

Advanced Aspects of Object-Oriented Programming (SS 2015)

Practice Sheet 8

Date of Issue: 09.06.15
Deadline: 16.06.15
(before the lecture as PDF via E-Mail)

Exercise 1 Abstraction

- Explain the necessity for JML's concept of *model fields*. Why are these especially important in the context of interface specifications?
- Specify the following Queue interface according to the „well-known behaviour“ of this data structure. *Hint: Look which model provided by JML may be appropriate.*

```
public interface Queue {
    Object peek() throws EmptyQueueException;
    Object dequeue() throws EmptyQueueException;
    void enqueue(Object item);
    boolean isEmpty();
    int size();
}

class EmptyQueueException extends Exception {}
```

- Take the interface with the specification and modify it, such that a) it declares a class Queue instead of an interface, b) implements all methods, c) relates the implemented methods to the specified model using depends and representation clauses.
- How would you rewrite model variables to assertions? How would represents clauses be translated?

Exercise 2 Behavioral Subtyping I

- Does JML allow specification inheritance for invariants?
- What is the meaning of the keyword "also" in JML specifications?
- Given the following code:

```
public class Parent {
    //@ requires i >= 0;
    //@ ensures \result >= i;
    int m(int i){ ... }
}

public class Child extends Parent {
    //@ also
    //@ requires i <= 0
    //@ ensures \result <= i;
    int m(int i){ ... }
}
```

What can the result of $m(0)$ be for a Child instance? Give a full specification of m for Child.

- Do we have behavioral subtyping for the following class hierarchies? Explain why (or why not). If not, give a code fragment, which shows that behavioral subtyping does not hold.

```

public class A {
    //@ invariant value >= 0;
    protected int value;

    /*@
     @ public behavior
     @ requires a >= 0;
     @*/
    public void set(int a){ value = a; }

    /*@
     @ public behavior
     @ ensures \result >= 0;
     @*/
    public int get(){ return value; }
}

public class B extends A {
    /*@
     @ public behavior
     @ requires a >= 10;
     @*/
    public void setLarge(int a){ value = a; }
}

```

```

public class C {
    protected int value = 1;

    /*@
     @ public behavior
     @ requires a > 0;
     @*/
    public void set(int a){ value = a; }

    /*@
     @ public behavior
     @ ensures \result > 0;
     @*/
    public int get(){ return value; }
}

public class D extends C {
    /*@
     @ also
     @ public behavior
     @ requires a > 10;
     @*/
    public void set(int a){ value = a; }

    /*@
     @ also
     @ public behavior
     @ ensures \result > 10;
     @*/
    public int get(){ return value; }
}

```

Does the situation change, if we add the following invariant to C:

```

//@ invariant value > 0;

```

```

public abstract class E {
    //@ public model int count;

    /*@
     @ public behavior
     @ ensures count == 0;
     @*/
    public abstract void reset();

    /*@
     @ public behavior
     @ ensures count > \old(count);
     @*/
    public abstract void increment();
}

public class F extends E {
    //@ private represents count <- value;
    private int value;

    public void reset() {
        value = 0;
    }

    /*@
     @ also
     @ public behavior
     @ ensures count == \old(count) + 1;
     @*/
    public void increment() {
        value += 1;
    }
}

```

Exercise 3 Behavioral Subtyping II

Take the annotated interface Queue of Exercise 1 b) and write a class ArrayQueue that implements it. Modify the specifications, such that they ensure that ArrayQueue is a behavioral subtype of Queue.