**RWTH Aachen**
**Lehrgebiet Theoretische Informatik**
Frohn–Reidl–Rossmanith–Sánchez–Ströder

**WS 2013/14**
Übungsblatt 8
16.12.2013

# Übung zur Vorlesung Programmierung

## Aufgabe T15

Betrachten Sie folgende Klassen:

```
public class A {
    public void f(long x) {
        System.out.println("A");
    }
}
public class B {
    public void f(int x) {
        System.out.println("B");
    }
}
public class C extends A {
    public void f(int x) {
        System.out.println("C");
    }
    public void g() {
        System.out.println("D");
    }
}
public class D {
    public static void main(String[ ] args) {
        A a = new A();
        B b = new B();
        C c = new C();
        A d = new C();
        *;
    }
}
```

Was ist die Ausgabe bzw. was für ein Fehler tritt auf, wenn man die *main* Methode der Klasse $D$ ausführt, wobei der $*$ durch die folgenden Ausdrücke ersetzt wurde und warum?

1. $a.f(0)$
2. $((C)b).f(0)$
3. $c.f(0L)$
4. $((A)c).f(0)$
5. $((C)a).f(0)$
6. $d.f(0)$
7. $((C)d).f(0)$
8. $((A)c).g()$
9. $d.g()$
10. $((C)d).g()$

**Aufgabe T16**

Auf der Homepage der Vorlesung finden Sie die Klassen *Person* und *PersonAdministration*. Führen Sie die Methode *PersonAdministration.main* aus. Finden Sie dabei heraus, warum sich das Programm unerwartet verhält, und beheben Sie die gefundenen Fehler. Ergänzen Sie die Klasse *PersonAdministration* anschließend um eine Methode *getPersonsSortedByName*, welche die gespeicherten Personen sortiert nach der alphabetischen Reihenfolge ihrer Namen zurückliefert. Dabei soll zunächst nach den Nachnamen und, falls diese gleich sind, anschließend nach den Vornamen sortiert werden. Schreiben Sie zu diesem Zweck eine Klasse, die das Interface *Comparator* implementiert. Achten Sie dabei darauf, dass Ihre Implementierung den in der Java-API beschriebenen Contract erfüllt. Sie dürfen bei der Lösung dieser Aufgabe voraussetzen, dass die Identifikationsnummern verschiedener Personen stets unterschiedlich sind (siehe erster Kommentar im Quellcode) und dass die Attribute *firstName* und *lastName* niemals *null* sind.

**Aufgabe H18 (3+3+3 Punkte)**

In dieser Aufgabe sollen Sie die Methode *Foo.bar* untersuchen:

```
public class Foo {
    abstract class A {
        void f() {
            System.out.println("A.f");
        }
        void h() {
            System.out.println("A.h");
        }
    }
    abstract class B extends A {
        void f() {
            System.out.println("B.f");
        }
        abstract void f(int i);
        abstract void h();
    }
    class C extends A {
        void f(int i) {
            System.out.println("C.f(int)");
        }
        void h() {
            super.h();
        }
    }
    class D extends B {
        void f(int i) {
            System.out.println("D.f(int)");
        }
        void h() {
            System.out.println("D.h");
        }
```

```
    }
    public void bar() {
      B b = new A();
      b.f();
      b.h();
      b = new B();
      b.f(0);
      b.h();
      A c = new C();
      c.f(0);
      c.h();
      B a = new D();
      a.f();
      a.h();
    }
}
```

1. Welche Anweisungen führen zu Kompilierfehlern? Warum? Falls eine Anweisung aus mehrerlei Gründen fehlerhaft ist, benennen Sie alle Gründe!

2. Welche der Klassen *A*, *B*, *C* und *D* stellt konkrete Implementierungen welcher Methoden zur Verfügung? Welche Methoden gehören zur Schnittstelle der Klassen, obwohl keine Implementierungen zur Verfügung stehen?

3. Was würde *Foo.bar* ausgeben, wenn Sie alle unzulässigen Konstruktoraufrufe, alle unzulässigen Methodenaufrufe und alle Methodenaufrufe auf Objekten, die nicht erzeugt werden können, da die zugehörigen Konstruktoraufrufe unzulässig sind, entfernen? Warum?

**Aufgabe H19 (10 Punkte)**[1]

Auf der Homepage der Vorlesung finden Sie die Klasse *Country*.

Wenn Sie versuchen, diese auszuführen, werden Sie auf Probleme stoßen. Bitte beheben Sie diese Probleme, ohne die Methode *Country.main* zu verändern. Ihre Änderung soll dabei insbesondere das im Code mit FIXME gekennzeichnete Problem lösen.

Außerdem finden Sie in der Methode *Country.main* zwei TODOs. Bitte vervollständigen Sie die Implementierung an diesen beiden Stellen sinnvoll.

Achten Sie bei all Ihren Änderungen darauf, dass Sie die in der Java-API festgelegten Contracts bezüglich des Verhaltens von Methoden, die Sie implementieren oder überschreiben, strikt einhalten. Sie dürfen dabei voraussetzen, dass zwei unterschiedliche Länder auch unterschiedliche Namen haben (siehe erster Kommentar im Quellcode) und dass das Attribut *name* niemals *null* ist.

*Tipp*: Werfen Sie einen Blick auf die Klassen *java.util.Comparable*, *java.util.Comparator*, *java.util.Arrays* und *java.util.Collections*.

**Aufgabe H20 (5 Punkte)**[1]

In dieser Aufgabe sollen Sie den Binärbaum aus der Übung T13 (auf der Webseite erhältlich) durch die Methode **void** *remove*(**int** $x$) erweitern, welche den Wert $x$ aus der Datenstruktur entfernen soll. Das Entfernen eines Wertes aus einem Binärbaum funktioniert dabei wie folgt:

- Ist $x$ in einem Blattknoten enthalten, hat also weder einen linken noch einen rechten Nachfolger, so entfernen wir schlicht dieses Blatt aus dem Baum.
- Hat der Knoten mit Inhalt $x$ einen linken Nachfolger, so bestimmen wir das maximale Element $y$ in diesem linken Teilbaum, entfernen es aus diesem und tauschen $x$ gegen $y$ im aktuellen Knoten aus.
- Hat der Knoten mit Inhalt $x$ nur einen rechten Nachfolger, so bestimmen wir das minimale Element $z$ in diesem rechten Teilbaum, entfernen es aus diesem und tauschen $x$ gegen $z$ im aktuellen Knoten aus.

*Hinweis:* Das rekursive Entfernen eines Wertes aus einem *BinaryTreeNode* wird erleichtert, wenn sie die Methode *BinaryTreeNode remove*(**int** $x$) so implementieren, dass sie die Wurzel des Teilbaumes nach Ausführen der Operation zurückliefert (in den meisten Fällen ist dies der gleiche Knoten wie zuvor). Auf diese Weise kann sich ein Blattknoten etwa selbst löschen, indem er *null* zurückliefert.

**Aufgabe H21 (10 Punkte)**[1]

In der Globalübung lernten Sie bereits die Koch'sche Schneeflocke kennen. In dieser Aufgabe widmen wir uns einem allgemeinen System, um solche und ähnliche Fraktale zu zeichnen. Dazu bedienen wir uns einer arbeitswilligen Schildkröte "Herrn $K$", die mit Hilfe einer einfachen Sprache Anweisungen ausführt. Zu Beginn setzen wir Herrn $K$ in den Koordinatenursprung und richten ihn in eine Startrichtung aus (etwa parallel zur x-Achse).

Herr $K$ versteht folgende Anweisungen:

- $F$: Bewege Dich einen Schritt in die Richtung, in die Du gerade schaust, und zeichne dabei einen Strich hinter Dir her.
- $f$: Bewege Dich einen Schritt in die Richtung, in die Du gerade schaust.
- $+$: Drehe Dich etwas nach rechts.
- $-$: Drehe Dich etwas nach links.
- [: Merke Dir Deine aktuelle Position und Ausrichtung in einer Liste.
- ]: Entferne den letzten Eintrag Deiner Liste und stelle den in diesem Eintrag beschriebenen Zustand (Ort und Ausrichtung) wieder her

Die Schrittweite sowie den Winkel, um den sich Herr $K$ bei den Anweisungen $+, -$ dreht, bezeichnen wir mit den Variablen $\ell$ bzw. $\delta$.

So bedeutet etwa die Anweisung [FF+f], dass sich Herr $K$ zunächst seinen Zustand (hier: im Ursprung, ausgerichtet parallel zur x-Achse) merkt, sich dann um $2\ell$ nach vorne bewegt

und dabei einen Strich zeichnet, sich dann um den Winkel $\delta$ nach rechts dreht, um danach um $\ell$ nach vorne zu laufen. Danach kehrt er zum Ursprung zurück und stellt seine anfängliche Ausrichtung wieder her.

Um ein Fraktal zu erzeugen, gibt man nun eine *Startregel* und eine *Ersetzungsregel* sowie eine Iterationstiefe $n$ an. Aus diesen drei Parametern erzeugen wir die endgültige Anweisung für Herrn $K$ wie folgt: Ist $n = 0$, so ist das Ergebnis die Startregel selbst. Für jedes andere $n$ erhält man das Ergebnis, indem man die Zeichenkette der $(n-1)$-ten Iteration nimmt und jedes Vorkommen des Zeichens $F$ durch die Ersetzungsregel ersetzt.

Beispielsweise ist das Ergebnis der Startregel F+F mit der Ersetzungsregel FF--F in der nullten Iteration die Zeichenkette F+F, in der ersten Iteration FF--F+FF--F und in der dritten FF--FFF--F--FF--F+FF--FFF--F--FF--F usw.

Auf unserer Webseite finden Sie das Programm `FracTrees`, jedoch unvollständig. Implementieren Sie die mit TODO gekennzeichneten Methoden in den Klassen *Turtle*, *DrawingTurtle*, *BoundaryTurtle* und *CommandBuilder*. Die gewünschte Funktionalität entnehmen Sie den dortigen Kommentaren. Testen Sie Ihre Implementierung mit Eingaben aus der Tabelle.

| Startregel | Ersetzungsregel | $\delta$ |
|---|---|---|
| F | F[+FF][-F]F | 25° |
| F-F-F-F-F | F-F++F+F-F-F | 72° |
| F++F++F | F-F++F-F | 60° |
| F | F+F-F-F+F | 90° |
| [--F][++F] | F[++FF+FF]FF[--FF-FF] | 17° |

## Abgabe zum 07.01.2014

[1] Bitte Quelltext für die Abgabe ausdrucken und zusätzlich per E-mail an den jeweiligen Tutor senden.