Prof. Dr. Peter Thiemann
Manuel Geffken
Matthias Keil

Summer Term 2013

Softwaretechnik

http://proglang.informatik.uni-freiburg.de/teaching/swt/2013/

Exercise Sheet 5

Exercise 1 (12 points)

The following Java class shows an implementation of queues in Java.

```
public class Queue {

  protected int in,out;
  protected Object[] buf;

  public Queue (int capacity) {
    buf = new Object[capacity];
  }

  public boolean empty() {
    return in - out == 0;
  }

  public boolean full() {
    return in - out == buf.length;
  }

  public void enqueue(Object o) {
    buf[in % buf.length] = o;
    in++;
  }

  public Object dequeue() {
    Object o = buf[out % buf.length];
    out++;
    return o;
  }
}
```

(i) Give reasonable pre- and postconditions (in first-order logic "syntax") for all methods and the constructor of the Queue class. In particular, keep in mind that integers may overflow.

(ii) A *weak class invariant* is defined as a condition that holds between calls to methods of the class, but not during the execution of such methods. Are there any weak class invariants for the Queue class? If yes, state these class invariants.

Exercise 2 (10 points)

The `Iterator<E>` interface has been presented in the lecture. Consider the following `ListIterator<E>` interface, which is a simplified variant of the standard library's list iterator. A `ListIterator<E>` supports back and forth navigation in lists.

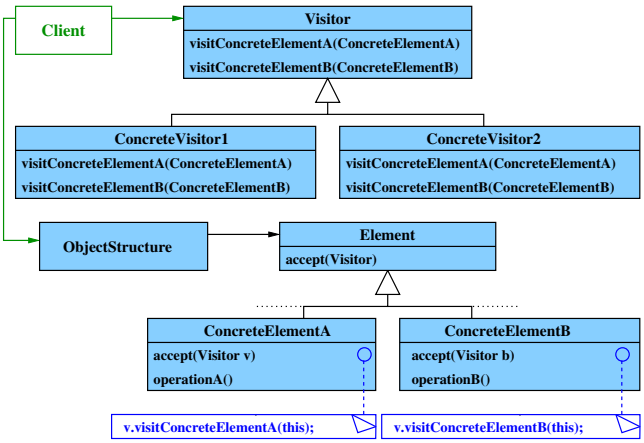
```
public interface ListIterator<E> extends Iterator<E> {

  /**
   * Returns true if this list iterator has more elements when
   * traversing the list in the forward direction.
   *
   * @return true if the list iterator has more elements when
   *         traversing the list in the forward direction
   */
  boolean hasNext();

  /**
   * Returns the next element in the list and advances the
      cursor position.
   *
   * @return the next element in the list
   */
  E next();

  /**
   * Returns true if this list iterator has more elements when
   * traversing the list in the reverse direction.
   *
   * @return true if the list iterator has more elements when
   *         traversing the list in the reverse direction
   */
  boolean hasPrevious();

  /**
   * Returns the previous element in the list and moves the
      cursor position backwards.
   *
   * @return the previous element in the list
   */
  E previous();
  ...
}
```

Provide a *UML state machine* representing the knowledge a user (caller) can gain about a list iterator that implements the `ListIterator<E>` interface.
Consider all operations that change the state of the iterator, provide additional, or reduce the knowledge about the list iterator's state. You may omit those operations that leave the state machine's state unchanged.

Hint: You should be able to distinguish at least 9 states.

Exercise 3 (10 points)

Recapitulate the *visitor pattern* presented in the lecture.



A concrete visitor (e.g. *ConcreteVisitor1*) wants to call operation *operationA* on *ConcreteElementA* elements and operation *operationB* on *ConcreteElementB* elements. Based on the class diagram shown above, provide a *UML sequence diagram* for applying the visitor to two elements, one of each class.