

Musterlösung 3

Aufgabe 1 (Anonyme Klassen)

Der Quelltext der Lösung ist zu finden unter <http://www.inf.fu-berlin.de/lehre/WS05/ALP3/loesungen/U3.A1.jar>.

Die entscheidende Stelle der Umwandlung war:

```
// Create and set up the content pane.  
MouseListenerPanel newContentPane = new MouseListenerPanel();  
newContentPane.addMouseListener(newContentPane);
```

Hier musste newContentPane durch eine anonyme Klasse ersetzt werden.

```
// Create and set up the content pane.  
MouseListenerPanel newContentPane = new MouseListenerPanel();  
newContentPane.addMouseListener(new MouseListener() {  
    public void mousePressed(MouseEvent e) {  
        .....  
    }  
});
```

Vorteile:

- Die Verantwortlichkeiten der Klasse MouseListenerPanel werden sauber ausgelöst: Darstellung einer graphischen Oberfläche und Bearbeitung von Mausereignissen.

Nachteile:

- Da eine anonyme Klasse nicht benannt ist, kann von ihr nur ein Exemplar erzeugt werden. Ein Konstruktor kann es in einer anonymen Klasse auf Grund des fehlenden Namens ebenfalls nicht geben.

Aufgabe 2 (Typklassen in Haskell, Schnittstellen in Java)

Der Quelltext der Lösung ist zu finden unter <http://www.inf.fu-berlin.de/lehre/WS05/ALP3/loesungen/U3.A2.zip> für die Haskell-Lösung und unter <http://www.inf.fu-berlin.de/lehre/WS05/ALP3/loesungen/U3.A2.jar> für die Java-Version.

Kurze Zusammenfassung der Implementierungen:

- Feld, allocate: Durchlaufe Feld und suche freien, zusammenhängenden Bereich, welcher der Größe der Anfrage entspricht. Markiere gefundene Feldelemente als belegt oder erzeuge Ausnahme/Fehler, wenn nicht genug Platz.
- Feld, deallocate: Überprüfe Parameter und entferne Markierung.
- Freispeicherliste, allocate: Durchlaufe Freispeicherliste und überprüfe für jedes Element, ob Größe der Anfrage entspricht. Wird das Freispeicherlistenelement vollständig durch die Anfrage aufgebraucht, entferne es aus der Freispeicherliste, ansonsten verkleinere es entsprechend.

- Freispeicherliste, deallocate: Überprüfe Parameter. Füge neues Element der Freispeicherliste hinzu, welches dem freigegebenen Bereich entspricht. Sortiere Freispeicherliste und verbinde zusammenhängende Bereiche zu einem Freispeicherlisteneintrag.

Aufgabe 3 (Typverträglichkeit bei Schnittstellen - Grundlagen)

Anweisung:	Statisch Korrekt? Wenn ja, Dynamisch korrekt? Wenn nein, welcher Fehler tritt auf?	Ausgabe
<code>A a = new A();</code>	Korrekt.	-
<code>a.foo();</code>	Korrekt.	A FOO
<code>a.bar();</code>	Statisch nicht gültig. Der Typ der Variablen a (A), unterstützt keine Methode bar().	
<code>B b = new B();</code>	Korrekt.	
<code>b.foo();</code>	Korrekt.	B FOO
<code>b.bar();</code>	Korrekt.	B BAR
<code>IA ia = new IA();</code>	Statisch inkorrekt. Es gibt keine direkten Exemplare einer Schnittstelle, sondern nur Exemplare einer Klasse, welche die Schnittstelle implementiert.	
<code>IA i = a;</code>	Korrekt. Der Typ der Variablen a implementiert die Schnittstelle IA.	
<code>i.foo();</code>	Korrekt. Der Typ der Variablen i (d.h. IA) garantiert, dass seine Werte auf Objekte verweisen, welche einer Klasse angehören, welche die Methode foo() implementieren.	A FOO
<code>i.bar();</code>	Statisch inkorrekt. Die Schnittstelle IA unterstützt keine Methode bar().	
<code>i = b;</code>	Korrekt. Der Typ von b ist Untertyp des Typs von i.	
<code>i.foo();</code>	Siehe erstes i.foo().	B FOO
<code>i.bar();</code>	Statisch inkorrekt. Die Schnittstelle IA unterstützt keine Methode bar(). Obwohl also der Aufruf von bar() von dem Objekt in i (d.h. b) unterstützt wird, lässt der Übersetzer, den Aufruf nicht zu.	
<code>b = i;</code>	Statisch inkorrekt. Der Typ von i ist nicht Untertyp des Typs von b.	
<code>b = (B)i;</code>	Statisch korrekt und dynamisch korrekt. Die Typanpassung (Classcast) auf der rechte Seite der Zu-	

	weisung stellt die statische Korrektheit des Ausdrucks sicher. Dynamisch wird dann überprüft, ob die Variable <code>i</code> momentan einen Verweis auf ein Exemplar der Klasse <code>B</code> enthält. Dem ist so und damit ist die Anweisung korrekt.	
<code>a = (A)i;</code>	Statisch korrekt und dynamisch inkorrekt . Die Typanpassung auf der rechte Seite der Zuweisung stellt die statische Korrektheit des Ausdrucks sicher. Dynamisch wird dann überprüft, ob die Variable <code>i</code> momentan einen Verweis auf ein Exemplar der Klasse <code>A</code> enthält. Dem ist nicht so (<code>i</code> enthält ja ein Exemplar der Klasse <code>B</code>) und damit wird während der Ausführung eine <code>ClassCastException</code> ausgelöst.	

```

}
}

```

Aufgabe 4 (Typklassen in Haskell)

<code>data BatchedQueue t = Brep[t][t]</code>	Festlegung des Typkonstruktors <code>BatchedQueue</code> durch Angabe des zugehörigen Datenkonstruktors.
<code>instance Queue BatchedQueue where</code>	Exemplardeklaration. Der Typ <code>BatchedQueue</code> ist ein Exemplar der Typklasse <code>Queue</code> .
<code>emptyq = Brep[][]</code>	Da wir den Datenkonstruktor nicht außerhalb des Moduls sichtbar machen wollen, müssen wir eine nullstellige Funktion bereitstellen, über welche eine <code>BatchedQueue</code> bezogen werden kann.
<code>append(x, Brep[_]) = Brep[x][[]]</code>	Ist die erste der beiden Listen leer und die zweite in einem beliebigen Zustand, dann wird das hinzuzufügende Element in die erste Liste gespeichert.
<code>append(x, Brep front rear) = Brep front (x:rear)</code>	In allen anderen Fällen wird das Element der zweiten Liste vorangestellt.
<code>remove(Brep [_]) = error "queue underflow"</code>	Der Versuch aus einer leeren Schlange ein Element zu entfernen führt zum Programmabbruch.
<code>remove(Brep[x]rear) = (x, Brep(reverse rear)[])</code>	Befindet sich nur noch ein Element in der ersten Liste, dann wird dieses zurückgegeben und die zweite Liste wird in umgekehrter Reihenfolge zur ersten Liste gemacht.
<code>remove(Brep(x:front)rear) = (x, Brep front rear)</code>	Befinden sich mehr als ein Element in der ersten Liste, dann werden diese abgespalten.
<code>isEmpty(Brep front</code>	Für die Funktion <code>isEmpty</code> muss nur die erste Liste ge-

rear) = null front

prüft werden.

emptyq	Brep[][]
(append 1 *)	Brep[1][]
(append 2 *)	Brep[1][2]
(append 3 *)	Brep[1][3,2]
(append 4 *)	Brep[1][4,3,2]
(remove *)	(1, Brep[2,3,4][])
(remove *)	(2, Brep[3,4])
(append 5 *)	Brep[3,4][5]
(append 6 *)	Brep[3,4][6,5]
(remove *)	(3, Brep[4][6,5])
(remove *)	(4, Brep[5,6][])
(remove *)	(5, Brep[6][])
(append 7 *)	Brep[6][7]
(append 8 *)	Brep[6][8,7]

An der Stelle von * steht jeweils der vorangehende Ausdruck.

Der Vorteil liegt in der ausschließlichen Verwendung des Listenkonstruktors : im Gegensatz zur Verwendung der Listenkonkatenation ++ in der Standard-queue. Ersterer ist wesentlich besser im Bezug auf sein Laufzeitverhalten und wiegt den zusätzlichen Einsatz der Funktion `reverse` auf.

Aufgabe 5 (Java Wiederholung)

Sie finden erneut Testfälle und eine Musterlösung auf der Webseite <http://projects.mi.fu-berlin.de/w/bin/view/SS/JavaWiederholung>.