Formal Methods in Software Development
Exercise 6 (December 13)

Wolfgang Schreiner
Wolfgang.Schreiner@risc.jku.at

November 12, 2010

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.

Exercise 6: JML Specifications

Formalize the following method specifications 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 `Exercise6.java`. For this purpose, extract the implementation of each method into a separate class `Exercise6_`*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 some calls of the corresponding method.

For each method, first use `jml` to type-check the specification. Then use the JML runtime assertion compiler `jmlc` and runtime assertion checker `jmlrac` to validate the specification respectively implementation by at least 3 calls of each method; the calls shall include valid and (if possible) also invalid inputs. Finally use the extended static checker `escjava2` to further validate your implementation.

Please note that various of the given specifications/implementations contain ambiguities/errors. If you detect such, explain them, fix them and re-run your checks.

The deliverables of this exercise consist

- a nicely formatted copy of the JML-annoated Java code for each class,
- the output of running `jml -Q` on the class,
- the output of running `jmlrac` on the class,
- the output of running `escjava2` 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` 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 `escjava2` reports a warning, this does not necessarily mean that the program indeed violates its specification (only that the tool could not verify its correctness).

1. Specify the method

   ```
   public static int minimumPosition1(int[] a)
   ```

   that takes a non-null and non-empty integer array *a* and returns the position of the smallest element in the array.

2. Specify the method

   ```
   public static int minimumPosition2(int[] a)
   ```

   that takes an integer array *a* and returns the position of the smallest element in the array (-1, if the array is null or empty).

3. Specify the method

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

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

4. Specify the method

   ```
   public static int[] cut(int[] a, int p, int n)
   ```

   that takes a non-null array a, a position p and a length n and returns a copy of a with n elements starting at position p removed (p and n must be so that the right-open interval $[p \ldots p + n[$ describes valid positions in a, n may be also 0).

5. Specify the method

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

   that takes a non-null array a and values x and y and replaces in a all elements x by y.

6. Specify the method

   ```
   public static boolean subtract1(int[] a, int[] b)
   ```

   that takes two non-null arrays a and b that hold non-negative integer numbers and subtracts from every element of a the corresponding element of b; if the element would become negative, it is set to 0. The return value of the function indicates, if such a "truncation" has happened.

7. Specify the method

   ```
   public static boolean subtract2(int[] a, int[] b)
     throws TruncatedException
   ```

   that takes two non-null arrays a and b that hold non-negative integer numbers and subtracts from every element of a the corresponding element of b, unless the element would become negative. In that case, this (and all the following) elements remain unchanged and an exception e is thrown such that e`.pos` indicates the corresponding position of the array.