Software Engineering
Exercise 4

Prof. Dr. Peter Thiemann
Sergio Feo-Arenis Sergiy Bogomolov

Albert-Ludwigs-University Freiburg

June 26th, 2014

Exercise 1: Design by Contract

```
/* A stack with a fixed maximum capacity */
public class Stack<X>
{
    int topIx;             // index in content of the top element
    final X[] content;     // array that stores elements of the stack

    @Inv{topIx < content.length ∧ isEmpty()?topIx = −1 : topIx ≥ 0}

    @Pre{capacity > 0}
    public Stack(int capacity)
    {
        this.content = (X[]) new Object[capacity];
        this.topIx = 1;
    }
    @Post{isEmpty()∧!isFull()}
```

Exercise 1: Design by Contract

```
@Pre{!isEmpty()}
public X top()
{
    return this.content[this.topIx];
}

@Pre{!isEmpty()}
public X pop()
{
    X res = this.content[this.topIx];
    this.topIx  ;
    return res;
}
@Post{!isFull() ∧ old.top() = pop}
```

Exercise 1: Design by Contract

```
@Pre{!isFull()}
public void push(X x)
{
    this.topIx++;
    this.content[this.topIx] = x;
}
@Post{!isEmpty() ∧ top() == x}

public boolean isEmpty()
{
    return this.topIx == 1;
}
public boolean isFull ()
{
    return (this.topIx ==
            (this.content.length    1));
}
public static void main(String[] args) { ... }
```

Exercise 2: Contract Monitoring

We consider the different method calls, that occur in the body of the `main` method of class `Run`.

(i) `c.getLowerBound()`: Correct, assuming the implementation of class `IntegerInterval` adheres to the postcondition.

(ii) `c.getUpperBound()`: Correct, assuming the implementation of class `IntegerInterval` adheres to the postcondition.

(iii) For the contract:

```
c.changeContent( c.getLowerBound()+
(c.getUpperBound()-c.getLowerBound)*i/10 )
```

violations may occur!

(iii) Before method call: we know, that `i >= 0 && i <= 10`
Hence: `c.getLowerBound() <= arg <= c.getUpperBound()`
On **entry**, the following conditions must hold:

- $\mathrm{PRE}_{\mathrm{IntegerInterval,changeContent}} \Rightarrow \mathrm{PRE}_{\mathrm{IntegerInterval,changeContent}}$: obviously true.
- $\mathrm{PRE}_{\mathrm{IntegerInterval,changeContent}} \equiv$ `this.getLowerBound() <= arg < this.getUpperBound()`: Wrong, for `i == 10` because then arg evaluates to `this.getUpperBound()`.

(iv) `n.changeContent(-42)`: Contract violations may occur!

On **entry**, the following conditions must hold:

- $\mathsf{PRE}_{\mathsf{NegqativeIntegerInterval,changeContent}} \equiv$
  `this.getLowerBound() <= -(-42) < this.getUpperBound()`:
  Depends on the implementation of `getLowerBound()` and `getUpperBound()`, respectively.
- $\mathsf{PRE}_{\mathsf{IntegerInterval,changeContent}} \Rightarrow \mathsf{PRE}_{\mathsf{NegativeIntegerInterval,changeContent}}$:
  Usually does not hold, because
  `this.getLowerBound() <= i < this.getUpperBound()`
  not always implies
  `this.getLowerBound() <= -i < this.getUpperBound()`!

Exercise 3: Hoare Calculus
(i)

```
{ x >= 10, y >= 0 } y:= y+x { x >= 0, y >= 5 }
```

$$\frac{(x \geq 10 \land y \geq 0) \Rightarrow (x \geq 0 \land y + x \geq 5), \quad \{x \geq 0, y + x \geq 5\}\ \texttt{y:= y+x}\ \{x \geq 0, y \geq 5\}}{\{x \geq 10, y \geq 0\}\ \texttt{y:= y+x}\ \{x \geq 0, y \geq 5\}}$$

Exercise 3: Hoare Calculus
(ii)

$\{$ `true` $\}$ `if (a > b) m:= a else m:= b` $\{$ `m = max(a, b)` $\}$

$R \equiv m = max(a, b)$

$$\cfrac{\cfrac{a > b \Rightarrow a = max(a,b), \quad \{a = max(a,b)\}\ \texttt{m:= a}\ \{R\}}{\{a > b\}\ \texttt{m:= a}\ \{R\}} \qquad \cfrac{a \leq b \Rightarrow b = max(a,b), \quad \{b = max(a,b)\}\ \texttt{m:= b}\ \{R\}}{\{a \leq b\}\ \texttt{m:= b}\ \{R\}}}{\{true\}\ \texttt{if (a>b) m:=a else m:=b}\ \{R\}}$$

Exercise 3: Hoare Calculus
(iii)

```
{ A, i < n } i:= i+1; sum:= sum+i { A }
```

$A \equiv i \leq n \wedge sum = i(i+1)/2$
$R_1 \equiv i+1 \leq n \wedge sum + i + 1 = (i+1)(i+2)/2$
$R_2 \equiv i \leq n \wedge sum + i = i(i+1)/2$

$$\dfrac{\dfrac{\begin{array}{c}(A \wedge i < n) \Rightarrow R1, \\ \{R1\}\ \texttt{i:=i+1}\ \{R2\}\end{array}}{\{A, i < n\}\ \texttt{i:=i+1}\ \{R2\}} \qquad \{R2\}\ \texttt{sum:=sum+i}\ \{A\}}{\{A, i < n\}\ \texttt{i:=i+1; sum:= sum+i}\ \{A\}}$$

Exercise 3: Hoare Calculus

(iv)

```
{ n >= 0, sum=0, i=0 }
while (i<n)
{
i:= i+1;
sum:= sum+i
}
{ sum = n*(n + 1)/2 }
```

Exercise 3: Hoare Calculus

(iv)

We use the loop invariant $A \equiv i \leq n \wedge sum = i(i+1)/2$.

```
{ n >= 0, sum=0, i=0 } ⟹
{ i <= n, sum = i*(i + 1)/2 }
while (i<n)
  { i <= n, sum = i*(i+1)/2, i<n } ⟹
  { i < n, sum+i+1 = i*(i+1)/2 +(i+1) } ⟹
  { i < n, sum+i+1 = (i*i + 3*i + 2)/2 } ⟹
  { i+1 <= n, sum+i+1 = (i+1)*(i+2)/2 }
  i:= i+1;
  { i <= n, sum+i = i*(i + 1)/2 }
  sum:= sum+i
  { i <= n, sum = i*(i + 1)/2 }
{ i <= n, sum = i*(i + 1)/2, i>=n } ⟹
{ sum = n*(n + 1)/2 }
```

Exercise 3: Hoare Calculus
(v)

```
{ n >= 0 }
sum:= 0; i:= 0;
while (i<n) {i:= i+1; sum: = sum+i }
{ sum = n*(n+1)/2 }
```

From (iv) we have the partial correctness of:

$\{n \geq 0, sum = 0, i = 0\}$ `while(i<n) {i:= i+1; sum:= sum+i}`$\{A, i \geq n\}$

The partial correctness of:
$\{n \geq 0\}$ `sum:= 0; i:= 0` $\{n \geq 0, sum = 0, i = 0\}$ is easy to prove.

```
{ n >= 0 }
sum:= 0;
{ n >= 0, sum=0 }
i:= 0
{ n >= 0, sum=0, i=0 }
```