

Advanced Aspects of Object-Oriented Programming (SS 2009)

Practice Sheet 9

Date of Issue: 23.06.09
Deadline: 29.06.09
(until 10 a.m. as PDF via E-Mail)

Exercise 1 Behavioral Subtyping

- a) Does JML allow specification inheritance for invariants?
b) What is the meaning of the keyword "also" in JML specifications?
c) 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.

- d) 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;

    /*@
    @ 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 2 Verification of Java Programs

ESC/Java2 is an automated program verification tool (aka extended static checker) for (JML annotated) Java programs.

Try to specify the TaxPayer example (Part 1) from <http://www.cs.ru.nl/~erikpoll/Teaching/JML/>. Use the ESC/Java2 tool to statically verify that the implementation meets its specification. The extended static checker can be invoked as follows: `escj Taxpayer.java`