Formal Methods in Software Development
Exercise 7 (December 12)

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,
   - the deliverables requested in the description of the exercise,
2. the JML-annotated Java files developed in the exercise.

Email submissions are *not* accepted.

Exercise 7: JML Specifications

Formalize the method specifications given below in the JML *heavy-weight* format by a precondition (`requires`), frame condition (`assignable`), and postcondition (`ensures`) and attach the specification to the method implementations provided in file `Exercise7.java`. For this purpose, extract the implementation of each method into a separate class `Exercise7_`*`I`* (where *I* is the number of the method in the list below) and give this class a `main` function that allows you to test the implementation by a call of the corresponding method.

Make preconditions as *weak* as possible; e.g. if the method can be reasonably applied to argument 0, do not require that the argument needs to be positive. Make postconditions as *strong* as possible; e.g. if a result is always positive, do not just ensure that the result is non-negative. Also do not forget to explicitly specify the null/non-null status and the lengths of arrays.

For each method, first use `jml` to type-check the specification. Then use the runtime assertion compiler `jmlc` and the corresponding executor `jmlrac` to validate the specification respectively implementation by at least three calls of each method; the calls shall contain at least two different valid inputs and (if possible) also one invalid input (for arrays, use arrays with wrong length or content, not just null pointers). Please print after each method call some output to make sure that the method has not silently crashed. You may also try the alternative tool set `openjmlrac/openjmlrun`; please report your experience with this. If you detect that the runtime assertion compiler fails for some part of the specification, you may comment it out as an informal property `(* ...*)` and repeat the check with the simplified specification.

Second, use the extended static checker `escjava2` to further validate the code; use the option `-NoCautions` to suppress any cautions you may get from system libraries. You may also try the alternative extended static checker `openjmlesc` (please report your experience).

The deliverables of this exercise consist of

- a nicely formatted copy of the JML-annotated Java code for each class,
- the output of running `jml -Q` on the class,
- the output(s) of running `jmlrac/openjmlrun` on the class,
- the output of running `escjava2/openjmlesc` on the class.

both for the original and for the modified implementation of the method (if the implementation was modified) including an explanation of the detected error and how you fixed it.

Please note that the fact that `escjava2/openjmlesc` does not give a warning does not prove that the function indeed satisfies the specification (only that the tool could not find a violation); on the other hand, if the checker reports a warning, this does not necessarily mean that the program indeed violates its specification (only that the tool could not verify its correctness).

Recommendation: it is better to split pre/post-conditions that form conjunctions into multiple `requires` respectively `ensure` clauses (one for each formula of the conjunction); if an error is reported, it is then clear, to which formula it refers.

1. Specify the method

   ```
   public static int minimumPosition(int[] a)
   ```

   that takes an integer array *a* and returns the position of the smallest element in the array.

2. Specify the method

   ```
   public static int minimumElement1(int[] a)
   ```

   that takes an integer array *a* and returns the smallest element in the array.

3. Specify the method

   ```
   public static int minimumElement2(int[] a)
   ```

   that takes an integer array *a* and returns the smallest element in the array.

4. Specify the method

   ```
   public static int[] append(int[] a, int[] b)
   ```

   that returns a new array that contains the elements of *a* followed by the elements of *b*.

5. Specify the method

   ```
   public static boolean replace(char[] a, char x, char y)
   ```

   that takes a character array *a* and replaces in it every character *x* by *y*. The return value of the function indicates whether any replacement has been performed.

6. Specify the method

   ```
   public static int add(int[] a, int[] b)
   ```

   that takes two arrays *a* and *b* that hold non-negative integers and adds to every element of *a* the corresponding element of *b* unless this would result in an overflow. The return value of the function denotes the minimum position in the array where an overflow would have occurred (-1, if none).

   Please note that the test `a[i] > Integer.MAX_VALUE-b[i]` must not be replaced by `a[i]+b[i] > Integer.MAX_VALUE`, neither in Java nor in JML (why?).

7. Specify the method

   ```
   public static void add2(int[] a, int[] b) throws OverflowException
   ```

   that takes two arrays *a* and *b* that hold non-negative integers and adds to every element of *a* the corresponding element of *b* unless this would result in an overflow. In this case, an exception is thrown that indicates the corresponding error position; all subsequent elements remain unchanged.

   You may ignore any warning about the possible violation of a `modifies` clause of class `OverflowException`; this is due to an underspecification of the superclass `Exception`.

Please note that the given specifications/implementations may be too weak, ambiguous, or erroneous. If you detect problems, explain them, fix them such that specification and code match and re-run your checks (concentrate on fixing specifications; change an implementation only, if it apparently contains a bug, i.e., no reasonable specification of the code is possible).