Softwarekonstruktion - Exercise 5

5 Design by Contract

This exercise should be solved until **Wednesday (23:59 latest), November 24th, 2010**. You have to submit your solution to your tutor by email:

Holger Schmidt: holger.schmidt [at] cs.tu-dortmund.de
Gregor Kotainy: gregor.kotainy [at] tu-dortmund.de

You have to work in groups of two or three persons. Only one person per group has to submit a group's solution. State the names and matriculation numbers of the group members in your email and as a comment in each of your source code files.

5.1 Pre- & Postconditions

Given are the classes `Element` and `Queue`. A queue consists of elements to which an integer-value has been assigned. An element holds a reference to its successor element.

class	**Element**
attribute	value: Integer
	next_element: Element
method	setNextElement(new_element: Element)
	getNextElement(): Element
	getValue(): Integer
end class	**Element**

class	**Queue**
attribute	nb_elements: Integer
	max_size : Integer
	first_element : Element
	last_element : Element
method	empty(): Boolean
	full(): Boolean
	add(new_element: Element)
	remove(): Element
	peek(): Element
end class	**Queue**

Specify pre- and postconditions for the operations of the class **Queue** (similar to the class **Stack** as presented in the lecture notes pp. 165). Note that you have to submit your solution electronically in PDF format. For example, you can use LaTeX, or you can prepare a scan of your handwritten solution.

`empty:` returns `true` if the queue is empty.

`full:` returns `true` if the queue is full.

`add:` adds the specified element to the end of the queue.

`remove:` retrieves and removes the first element of the queue.

`peek:` retrieves, but does not remove, the first element of the queue.

- **empty() : Boolean**

  pre true

  post Result = true $\iff$ nb_elements = 0

- **full(): Boolean**

  pre true

  post Result = true $\iff$ nb_elements = max_size

- **add(new_element: Element)**

  pre **not** full **and** new_element $\neq$ NULL

  post last_element = new_element **and**
  nb_elements = **old** nb_elements + 1 **and** (
  ((**old** nb_elements) = 0 **and** first_element = new_element) **or**
  ((**old** nb_elements) > 0 **and** (**old** last_element).getNextElement() = new_element)
)

- **remove() : Element**

  pre **not** empty

  post Result = **old** first_element **and**
  nb_elements = **old** nb_elements - 1 **and**
  first_element = (**old** first_element).getNextElement() **and**
  old nb_elements = 1 $\Rightarrow$ last_element = NULL

- **peek(): Element**

  pre **not** empty

  post Result = first_element

5.2 Java

Implement the classes `Element` and `Queue` in Java. When creating a new *Java Project* in *Eclipse*, choose *Use default JRE*. Use *assertions* (http://java.sun.com/j2se/1.4.2/docs/guide/lang/assert.html) to test the pre- and postconditions of the methods `add`, `remove` and `peek`:

- Use the following form of the assertion statement:
  `assert` $Expression_1$ `:` $Expression_2$
  When the system runs the assertion, it evaluates $Expression_1$ and if it is `false` throws an `AssertionError`. The system passes the value of $Expression_2$ to the `AssertionError` constructor, which uses the string representation of the value as the error's detail message.

- Because assertions may be disabled, programs must not assume that the boolean expression contained in an assertion will be evaluated. Do *not* use assertions to do any work that your application requires for correct operation.

- Occasionally it is necessary to save some data prior to performing a computation in order to check a postcondition. You can do this with two `assert` statements:
```
int oldValue = 0;
assert (oldValue = nb_elements) == nb_elements :  "saving data"
...
assert nb_elements == oldValue + 1 :  "nb_elements not increased";
```

- To enable assertions in Eclipse, go to *Window* → *Preferences* and select *Java* → *Installed JRE's*. Select your JRE, click *Edit* and enter **-ea** under *Default VM Arguments*.

```
1  public class Element {
2          private int value;
3          private Element next_element;
4          public Element(int value) {
5                  this.value = value;
6          }
7          public void setNextElement(Element new_element) {
8                  this.next_element = new_element;
9          }
10         public Element getNextElement() {
11                 return next_element;
12         }
13         public int getValue() {
14                 return value;
15         }
16 }
```

Listing 1: element.java

```
1  public class Queue {
2          private int nb_elements;
3          private int max_size;
4          private Element first_element;
5          private Element last_element;
6          public Queue(int max_size) {
7                  this.max_size = max_size;
8                  this.nb_elements = 0;
9                  this.first_element = null;
10                 this.last_element = null;
11         }
12         /**
13          * pre: true
14          * post: Result = true <=> nb_elements = 0
15          */
16         public boolean empty() {
17                 return nb_elements == 0;
18         }
19         /**
20          * pre: true
21          * post: Result = true <=> nb_elements = max_size
22          */
23         public boolean full() {
24                 return nb_elements == max_size;
25         }
26         /**
27          * pre: not full AND new_element != NULL
28          * post: last_element = new_element
29                           AND nb_elements = old nb_elements + 1
30                           AND(
31                                   ((old nb_elements) = 0 AND first_element =
                                        new_element)
32                           OR      ((old_nb_elements) > 0 AND (old last_element).
                                 getNextElement() = new_element)
33                                   )
34          */
35         public void add(Element new_element) {
36                 // test of precondition
37                 assert full() == false : "pre of 'add' violated: Queue is full"
                       ;
38                 assert new_element != null : "pre of 'add' violated:
                       new_element is NULL";
39                 if (empty()) {
40                         // save nb_elements for test of postcondition
41                         int oldValue = 0;
42                         assert (oldValue = nb_elements) == nb_elements : "
                               saving nb_elements";
43                         first_element = new_element;
44                         last_element = new_element;
45                         nb_elements = nb_elements + 1;
46                         // test of postcondition
47                         assert first_element == new_element : "post of 'add'
                               violated: first_element != new_element";
48                         assert last_element == new_element: "post of 'add'
                               violated: last_element != new_element";
49                         assert nb_elements == oldValue + 1 : "post of 'add'
                               violated: nb_elements not increased";
50                 } else {
51                         // save nb_elements for test of postcondition
52                         int oldValue = 0;
```

```
53                          assert (oldValue = nb_elements) == nb_elements : "
                                saving nb_elements";
54                          // save last_element for test of postcondition
55                          Element e = null;
56                          assert ((e = last_element) != null) : "saving
                                last_element";
57                          last_element.setNextElement(new_element);
58                          last_element = new_element;
59                          nb_elements = nb_elements + 1;
60                          // test of postcondition
61                          assert e.getNextElement() == new_element : "post of '
                                add' violated: new element not added to the list
                                properly";
62                          assert last_element == new_element : "post of 'add'
                                violated: last_element != new_element";
63                          assert nb_elements == oldValue + 1 : "post of 'add'
                                violated: nb_elements not increased";
64                  }
65          }
66          /**
67           * pre: not empty
68           * post: Result = old first_element
69                           AND nb_elements = old nb_elements - 1
70                           AND first_element = (old first_element).getNextElement
                                ()
71                           AND old nb_elements = 1 => last_element = NULL
72           */
73          public Element remove() {
74                  // test of precondition
75                  assert empty() == false : "pre of 'remove' violated: Queue is
                        empty";
76                  if (nb_elements == 1) {
77                          // save nb_elements for test of postcondition
78                          int oldValue = 0;
79                          assert (oldValue = nb_elements) == nb_elements : "
                                saving nb_elements";
80                          Element returnElement = first_element;
81                          first_element = first_element.getNextElement();
82                          last_element = first_element;
83                          nb_elements = nb_elements - 1;
84                          // test of postcondition
85                          assert first_element == null : "post of 'remove'
                                violated: first_element != null";
86                          assert last_element == null : "post of 'remove'
                                violated: last_element != null";
87                          assert nb_elements == oldValue - 1 : "post of 'remove'
                                violated: nb_elements not decreased";
88                          return returnElement;
89                  } else {
90                          // save nb_elements for test of postcondition
91                          int oldValue = 0;
92                          assert (oldValue = nb_elements) == nb_elements : "
                                saving nb_elements";
93                          // save second element
94                          Element e = null;
95                          assert ((e = first_element.getNextElement()) != null) :
                                 "saving second element";
96                          Element returnElement = first_element;
97                          first_element = first_element.getNextElement();
98                          nb_elements = nb_elements - 1;
99                          // test of postcondition
```

```
100                         assert first_element == e : "post of 'remove' violated:
                                first_element != old second element";
101                         assert nb_elements == oldValue - 1 : "post of 'remove'
                                violated: nb_elements not decreased";
102                         return returnElement;
103                 }
104         }
105         /**
106          * pre: not empty
107          * post: Result = first_element
108          */
109         public Element peek() {
110                 // test of precondition
111                 assert empty() == false : "pre of 'peek' violated: Queue is
                        empty";
112                 return first_element;
113         }
114 }
```

Listing 2: queue.java

```
1  public class Main {
2          public static void main(String[] args) {
3                  Queue q = new Queue(5);
4                  System.out.println(q.empty());
5                  System.out.println(q.full());
6                  //System.out.println(q.peek().getValue()); // violation of pre
                       of 'peek'
7                  //System.out.println(q.remove().getValue()); // violation of
                       pre of 'remove'
8                  q.add(new Element(1));
9                  q.add(new Element(2));
10                 q.add(new Element(3));
11                 q.add(new Element(4));
12                 q.add(new Element(5));
13                 //q.add(new Element(6)); // violation of pre of 'add'
14                 System.out.println(q.peek().getValue());
15                 System.out.println(q.remove().getValue());
16                 System.out.println(q.peek().getValue());
17                 System.out.println(q.remove().getValue());
18                 System.out.println(q.peek().getValue());
19                 System.out.println(q.remove().getValue());
20                 System.out.println(q.peek().getValue());
21                 System.out.println(q.remove().getValue());
22                 System.out.println(q.peek().getValue());
23                 System.out.println(q.remove().getValue());
24                 //System.out.println(q.peek().getValue()); // violation of pre
                       of 'peek'
25                 //System.out.println(q.remove().getValue()); // violation of
                       pre of 'remove'
26         }
27 }
```

Listing 3: main.java