Technische Universität München
Fakultät für Informatik
R. Ebner, C. Herzog

WS 1998/99
Lösungsvorschläge zu Blatt 5
30. Oktober 1998

Übungen zum Programmieren in Java

Aufgabe 1 Exceptions

a) Folgende Fehlersituationen wurden in den vier konkreten Realisierungen „stillschweigend“ dadurch bereinigt, daß der Aufruf von `insert()` wirkungslos blieb:

- In jeder der vier Konkretisierungen:
  Das neu einzufügende Element ist bereits in der Menge enthalten.
  Dies könnte unbeabsichtigt sein und auf einen Fehler auf der Seite des Aufrufers hindeuten.
- In der Klasse `ArrayIntegerSet`:
  Das zur Aufnahme der Elemente angelegte Feld ist bereits voll.
- In der Klasse `BoolArrayIntegerSet`:
  Der `Integer`–Wert liegt außerhalb des darstellbaren Bereiches `[low,high]`.

b) Die Oberklasse `IntegerSetException` dient dazu, alle Exceptions, die bei der Realisierung der Mengen auftreten können, gemeinsam abfangen zu können, und ist nach dem allgemeinen Schema zur Definition eigener Exception realisiert:

```
public class IntegerSetException extends Exception {
    public IntegerSetException() {
        super();
    }
    public IntegerSetException(String s) {
        super(s);
    }
}
```

Für die in Teilaufgabe a) gefundenen Fehlersituationen werden nun entsprechende Unterklassen dazu definiert. Ein zusätzlicher Konstruktor mit passenden Parametern erzeugt dabei jeweils geeignete Fehlermeldungen:

- Mehrfaches Einfügen desselben Elements:

```
public class MultipleElementIntegerSetException
                            extends IntegerSetException {
    public MultipleElementIntegerSetException() {
        super();
    }
    public MultipleElementIntegerSetException(String s) {
        super(s);
    }
```

```
        public MultipleElementIntegerSetException(Integer i) {
            super("multiple Element: " + i);
        }
    }
```

- Überfüllung des Feldes:

```
    public class SizeArrayIntegerSetException
                          extends IntegerSetException {
        public SizeArrayIntegerSetException() {
            super();
        }
        public SizeArrayIntegerSetException(String s) {
            super(s);
        }
        public SizeArrayIntegerSetException(int maxSize) {
            super("maximum size is " + maxSize);
        }
    }
```

- Wert liegt außerhalb des zulässigen Bereiches:

```
    public class RangeBoolArrayIntegerSetException
                               extends IntegerSetException {
        public RangeBoolArrayIntegerSetException() {
            super();
        }
        public RangeBoolArrayIntegerSetException(String s) {
            super(s);
        }
        public RangeBoolArrayIntegerSetException
                                    (Integer i, int low, int high) {
            super(i + " is out of range [" + low + "," + high + "]");
        }
    }
```

Anmerkung: Alternativ zu den zusätzlichen Konstruktoren könnte man die entsprechenden Fehlermeldungen auch beim Erzeugen einer Ausnahme generieren, insbesondere dann, wenn die Ausnahme nur an einer Stelle im Programm erzeugt wird.

c) (i) Um Namenskonflikte zu vermeiden, wurde allen Klassen von Übungsblatt 3 durch Voranstellen eines `E` ein neuer Namen gegeben.

Folgende Änderungen ergeben sich bei der Definition der `insert()`–Methode

- in der Klasse `EListIntegerSet`:

```
public void insert(Integer i)
        throws MultipleElementIntegerSetException   {
    if (contains(i))
       throw new MultipleElementIntegerSetException(i);
    head = new ENode(i, head);
}
```

- in der Klasse `EArrayIntegerSet`:

```
public void insert(Integer i)
        throws MultipleElementIntegerSetException,
               SizeArrayIntegerSetException       {
```

```
        if (contains(i))
           throw new MultipleElementIntegerSetException(i);
        if (currentSize >= array.length)
           throw new SizeArrayIntegerSetException(array.length);
        array[currentSize++] = i;
    }
```

- in der Klasse `EOrderedListIntegerSet`:

```
    public void insert(Integer i)
            throws MultipleElementIntegerSetException   {
        int k = i.intValue();
        if (head == null || ((Integer)head.data).intValue() > k)
            head = new ENode(i, head);
        else
        {
            ENode currNode = head;
            while(currNode.next != null &&
                  ((Integer)currNode.next.data).intValue() < k)
                currNode = currNode.next;
            if(currNode.next == null ||
               ((Integer)head.next.data).intValue() > k)
                currNode.next = new ENode(i, currNode.next);
            else
                throw new MultipleElementIntegerSetException(i);
        }
    }
```

- in der Klasse `EBoolArrayIntegerSet`:

```
    public void insert(Integer i)
            throws MultipleElementIntegerSetException,
                   RangeBoolArrayIntegerSetException        {
        int index = indexOf(i);
        if (index < 0 || index >= array.length)
           throw new RangeBoolArrayIntegerSetException(i,low,high);
        if (array[index])
           throw new MultipleElementIntegerSetException(i);
        array[index] = true;
    }
```

Außerdem muß in der Klasse `EAbstractIntegerSet` bekanntgemacht werden, welche Exceptions in ihren Unterklassen möglicherweise durch die Methode `insert` erzeugt werden. Dafür verwenden wir die Oberklasse all dieser Exceptions, nämlich `IntegerSetException`:

```
public abstract void insert(Integer i)
    throws IntegerSetException ;
```

(ii) Beim Aufruf der `insert()`–Methoden müssen nun die möglicherweise auftretenden Exceptions geeignet behandelt werden:

- In den Klassen `EOrderedListIntegerSet` und `EBoolArrayIntegerSet` tritt der Aufruf von `insert()` jeweils in dem Konstruktor auf, der eine übergebene Menge kopiert. Dort können die oben angesprochenen Fehlersituationen nicht entstehen, falls die übergebene Menge korrekt besetzt ist. Deshalb werden die Ausnahmen dort nicht behandelt:

```
    public EOrderedListIntegerSet(EAbstractIntegerSet s)
                throws MultipleElementIntegerSetException   { ... }
```

und analog

```
public EBoolArrayIntegerSet(EAbstractIntegerSet s)
            throws MultipleElementIntegerSetException,
                   RangeBoolArrayIntegerSetException    { ... }
```

- Im Hauptprogramm in der Klasse `EMengenTest` wird der Aufruf von `insert()` eingebettet in eine neue Methode `myInsert()`, die alle auftretenden Ausnahmen dadurch behandelt, daß Typ und Meldung der Ausnahme in die Standard-Fehlerausgabe `System.err` geschrieben wird:

```
public class EMengenTest
{
   static void myInsert(EAbstractIntegerSet s, Integer i) {
      try { s.insert(i); }
      catch (IntegerSetException e) {
          System.err.println("Achtung: Beim Einfügen von " + i +
              " in die Menge " + s);
          System.err.println("        trat eine Exception auf!");
          System.err.println("        Klasse: " + e.getClass().getName());
          System.err.println("        Meldung: " + e.getMessage());
      }
   }

   public static void main (String[] args)
   {
      // Fuer jeden konkreten Typ eine Menge definieren:
      EListIntegerSet ls = new EListIntegerSet();
      EArrayIntegerSet as = new EArrayIntegerSet();
      EOrderedListIntegerSet os = new EOrderedListIntegerSet();
      EBoolArrayIntegerSet bs = new EBoolArrayIntegerSet();

      // Bei den folgenden Einfuegungen sollte keine Exception auftreten:
      for (int i=1; i<11; i++) {
         myInsert(ls, new Integer(i));
         myInsert(as, new Integer(i));
         myInsert(os, new Integer(i));
         myInsert(bs, new Integer(i));
      }

      // Jetzt sollte jeweils eine MultipleElement-Exception auftreten:
      myInsert(ls, new Integer(5));
      myInsert(as, new Integer(5));
      myInsert(os, new Integer(5));
      myInsert(bs, new Integer(5));

      // Erzwingen einer Size-Exception:
      for (int i=11; i<102; i++)
         myInsert(as, new Integer(i));

      // Erzwingen einer Range-Exception:
      myInsert(bs, new Integer(104));
   }
}
```

Aufgabe 2 Clock-Applet

a) Da die meisten Browser die HTML-Datei verstehen, selbst wenn nur das APPLET-Tag allein vorhanden ist, reicht Folgendes:

```
<applet code="Clock.class" width=300 height=300>
     <param name=interval value="500">
</applet>
```

b)
```
String s = getParameter ("interval");
repaintInterval = (s == null) ? 1000 : Integer.parseInt (s);
```

Beachten Sie, dass der Parameter `interval` ein String ist.

c) Eine Übung zum Zeichnen einfacher Graphik-Elemente:

```
g.setColor (Color.blue);
g.drawLine (centerX, centerY, hourX, hourY);
g.drawLine (centerX, centerY, minuteX, minuteY);
g.setColor (Color.red);
g.drawLine (centerX, centerY, secondX, secondY);
g.setColor (Color.black);
g.drawOval (0, 0, clockSize, clockSize);
```

d) Die `start()`-Methode eines Threads ruft die Methode `run()` des `Runnable`-Objekts auf, das dem Thread-Konstruktor als Argument übergeben wurde. Entsprechend muss in die Klasse `Clock` das Interface `Runnable` implementieren.

Beachte: Wurde der Thread durch Aufruf von `Thread()` ohne Runnable-Argument erzeugt, wird durch `start()` die (standardmäßig leere) run()-Methode des Threads selbst aufgerufen. In dieser Aufgabe spart man sich aber die Spezialisierung der Klasse `Thread`, indem `run()` einfach als Methode des Applets definiert wird.

e) Mit dem Kommando `threads` im Debugger `jdb` (bzw. `appletviewer -debug`) kann man feststellen, dass neben den beiden Benutzer-Threads (Clock-Applet und repaint-Thread) noch vier weitere Threads im Hintergrund laufen:

Klasse	Bezeichnung	Zweck
java.awt.EventDispatchThread	AWT-EventQueue-0	Abarbeitung der Event-Schlange
sun.awt.motif.InputThread	AWT-Input	Tastatur/Maus-Eingabe
java.lang.Thread	AWT-Motif	Window-Manager
sun.awt.ScreenUpdater	Screen Updater	Graphik-Thread des Applets

f) Eine weitere einfache Aufgabe im Zusammenhang mit Threads:

```
public void stop() {
    if (repaintThread != null) {
        repaintThread.stop();
        repaintThread = null;
    }
}
```