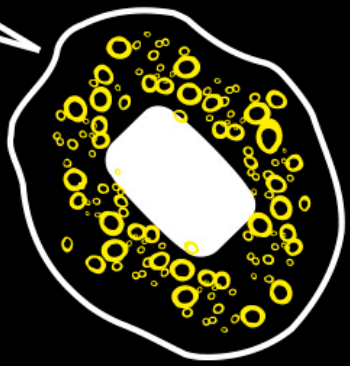
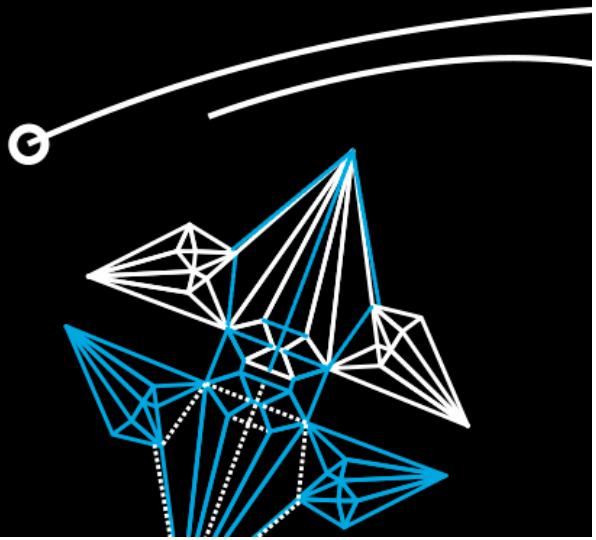
UNIVERSITY OF TWENTE.



Module 2 Programming Q&A CONTRACTS

Lecturer: Marieke Huisman



CONSIDER THIS STUB FOR CLASS STUDENT

```
public class Student {
            public static final int bachelor = 0;
            public static final int master = 1;

            private String name;
            private int credits;
            private int status;

            public void changeStatus(){
            }
            public void setName(String n) {
            }

            /*@ pure */ public String getName() {
                        return "";
            }
            /*@ pure */ public int getCredits() {
                        return 0;
            }
            /*@ pure */  public int getStatus() {
                        return 0;
            }
```

WHEN CAN CHANGESTATUS() BE CALLED?

```
/*@ requires getCredits() >= 180;
  @ requires getStatus() == bachelor;
  @ ensures getCredits() == \old(getCredits());
  @ ensures getStatus() == master;
*/
public void changeStatus(){
}
```

a. Always
b. Whenever the student has obtained 180 credits or more
c. For every bachelor student
d. When the student is a bachelor student, and has obtained 180 credits or more

WHAT DOES METHOD CHANGESTATUS() DO?

```
/*@ requires getCredits() >= 180;
  @ requires getStatus() == bachelor;
  @ ensures getCredits() == \old(getCredits());
  @ ensures getStatus() == master;
*/
public void changeStatus(){
}
```

a. Update the credits of the students
b. Change the status of the student to master
c. Status to master, credits unchanged
d. Status to master, credits updated

WHAT IS NOT SPECIFIED HERE?

```
/*@ requires getCredits() >= 180;
  @ requires getStatus() == bachelor;
  @ ensures getCredits() == \old(getCredits());
  @ ensures getStatus() == master;
*/
public void changeStatus(){
}
```

a. Students must always have 180 credits or more
b. After changeStatus(), a student always has 180 credits or more
c. changeStatus() will not change the credits
d. After changeStatus(), a student has master status

WHAT IS NOT EXPRESSED BY THESE INVARIANTS?

```
/*@ invariant getCredits() >= 0;
  @ invariant getStatus() == bachelor || getStatus() == master;
  @ invariant getStatus() == master ==> getCredits() >= 180;
*/
```

a. If number of credits is above 180, student is master student
b. If status is master, student must have 180 credits or more
c. Number of credits is always positive
d. Students is always bachelor or master student

WHAT IS CAPTURED BY THIS SPECIFICATION?

```
/*@ requires n != null;
  @ ensures getName().equals(n);
*/
public void setName(String n) {
}
```

a. name is always non-null
b. field name has same contents as string n
c. field name points to string n
d. name is a constant (and can never be changed)

WHAT IS A GOOD SPECIFICATION FOR CONSTRUCTOR?

```
public class Point {
    private int x;
    private int y;

    public Point(int n, int m) {
        x = n;
        y = m;
    }
    public int getX() { return x; }
    public int getY() { return y; }
}
```

a. requires x == 0 && y == 0;
b. requires x < n && y < m;
c. ensures x == n && y == m;
d. ensures x < n && y < m;

WHICH IMPLEMENTATION DOES NOT RESPECT THIS SPECIFICATION?

```
public int x;

/*@ requires n >= 0;
      ensures x >= \old(x);
*/
public void inc(n) {
   // body goes here
}
```

a. x = n;
b. x = x + n;
c. x = x + 1;
d. x = x + 2*n;

WHAT IS A CORRECT SPECIFICATION FOR *POWER*?

```
public int power(int e1, int e2) {
    int res = 1;
    int count = 0;
    if (e1 > 0) {
        while (count < e2) {
            res = res * e1;
            count = count + 1;
        }
        return res;
    } else {
        return 1;
    }}
```

a. ensures \result >= 1;

b. ensures \result == e1 + e2;

c. invariant res >= 0;

d. requires e1 == e2;

WHAT SPECIFICATION IS CHECKED BY THE ASSERTS?

```
public int compute (int m, int n) {
    assert m >= 0;
    assert n >= 0;
    return <complicated expression using m and n>
}
```

a. requires m >= 0 && n >= 0;
b. ensures m >= 0 && n >= 0;
c. requires m >= 0 || n >= 0;
d. ensures m >= 0 || n >= 0;

COUNTER CLASS EXAMPLE

```
public class Counter {

   /*@
    *    invariant value >= 0;
    */

…
   /*@ requires value >= 0;
   */
   public void setValue(int value) {
         this.value = value;
   }
```

```
/*@ ensures getValue() ==
          \old(getValue) + 1;
*/
public void increment() {
          value++;
   }
}
```

COUNTER CLASS EXAMPLE

```
public class Counter {

   /*@
    *    invariant getValue() >= 0;
    */

   private int value;

   public Counter() {
        value = 0;
   }

   public int getValue() {
         return value;
   }

   public void setValue(int value) {
         this.value = value;
   }

   public void increment() {
         value++;
   }
}
```

Public invariant: refers to publicly visible methods, documents class behaviour

OTHER QUESTIONS?