Formal Methods in Software Development
Exercise 9 (January 21)

Wolfgang Schreiner
Wolfgang.Schreiner@risc.jku.at

January 10, 2013

The result is to be submitted by the deadline stated above *via the Moodle interface* of the course as a *.zip or .tgz* file which contains

1. a PDF file with
   - a cover page with the course title, your name, Matrikelnummer, and email address,
   - the deliverables requested in the description of the exercise,
2. the JML-annotated Java files developed in the exercise,

Email submissions are *not* accepted.

8a (60 points): A Private JML Class Specification

Take the attached source code of a class `BoundedQueue` which implements a bounded queue of integers and extend it by a *private* specification in the *heavy-weight* JML format that is as expressive as possible.

Use `jml -Q` to check the specification (which must not yield an error). Run `escjava2 -NoCautions` on the specification; if the tool gives warnings, take these seriously.

The result of this exercise contains the JML-annotated file `BoundedQueue.java` and the output of `jml -Q` and `escjava2 -NoCautions` on this file.

Hint: try to express the main knowledge about the ranges of the variables by invariants. Also state in an invariant, that the number of elements `count` can be deduced uniquely from `head` and `tail`, unless `head` equals `tail`; in that case, the queue may be either full or empty.

8b (40 points): A Public JML Class Specification

Take the previously JML-annotated file `BoundedQueue.java` and modify it for an appropriate *public* specification of class `BoundedQueue`; this public specification is to be written into file `BoundedQueue.jml` and shall be based on the abstract datatype `QueueModel` which specifies an unbounded queue in the attached file `QueueModel.java`.

The core idea of modeling a bounded queue (`BoundedQueue`) by an unbounded queue (`QueueModel`) is is that the public function `size()` in `BoundedQueue` poses an upper limit on the length of the model queue; we can simply express this by an invariant. A constructor call `BoundedQueue(n)` sets the limit to `n`, which has to be appropriately specified. The limit is not changed by any of the other functions, which can be specified by a corresponding constraint. A call of `enqueue()` is only allowed, if the upper limit is not reached, which can be expressed by a corresponding precondition.

Some further hints:

- Generally the basic specification strategy is the same as shown in class for the model-based public specification of class `IntStack`.
- Introduce in `BoundedQueue.jml` a model field of type `QueueModel` which receives its value from a model function `toModel()`.
- Give in `BoundedQueue.jml` public specifications of the public functions using the model field and the corresponding operations on `QueueModel`.
- Annotate `BoundedQueue.java` by a `refines` annotation that indicates that the definition of class `BoundedQueue` in this file is a refinement of the class declared in `BoundedQueue.jml`. Add the keyword `also` to the private specifications of all public methods.
- Give a specification-only definition of the abstraction function `toModel` as

```
/*@ public pure model QueueModel toModel() {
  @   QueueModel q = new QueueModel();
  @   int index = head;
  @   for (int i=0; i<count; i++)
  @   {
  @     q = q.enqueue(a[index]);
  @     index = index+1;
  @     if (index == a.length) index = 0;
  @   }
  @   return q;
  @ }
  @*/
```

Annotate this definition with a *private* behavior specification that relates the constructed `QueueModel` to the current `BoundedQueue` object.

- Add the private object variables to the data group of the model variable; thus whenever an assignment on the model variable in the public specification is allowed, also an assignment to the private variables in the implementation is allowed.

First use `jml -Q` to type-check `BoundedQueue.jml` in a directory that contains also `QueueModel.java` but does not contain `BoundedQueue.java` (otherwise also this file will be immediately type-checked). As soon as the type-check succeeds, also add the file `BoundedQueue.java` from the previous exercise to this directory and extend it as indicated above. Now use `jml -Q` again to type-check the files.

The result of the exercise contains the files `BoundedQueue.jml`, `BoundedQueue.java`, and also `QueueModel.java`, and the output of `jml -Q`.