

Advanced Aspects of Object-Oriented Programming (SS 2015)

Practice Sheet 7 (Hints and Comments)

Exercise 1 Introduction to JML

- a) • Advantages
- Formal Specifications are precise, therefore they avoid the ambiguities of natural languages.
 - Formal Specifications can be executable. This allows a (semi-) automatical verification (→ theorem provers, tools like boogie, etc.).
 - Enforces “Think before you code”. The developer has to think about the precise meaning of the interfaces he uses, this allows to detect errors and problem quite early in the development process.
- Disadvantage
- Formal Specifications are more difficult to write than code, because they need more knowledge about mathematics, abstraction, etc.
 - Another language to learn.
 - Formal specifications can be longer than descriptions in natural languages or than the specified source.
 - Every method is written twice, once in den specification language and once in code.
 - Problem: How to decide, that a specification is complete/precise enough?

b) -

c) -

d) `class` Container {

```
    /*@ non_null @*/           // part of the class invariant, otherwise extractMin fails.
    int[] a;
    int n;

    /*@invariant 0 <= n && n <= a.length; // n-1 is a valid index in a[]

    /*@ requires input != null;
    @ assignable n, a;
    @ ensures n == input.length &&
    @ (\forall int i; 0 <= i && i < input.length; a[i] == input[i]) &&
    @ a.length == input.length;
    */
    Container( int[] input ){
        n = input.length;
        a = new int[n];
        System.arraycopy(input, 0, a, 0, n);
    }

    /*@ requires 0 < n;
    @ assignable n, a;
    @ ensures \result == (\min int i; 0 <= i && i < \old(n); \old(a[i]));
    */
    int extractMin() {
        int m = Integer.MAX_VALUE;
        int mindex = 0;
        /*@ maintaining m >= (\min int j; 0 <= j && j < n; a[j]);
        */
        for (int i = 0; i < n; i++) {
            if (a[i] < m) {
                mindex = i;
                m = a[i];
            }
        }
        n--;
        a[mindex] = a[n];
    }
}
```

```

        return m;
    }
}

```

e) Add $n < \text{old}(n)$ to the postcondition of `extractMin` or use a history constraint like constraint $n < \text{old}(n)$

f) Translate the documentation of the class `ByteArrayInputStream` directly into jml.

```

public class ByteArrayInputStream extends InputStream {
    protected /*@ spec_public non_null @*/ byte[] buf;
    protected /*@ spec_public @*/ int count;
    protected /*@ spec_public @*/ int mark;
    protected /*@ spec_public @*/ int pos;

    /*@ public invariant (count>=0 && count<=buf.length) &&
    @ (mark>=0 && mark<=count) &&
    @ (pos>=0 && pos<=count);
    @*/

    /*@ requires b!=null;
    @ assignable buf, pos, count, mark;
    @ ensures buf==b &&
    @ pos==0 &&
    @ count==b.length &&
    @ mark==0;
    @*/
    public ByteArrayInputStream(byte[] b);

    /*@ requires b!=null &&
    @ offset>=0 &&
    @ length>=0 &&
    @ assignable buf, pos, count, mark;
    @ ensures buf==b &&
    @ pos==offset &&
    @ count==(offset+length<buf.length) ? offset+length:buf.length &&
    @ mark==offset;
    @*/
    public ByteArrayInputStream(byte[] b,
                                int offset,
                                int length);

    /*@ requires true;
    @ ensures \result==(count-pos);
    @*/
    public /*@ pure @*/ int available();

    /*@ requires true;
    @ ensures true;
    @*/
    public void /*@ pure @*/ close() throws IOException;

    /*@ requires true;
    @ assignable mark;
    @ ensures mark==pos;
    @*/
    public void mark(int readAheadLimit);

    /*@ requires true;
    @ ensures \result==true;
    @*/
    public boolean /*@ pure @*/ markSupported();

    /*@ requires true;
    @ assignable pos;
    @ ensures available()==0 => (\result==-1 && pos==\old(pos)) &&
    @ available()>0 => ((\result==buf[pos] && pos==\old(pos)+1) &&
    @ \result>=0 && \result<=255);
    @*/
    public int read();

    /*@ requires b!=null &&
    @ off>=0 &&
    @ len>=0 &&
    @ (off+len)<b.length;
    @ assignable pos;

```

```

    @ ensures    available()==0 => (\result==-1 && pos==\old(pos)) &&
    @          available()>0  => ((\result==(len<available()) ? len:available() &&
    @              pos==\old(pos)+\result) &&
    @              (\forall int k; 0<k && k<\result; buf[off+k]==b[off+k]));
    @*/
public int read(byte[] b, int off, int len);

/*@ requires  true;
   @ assignable pos;
   @ ensures   pos==mark;
   @*/
public void reset();

/*@ requires  n>=0;
   @ assignable pos;
   @ ensures   \result==(available()<n) ? available():n &&
   @           pos==\old(pos)+\result;
   @*/
public long skip(long n);
}

```

Exercise 2 Pre- and Postcondition using Assertions

a)

```

b) public class Person {
    private String name;
    private int weight;

    public String toString() {
        assert name != null : "invariant:_name_!=_null";           // from non-null annotation
        assert !name.equals("") : "invariant:_!name.equals(\"\")"; // invariant
        assert weight >= 0 : "invariant:_weight_>=_0";           // invariant

        String result = "Person(\"" + name + "\", "
            + weight + ")";

        assert result != null : "result_of_toString_not_null";     // postcondition
        assert name != null : "invariant:_name_!=_null";           // from non-null annotation
        assert !name.equals("") : "invariant:_!name.equals(\"\")"; // invariant
        assert weight >= 0 : "invariant:_weight_>=_0";           // invariant

        return result;
    }

    public int getWeight() {
        assert name != null : "invariant:_name_!=_null";           // from non-null annotation
        assert !name.equals("") : "invariant:_!name.equals(\"\")"; // invariant
        assert weight >= 0 : "invariant:_weight_>=_0";           // invariant

        int result = weight;

        assert result == weight : "result_of_getWeight_not_the_weight"; // postcondition
        assert name != null : "invariant:_name_!=_null";           // from non-null annotation
        assert !name.equals("") : "invariant:_!name.equals(\"\")"; // invariant
        assert weight >= 0 : "invariant:_weight_>=_0";           // invariant

        return result;
    }

    public Person(String n) {
        // invariants may not yet hold.
        assert n != null && !n.equals("") : "name_not_valid";     // precondition

        name = n; weight = 0;

        assert n.equals(name)
            && weight == 0 : "wrong_initialization_of_Person";     // postcondition
        assert name != null : "invariant:_name_!=_null";           // from non-null annotation
        assert !name.equals("") : "invariant:_!name.equals(\"\")"; // invariant
        assert weight >= 0 : "invariant:_weight_>=_0";           // invariant
    }
}

```

```

public void addKgs(int kgs) {
    assert name != null : "invariant:_name_!=_null"; // from non-null annotation
    assert !name.equals("") : "invariant:_!name.equals(\\\"\\\")"; // invariant
    assert weight >= 0 : "invariant:_weight_>=0"; // invariant
    assert kgs >= 0 : "kgs_is_negative"; // precondition
    assert weight + kgs >= 0 : "resulting_weight_is_negative"; // precondition
    int old_weight = weight;
    int old_kgs = kgs;

    if (kgs >= 0) {
        weight += kgs;
    } else {
        throw new IllegalArgumentException();
    }

    assert weight == old_weight + old_kgs : "weight_not_set_correctly"; // postcondition
    assert name != null : "invariant:_name_!=_null"; // from non-null annotation
    assert !name.equals("") : "invariant:_!name.equals(\\\"\\\")"; // invariant
    assert weight >= 0 : "invariant:_weight_>=0"; // invariant
}
}

```

Exercise 3 Assertions for JML using Method Calls

a) `public class C {`

```

    private int f = 0;

    public void m(O o) {
        assert f >= 0 : "invariant";
        f++;

        assert f >= 0 : "invariant";
        o.notify();
        assert f >= 0 : "invariant";

        f--;
        assert f >= 0 : "invariant";
    }
}

```

- b) The invariant also has to hold when leaving method `m`. This is also the case when calling `o.notify()`. Therefore the invariant has to be checked before the call to `o.notify()` after incrementing. If the invariant would be `f == 0` the check would not be successful.