliefert.

Es ist nicht möglich, aus einer Methode in `C` heraus direkt die objektlokale Variablen `i` aus der Klasse `A` mit einem Aufruf wie **`super.super.i`** anzusprechen.

Daher muss man eine der beiden folgenden Lösungen wählen: Man kann den impliziten Parameter auf den entsprechenden Typ *casten*, also

```
class C extends B { // ...
    int sum_i() { return i + super.i + ((A)this).i; }
}
```

oder man kann entweder in A oder in B eine Methode implementieren, die dieses i liefert, also zum Beispiel:

```
class B extends A { // ...
    int sum_i() { return i + super.i; }
}
class C extends B { // ...
    int sum_i() { return i + super.sum_i(); }
}
```

Wichtig an dieser Aufgabe ist u.a. zu verstehen, dass Attributzugriff statisch gebunden ist, also vom statischen Typ abhängt, wohingegen Methoden (bis auf private) dynamisch gebunden sind. Deshalb funktioniert die Lösung mit *casten*.

Hinweis: Es gibt in Java die Syntax `A.super.i`*, sie erlaubt es jedoch* ***nicht*** *auf das Attribut* `i` *einer* ***eigenen*** *Superklasse* `A` *zuzugreifen. Stattdessen wird mit dieser Syntax auf das Attribut* `i` *der Superklasse einer* ***äußeren*** *Klasse* `A` *zugegriffen. Die Syntax verhält sich also analog zu* `A.this.i`*, welche auf das (evtl. verdeckte) Attribut* `i` *der äußeren Klasse* A *zugreift.*

c) Kann auch in einer Methode, die in der Klasse B deklariert wurde, auf jede dieser objektlokalen Variablen zugegriffen werden? Wenn ja, wie, und wenn nicht, warum nicht?

Ja, es geht. Der Zugriff auf die „eigene" Variable ist kein Problem, die der Superklasse erreicht man mit `super.i`. Für den Zugriff auf die objektlokale Variablen, die in der Subklasse erst deklariert wird, muss man den impliziten Parameter auf C casten, also zum Beispiel: `x = ((C)this).i`

d) Liefert die Methode `typtest()` der Klasse A stets **true**? Welchen *speziellsten* dynamischen Typ kann der implizite Parameter dieser Methode besitzen? Von welcher Klasse ist der implizite Parameter der Methode in Zeile 3 bei Ausführung von `boolean b = new B().typtest();`? Begründen Sie alle Antworten!

Ja, die Methode `typtest` liefert immer **true**, denn "*e* **instanceof** *class*" liefert **true**, wenn *e* ein Ausdruck vom Typ *class* oder ein Subtyp ist.

In diesem Szenario (wenn man unterstellt, es sei abgeschlossen gegen Erweiterungen) kann der speziellste Typ nur C sein. Da diese Klasse aber nicht als *final* deklariert ist, sind auch weitere Subtypenbildungen möglich, die dann einen spezielleren Typ von **this** zur Folge hätten.

In der Beispielzeile ist der implizite Parameter vom Typ B.

e) Was wird jeweils ausgegeben, wenn die Methode f auf je einem Objekt des Typs A, B und C aufgerufen wird? Begründen Sie das Resultat!

Objekt vom Typ A:

Ausgabe:

```
g in A
h in A
```

Gründe: beide Implementierungen der Klasse A werden aufgerufen, was auch zu erwarten war.

Objekt vom Typ B

Ausgabe:

```
g in B
h in A
```

Gründe: g ist **public**, wird also dynamisch gebunden. Somit wird die Implementierung aus B aufgerufen. h ist in A als **private** deklariert, damit wird der Aufruf von A aus statisch gebunden und die dortige Implementierung verwendet. Die Methode h aus B hat eine höhere Sichtbarkeit und ist damit keine Überschreibung, sondern eine neu eingeführte Methode.

Objekt vom Typ C

Ausgabe:

```
g in C
h in A
```

Gründe: Die gleichen, wie bei B.

Aufgabe 2 Java Collection Klassen

Neben den eigentlichen Sprachfeatures zeichnet sich Java durch eine gut ausgestattete Standardbibliothek aus, welche durch eine Vielzahl von Klassen und Schnittstellen Implementierungen vereinfacht, indem auf die zur Verfügung gestellten Bausteine mittels Vererbung, Subtyping usw. zurückgegriffen werden kann.

Die im Paket `java.util` angesiedelten *Collection*-Klassen stellen unter anderem Implementierungen für im Programmieralltag häufig auftretende Datenstrukturen bereit. Die Dokumentation zu diesem und den anderen Bestandteilen des aktuellen JDKs ist unter http://java.sun.com/javase/6/docs/api verfügbar.

a) Betrachten Sie die Klasse `LinkedList`. Wo ist die Implementierung der Methode `containsAll` zu finden?

Die Methode `containsAll` wird von der (abstrakten) Klasse `AbstractCollection` geerbt. Da die Methode in der Klasse `AbstractCollection` nicht als abstrakte Methode definiert wurde, ist die Implementierung dort zu finden.

b) Gruppieren Sie die verschiedenen Implementierungen von Datenstrukturen nach typischen Grundcharakteristika wie beispielsweise *mengenartig* oder *listenartig*. Mit welchen Java-Sprachmitteln werden diese Grundcharakteristika für den Software-Entwickler abgebildet?

„mengenartig“: `HashSet`, `TreeSet`, ...

„listenartig“: `ArrayList`, `LinkedList`, ...

„assoziativ“: `HashMap`, `Hashtable`, ...

...

Ein mögliches Java-Sprachmittel ist die Implementierung entsprechender Interfaces (→ `Map`, `Set`, `List`, ...) oder die Verwendung von Vererbung (→ `Dictionary`, `Hashtable`), wobei die Verwendung von Interfaces im Kontext der *Collection*-Klassen in der Regel die bessere Entscheidung ist.

c) Betrachten Sie die Klasse `HashSet`. Mittels der generischen Methode `toArray` lässt sich ein Array erzeugen, welches die von der `HashSet`-Instanz verwalteten Elemente enthält. Wieso sollte die Methode nicht wie im folgenden Fragment

```
...
Set<String> set = new HashSet<String>();

set.add("A");
set.add("B");
set.add("C");
set.add("E");
set.add("X");

String[] array = set.toArray(new String[3]);
...
```

verwendet werden? Wie sieht ein sinnvoller Aufruf der Methode `toArray` aus?

Das der Methode übergebene Array ist im angegebenen Fall zu klein, um alle Elemente der Datenstruktur aufzunehmen. Somit wird dieses verworfen und ein neues Array muss erzeugt werden. Sinnvoll ist es daher, die passende Größe für das Array zu verwenden, sodass eine erneute Erzeugung vermieden wird.

Normalerweise wird daher ein `toArray`-Aufruf die Form `toArray(new String[set.size()])` besitzen.

Anzumerken ist hier auch, dass es keine Variante dieser Methode ohne Parameter gibt, da ein generisches Array zurückgegeben werden soll, aber die generischen Typinformationen zur Laufzeit nicht mehr zur Verfügung stehen um eins vom passenden Typ zu erzeugen.

d) Beschreiben Sie die eventuell vorhandenen Beziehungen zwischen `Collection`, `HashSet`, `List`, `Vector` und `Stack` durch Anfertigung eines Klassendiagramms.

Unter der URL http://java.sun.com/javase/6/docs/api/java/util/package-tree.html finden Sie eine Übersicht mit allen nötigen Informationen zum Erstellen dieses Diagramms.

e) Erläutern Sie, weshalb die Beziehung zwischen der Klasse `Stack` und der Klasse `Vector` zumindest sehr

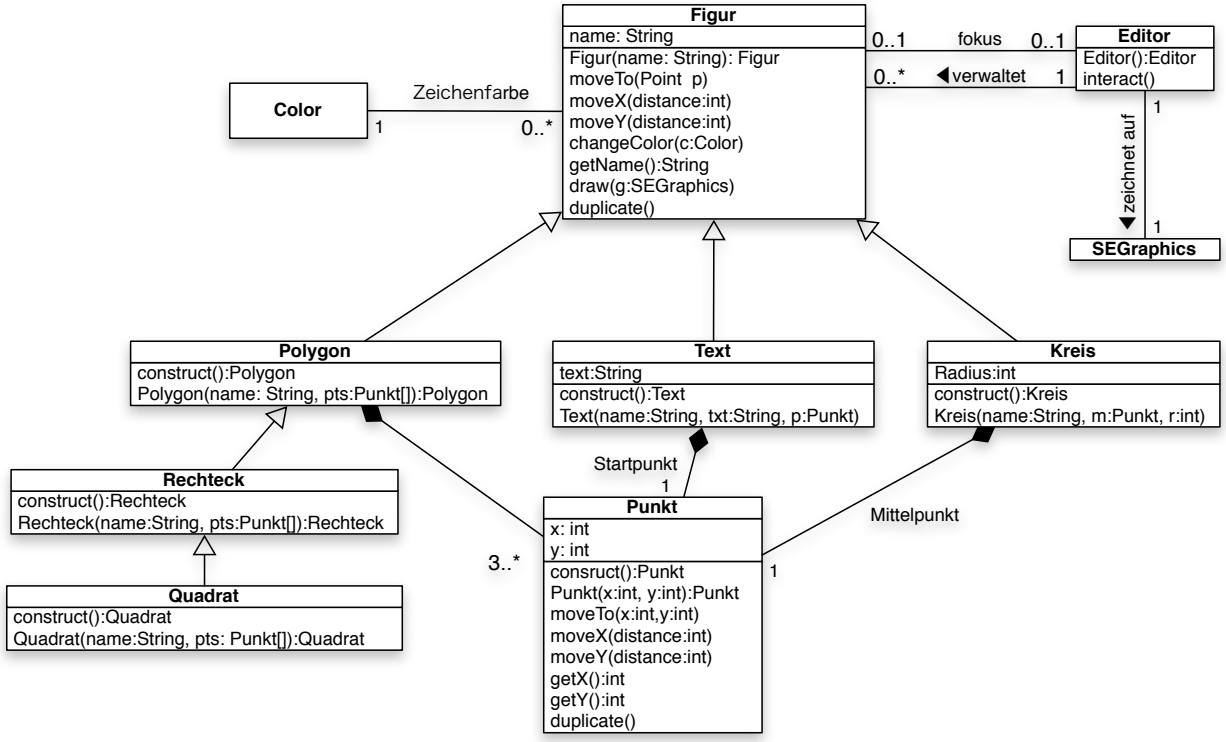
unglücklich gewählt ist.

Die Klasse `Stack` ist eine Subklasse von `Vector` und besitzt damit auch alle Methoden, die von dieser Klasse angeboten werden. Mindestens die Methode `insertElementAt` verletzt die (intuitiven) Eigenschaften des Datentyps *Stack*. Damit ist die Klasse `Stack` ein eher unschönes Beispiel für den Einsatz von Vererbung.

Aufgabe 3 Grafikeditor 2

In dieser Aufgabe werden wir den Grafikeditor von Übungsblatt 12 weiterentwickeln. Der Editor soll jetzt nicht mehr nur Polygone sondern verschiedene Figuren verwalten und darstellen können. Folgende Figuren sollen möglich sein: Rechteck, Quadrat, Kreis und Beschriftungen und natürlich weiterhin Polygone.

a) Entwerfen Sie eine Klassenhierarchie für verschiedene Arten von Figuren. Verwenden Sie eine abstrakte Klasse Figur als gemeinsame Superklasse. Verändern Sie ihr Klassendiagramm von Übungsblatt 12 Aufgabe 4, so dass es nun die neuen Klassen und ihre Beziehungen enthält.



Bei dieser Aufgabe muss man sich entscheiden, ob ein Rechteck Subtyp eines Quadrat ist oder umgehrt. Für beides scheint es gute Gründe zu geben. Zum Beispiel braucht ja ein Quadrat nur eine Kantenlänge, während ein Rechteck zwei braucht. Aus Vererbungssicht macht es also mehr Sinn, wenn ein Rechteck Subtyp eines Quadrat ist. Das Dilemma ist hier, das in Java Subtypbeziehung und Vererbung auf Klassen vermischt sind. Die beiden Argumentation lassen sich damit relativ genau auf diese beiden Sichten aufteilen. Am wichtigsten ist jedoch die Subtypbeziehung, da das *Substitutionsprinzip* jederzeit gelten muss. D.h. ein Subtyp muss überall da verwendbar sein, wo der Supertyp erwartet wird. Es macht keinen Sinn ein Rechteck zu erlauben, wenn ein Quadrat gefordert war. Es ist damit eindeutig, wie die Entscheidung hier zu treffen ist!

b) Implementieren Sie die neuen und ändern Sie Ihre bestehenden Klassen, um die neuen Figuren eingeben und darstellen zu können.

c) Erweitern Sie den Editor um die Funktion *Figur duplizieren*, die die aktuell im Fokus stehende Figur dupliziert. Beide Figuren (das Original und das Duplikat) sollen anschließend unabhängig voneinander verschoben werden können.

Bei dieser Funktion ist nur darauf zu achten, dass auch die Punkte dupliziert werden. Stichwort: Deep Copy statt Shallow Copy.