SINF2224 – Programming Methods
Assignment 2
2015–2016

Guillaume Maudoux Charles Pecheur

March 17, 2016

Assignment due Monday Apr 11, 2016

This assignment can be performed in **groups of two students**. Individual submissions are also accepted. Interaction between students and groups is allowed but plagiarism will not be tolerated. Contact *Guillaume Maudoux* (`guillaume.maudoux@uclouvain.be`) for any question regarding this assignment.

1 Subject

Priority Queue

The `Queue` class implements a bounded priority queue (of integers) as a sorted array. The objective of the mission is to completely specify the class in JML, including class invariants, and verify the specifications using ESC/JAVA2. In addition, you will have to modify the class `Queue` into a class `Set` that only keeps one copy of any value.

The queue has a fixed capacity (the size of the array) and keeps its elements sorted in increasing order. The main methods are `enqueue(n)`, which adds element `n` in its appropriate position, and `dequeue()`, which returns and removes the highest (hence last) element in the queue. A few additional methods are also provided, including a method `noDup()` that removes all duplicate elements in the queue.

The Java source code file `Queue.java` is provided. It provides the full source code for the `Queue` class, with concise natural language documentation. A test class `Test` is also provided, with a single static method `test(n)`

that exercises an instance of `Queue` in various ways and contains JML assertions. This class should pass ESC/JAVA2 verification too, without any modification.

Object-orientation is an important aspect of this mission. Make sure that all properties of the priority queue in itself are properly specified as class invariants rather than pre/post-conditions of each method.

2 Requirements

You will provide the JAVA code of class `Queue`, with JML specifications, meeting the following requirements:

1.1 The JML specifications will prevent generic run-time errors (null dereferences, array bound errors, etc).

1.2 The JML specifications will cover and correspond to all the provided natural language specifications. In particular, class invariants will be used to specify object properties.

1.3 The fully specified code will pass ESC/JAVA2 verification, *without loop induction*, for up to `-loop 3`.

1.4 The *unmodified* code of class `Test` will pass ESC/JAVA2 verification, *without loop induction*, for up to `-loop 3`.

In addition, you will provide the JAVA code of class `Set`, with JML specifications, meeting the following requirements:

2.1 Class `Set` will contain the same methods as class `Queue`, with the same specifications, except that a `Set` will keep only one copy of any value. In particular, `enqueue(n)` will not insert `n` if it already occurs in the set. `noDup()` becomes irrelevant and can be deleted.

2.2 The JML specifications will prevent generic run-time errors (null dereferences, array bound errors, etc).

2.3 Natural language and JML specifications will be modified accordingly and will correspond to each other.

2.4 The fully specified code will pass ESC/JAVA2 verification, *without loop induction*, for up to `-loop 3`.

No modified version of class `Test` will be taken into account. Any results obtained with a modified version of class `Test` will instead be reported and discussed as comments in class `Queue`.

3 Hints

- As for the first assignment, work incrementally. Start with generic runtime errors, then class invariants, then specific method specifications. The hardest part is the `noDup()` method.
- Use your judgement if you find that documentation is ambiguous or inaccurate. Verification will confirm whether your interpretation is consistent with the code.
- You need **not** use inductive verification of loop invariants (option `-loopsafe`) in this mission to avoid technical issues when verifying class invariants. This means that you need not specify all loop invariants either, which makes the specification work considerably lighter.
- Specifications can be expressed in terms of actual instance fields. Model variables and represents clauses are **not** needed; class invariants suffice.
- You may need to reinforce some method specifications to prove that the class invariants hold.
- You will have to exclude sharing of class fields using the `owner` ghost field.
- Note that some methods are purely functional and can be specified as such and used in specifications.
- For `noDup()`, you need to specify that, after execution, the queue (i) has no duplicates and contains (ii) *all* and (iii) *only* elements that were initially in the queue. Using ghost variables might be needed to verify (ii) and (iii). Use assumptions to exclude aliasing between (actual and/or ghost) variables that may compromise your proofs. As always, be cautious not to add inconsistent assumptions.
- Class invariants and ghost variables may interact in subtle ways, resulting in invariant violations. In the reported counterexample context, look for unintended aliasing between regular and ghost variables,

e.g. `(brokenObj<2>%0).(data:31.16) == this.(oldpos:172.10)`, and refine your specifications to exclude such aliasing.

4 Deliverables

Assignment deliverables will be returned as an archive file (`.zip` or `.gz`) whose base name is the last names of the students (e.g. `Pecheur_VanderMeulen.zip`, no accents please). Please submit your deliverables using the *Assignment* tool of iCampus. Late submissions will not be accepted. The archive should at least contain the following items:

- A fully documented version of `Queue.Java` with JML specifications and detailed comments.
- A fully documented version of `Set.Java` with JML specifications and detailed comments.
- The *unmodified* provided `Test.java`.
- a shell script that reproduces the reported results.

If other files are provided, their use will be explained in the report.

Your code is your report! Since the program code will be the main deliverable, a particular care should be taken to provide a good-quality, easy-to-read document that clearly describes your work. *Report all results and observations* as comments in the program. Include *complete identification* in the header (course, year, assignment, student names and sections, date). Provide *cleanly formatted and indented code* with lines no longer than 70 characters.

All active Jml specifications will be assumed to pass verification. Failed JML specifications should be commented out and flagged as failed, e.g.

```
\\@ \\ assert x < y;        \\ FAILS
```