Prof. Dr. Arnd Poetzsch-Heffter
Dipl.-Inf. Jan Schäfer
Dipl.-Inf. Jean-Marie Gaillourdet
Dipl.-Inform. Markus Reitz

Technische Universität Kaiserslautern
Fachbereich Informatik
AG Softwaretechnik

Aufgabenblatt 3: Praktikum Komponententechnik (WS 06/07)

Ausgabe: 14. November 2006

Aufgabe 1 Anwendung des Box-Modells

Prof. Poetzsch-Heffter hat letzte Woche das Box-Modell vorgestellt. In dieser Woche geht es darum das Modell anzuwenden und eventuell nötige Refactoring-Schritte vorzunehmen *und* diese zu dokumentieren.

Die Aufgabe besteht darin, dass für alle Typen die in Interfaces und in der Implementierung vorkommen überlegt werden soll welche Objekte dort referenziert werden können und die Typen dann ensprechend zu annotieren.

Zur Annotation sollen Java 5-Annotationen verwendet werden. Da bei `new`-Ausdrücken und bei Typ-Parametern keine Annotationen möglich sind soll dort die Annotation in einen Kommentar geschrieben werden.

Es gibt die folgenden Annotationen die an Typen geschrieben werden können:

- `@local` - Objekte in der local-Domaine der Box.
- `@boundary("x")` - Objekte in der boundary-Domaine der Box `x`. Wobei `x` eine lokale Variable oder ein Feld bezeichnet das eine Referenz auf das Owner-Objekt der entsprechenden Box hat. Wenn kein Parameter angegeben wird bezieht sich `@boundary` auf die aktuelle Box.
- `@global` - Objekte in der globalen Box.
- `@external` - Objekte die extern zu der aktuellen Box sind.
- `@boxowner` - Representiert das Owner-Objekt der Box.
- `@ro` - Read-Only Referenzen. D.h. auf solchen Referenzen dürfen keine Feld-Zuweisungen vorgenommen werden und auch nur Methoden aufgerufen werden, die nur lesend sind.
- `@constant` - Referenzen auf konstante Objekte. D.h. Objekte die ihren Zustand nie ändern. Auch hier sind nur lesende Methoden und keine Feld-Zuweisungen erlaubt.

Die Annotationen der Typen schränken ihre Benutzbarkeit ein. Im Allgemeinen gilt, dass man die Typ-Annotation bei Zuweisungen beachten muss. D.h. es ist z.B. nicht möglich eine Variable vom Typ `@local` einer anderen Variable vom Typ `@boundary` zuzuweisen. Diese Regel hat allerdings eine Ausnahme, und zwar ist es erlaubt alles an `@ro` zuzuweisen.

`@local`-Typen sind nur innerhalb der Box sichtbar. Im Gegensatz zu `private` in Java bezieht sich `@local` nur auf die aktuelle Box-Instanz, d.h. andere Instanzen der gleichen Box-Klasse können nicht gegenseitig auf den lokalen Bereich zugreifen.

Interfaces und Klassen können mit folgenden zwei Annotationen gekennzeichnet werden.

- `@Box` - Um eine Box-Klasse zu kennzeichnen.
- `@Immutable` - Um eine Klasse als immutable zu kennzeichnen. Objekte von immutable Klassen ändern ihren Zustand nach ihrere Erzeugung nicht mehr. Sie sind implizit konstant.

Mit der `@ReadOnly`-Annotation kann man Methoden kennzeichnen, die nur lesend auf das this-Objekt zugreifen. Dies sind die einzigen Methoden die auf `@ro` annotierten Typen aufgerufen werden dürfen.

Aufgabe 2 Alles Dokumentieren!

Wichtig bei dieser Aufgabe ist es nicht nur die Typen zu annotieren, sondern bei allen Fällen wo ihr merkt, dass ihr den Code nicht richtig annotieren könnt, ohne eine Regel zu verletzen, diesen Sonderfall aufzuschreiben. Falls ihr den Code ändern könnt, sodass er dem Box-Modell entspricht, dies ebenfalls aufschreiben. Und zwar was ihr geändert habt, warum und ob das neue daraus resultierende Design Vor- oder Nachteile hat.

Aufgabe 3 Beispiel

Hier ein kurzes Beispiel wie die Annotationen aussehen sollen. Es wird hier eine LinkedList-Box annotiert. Eine LinkedList-Box hat Node-Objekte in ihrer lokalen Domain, die von außen nicht zugreifbar sein sollen. Es gibt außerdem Iterator-Objekte die von außen zugreifbar sein sollen, die aber selbst wiederum auf lokale Objekte der Box zugreifen müssen. Die Daten die innerhalb der Liste gespeichert werden liegen in der externen Domaine der Box.

```
@Box
class LinkedList<D> {
  @local Node<D> head;
  void add(@external D d) {
    head = new /*@local*/ Node<D>(d,head);
  }
  @boundary Iterator<D> iterator() {
    return new /*@boundary*/ Iterator<D>(this)
  }
}

class Node<D> {
  @external D data;
  @local Node<D> next;
  Node(@external D d, @local Node<D> n) {
    data = d;
    next = n;
  }
}

class Iterator<D> {
  @local Node<D> current;
  Iterator(@boxowner LinkedList<D> list) {
     current = list.head;
  }

  @ReadOnly
  @external D next() {
     @local Node<D> temp = current;
     current = current.next;
     return temp.data;
  }
}

class Client {
  @local List</*@global*/ Object> list;
  @ro List</*@global*/ Object> getList() {
     return list; // Zuweisung von @local auf @ro
  }

  void test() {
    @boundary("list") Iterator</*@global*/ Object> it = list.iterator();
    @global Object o = it.next();
  }
}
```