Formal Methods in Software Development
Exercise 7 (January 16)

Wolfgang Schreiner
Wolfgang.Schreiner@risc.jku.at

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,
   - a section for each part of the exercise with the requested deliverables and optionally any explanations or comments you would like to make;
2. the JML-annotated `.java`/`.jml` file(s) used in the exercise,
3. the proof files generated by the KeY prover (use the menu option "Save").

Email submissions are *not* accepted.

Exercise 7a (70 points): A Private JML Class Specification

Take the attached source code of a class `BoundedQueue` which implements a "queue" (a first-in/first-out sequence) of integers with an upper bound on the number of elements in the queue. Extend this source by a *private* specification in the *heavy-weight* JML format that is as expressive as possible. Pay attention to provide a suitable object invariant that describes the ranges of the variables and the contents of the arrays as accurately as possible.

Use `jml -Q` and `openjml` to check the specification (which must not yield an error). Run `escjava2 -NoCautions` and `openjmlesc` on the specification; if these tools give warnings, take them seriously. Use KeY to verify the contracts of the various methods as far as possible.

The result of this exercise contains the JML-annotated file `BoundedQueue.java`, the output of `jml -Q`, `openjml`, `escjava2 -NoCautions`, and `openjmlesc` on this file, and a screenshot of the final state of KeY for the verification of each method plus an explicit statement whether the verification suceeded (if not, then try to analyze the failed verification and give your estimation, why it did not succeed).

Exercise 7b (30 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 the file `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`.