Universität Karlsruhe (TH)
Lehrstuhl für Programmierparadigmen

Fortgeschr. Objektorientierung SS 2008 — http://pp.info.uni-karlsruhe.de/
Dozent: Prof. Dr.-Ing. G. Snelting — snelting@ipd.info.uni-karlsruhe.de
Übungsleiter: Daniel Wasserrab — wasserra@ipd.info.uni-karlsruhe.de
Andreas Lochbihler — lochbihl@ipd.info.uni-karlsruhe.de

Übungsblatt 4 — Ausgabe: 26.5.2008 — Besprechung: 28.5.2008

1. **Verhaltenskonformanz bei reelen und komplexen Zahlen**

   Betrachten Sie zwei Klassen *Real* und *Complex* mit den Methoden `void` *plus*`(`*Complex c*`)` und `void` *square*`()`. Ist die Standarddefinition der Mathematik ("eine reelle Zahl ist eine komplexe Zahl") verhaltenskonformant? Wie sieht es mit der Umkehrung aus? Begründen Sie Ihre Argumentation anhand von Klasseninvarianten und Methoden-Vor- und Nachbedingungen.

2. **Generische Klassen und Vererbung**

   Verwendet man generische Klassen für Container, möchte man natürlich auch generische Klassen für Iteratoren verwenden:

   (a) Erstellen Sie die Signatur einer generischen Klasse *Node*`<`*A*`>`, in der jeder Knoten eine Liste von `<`*A*`>` enthält. Weiterhin soll jeder Knoten eine Liste von Vorgänger und Nachfolge-Knoten enthalten, auf die nur über Iteratoren zugegriffen werden darf.

   (b) Erstellen Sie die Signatur einer Klasse *SpecialNode*, die von *Node*`<Integer>` erbt und weiterhin die Methode `void` *foo*() und `void` *bar*() beinhaltet.

   (c) Schreiben Sie den Code für die Methode *bar*. Sie soll für alle Nachfolger des aktuellen Knotens die Methode *foo* aufrufen. Welches Problem entsteht dabei? Sehen Sie eine Möglichkeit, es zu umgehen?

3. **Generische Typanpassung in C++**

   Es gibt eine generische Matrixklasse `Matrix<`*A*`>`, von der eine eigene Matriximplementierung abgeleitet ist, die Einträge des Typs `double` besitzt:

```
1   template<class A> class Matrix {
2     A* elements;
3     ...
4   };
5
6   MyMatrix : class Matrix<double> {
7     ...
8     double entry = ...
9     ..
10  };
```

   (a) Sie wollen nun Ihre Implementierung so ändern, dass in *MyMatrix* die Einträge vom Typ `float` sind. Welche Anpassungen müssen Sie machen? Welches Problem gibt es dabei?

(b) C++ bietet hierzu jedoch eine elegante Lösung an. Welche?

4. **Bounded Polymorphism**

Geben Sie je ein Beispiel an, das zeigt, dass es sinnvoll sein kann,

(a) zwei Typschranken für einen Parameter

(b) zwei verschiedene Typparameter mit verschiedenen Schranken

haben zu können.

5. **Probleme mit Typcasts bei Generics in Java**

Betrachten sie folgendes Java-Codefragment:

```
1       public interface Board {
2         ...
3         Board clone();
4         ...
5       }
6
7       public class SudokuBoardSolver
8       implements SudokuSolver {
9         ...
10        private <B extends Board> LinkedList<B>
11        backtrack(B board) {
12          ...
13          B newBoard = (B) board.clone();
14          ...
15        }
16        ...
17      }
```

Der Java Compiler liefert standardmäßig folgende Bemerkung:

```
Note: sudoku/SudokuBoardSolver.java uses unchecked or
  unsafe operations.
```

Bei Verwendung des Parameters `-Xlint:unchecked` erhält man folgende Meldung:

```
sudoku/SudokuBoardSolver.java:86: warning:
    [unchecked] unchecked cast
found   : sudoku.Board
required: B
            B newBoard = (B) board.clone();
                                           ^
1 warning
```

(a) Wieso tritt diese Meldung auf? Überlegen Sie sich, was intern in der JVM passiert.

(b) Kann man dies verhindern? Wenn ja, wie, wenn nein, warum nicht.