Introduction to Programming

Lab Session 3

(With material from the ETH Zurich course "Introduction to Programming")

September 6, 2016

In this Lab

- Understanding contracts: preconditions, postconditions (a second attempt)
- Exercises on class invariant.

Contracts

Precondition (from the previous lab)

Property that a feature imposes on clients:

Precondition (from the previous lab)

Property that a feature imposes on clients:

square_root (x: REAL): REAL

-- Returns the square root number of 'x'

Precondition (from the previous lab)

Property that a feature imposes on clients:

square_root (x: REAL): REAL

-- Returns the square root number of 'x'

require

x_positive: $x >= 0$

Precondition (from the previous lab)

Property that a feature imposes on clients:

square_root (x: REAL): REAL

-- Returns the square root number of 'x'

```
require
    x_positive: x >= 0
```

A feature without a **require** clause is always applicable, as if the precondition reads

```
require
    always_ok: True
```

Precondition principle

Precondition principle

A *client* calling a feature must make sure that the *precondition* holds before the call.

Precondition principle

Precondition principle

A *client* calling a feature must make sure that the *precondition* holds before the call.

A client that calls a feature without satisfying its precondition is faulty (*buggy*) software.

Postcondition (from the previous lab)

Property that a feature guarantees on termination

Postcondition (from the previous lab)

Property that a feature guarantees on termination

square_root (x: REAL): REAL

-- Returns the square root number of 'x'

Postcondition (from the previous lab)

Property that a feature guarantees on termination

```
square_root (x: REAL): REAL
            -- Returns the square root number of 'x'
    require
        x_positive: x >= 0
```

Postcondition (from the previous lab)

Property that a feature guarantees on termination

```
square_root (x: REAL): REAL
        -- Returns the square root number of 'x'
    require
        x_positive: x >= 0
    ensure
        result_value: x = Result * Result
```

Postcondition (from the previous lab)

Property that a feature guarantees on termination

```
square_root (x: REAL): REAL
            -- Returns the square root number of 'x'
    require
        x_positive: x >= 0
    ensure
        result_value: x = Result * Result
```

A feature without an **ensure** clause always satisfies its postcondition, as if the postcondition reads

```
    ensure
        always_ok: True
```

Postcondition principle

Postcondition principle

A feature must make sure that, if its precondition held at the beginning of its execution, its *postcondition* will hold at the end.

Postcondition principle

Postcondition principle

A feature must make sure that, if its precondition held at the beginning of its execution, its *postcondition* will hold at the end.

A feature that fails to ensure its postcondition is *buggy* software.

Precondition example (attempt 1)

Add pre- and postconditions to:

square_root (x: REAL): REAL

-- Returns the square root number of ‘x’.

Precondition example (attempt 1)

Add pre- and postconditions to:

square_root (x: REAL): REAL

```
        -- Returns the square root number of 'x'.
    do
        -- to implement later
    ...
    end
```

Precondition example (attempt 1)

Add pre- and postconditions to:

square_root (x: REAL): REAL

```
        -- Returns the square root number of 'x'.
    do
        -- to implement later
    ...
    end
```

The precondition is *True*

Precondition example (attempt 1)

Add pre- and postconditions to:

```
square_root (x: REAL): REAL
            -- Returns the square root number of 'x'.
    do
        -- to implement later
    ...
    end
```

The precondition is *True*: it is too weak. It accepts incorrect values (give me an example), together with all correct ones

Precondition example (attempt 1)

Add pre- and postconditions to:

square_root (x: REAL): REAL

```
        -- Returns the square root number of 'x'.
    do
        -- to implement later
    ...
    end
```

The precondition is *True*: it is too weak. It accepts incorrect values (give me an example), together with all correct ones: the precondition is *complete* and *unsound*.

Precondition example (attempt 2)

Add pre- and postconditions to:

square_root (x: REAL): REAL

-- Returns the square root number of ‘x’.

Precondition example (attempt 2)

Add pre- and postconditions to:

```
square_root (x: REAL): REAL
            -- Returns the square root number of 'x'.
    require
        correct_x: x > 10
    do
        -- to implement later
    ...
    end
```

Precondition example (attempt 2)

Add pre- and postconditions to:

```
square_root (x: REAL): REAL
            -- Returns the square root number of 'x'.
    require
        correct_x: x > 10
    do
        -- to implement later
    ...
    end
```

The precondition is too strong.

Precondition example (attempt 2)

Add pre- and postconditions to:

square_root (x: REAL): REAL

```
        -- Returns the square root number of 'x'.
    require
        correct_x: x > 10
    do
        -- to implement later
    ...
    end
```

The precondition is too strong. It accepts all correct values, but not all the possible correct values (give me an example)

Precondition example (attempt 2)

Add pre- and postconditions to:

```
square_root (x: REAL): REAL
            -- Returns the square root number of 'x'.
    require
        correct_x: x > 10
    do
        -- to implement later
    ...
    end
```

The precondition is too strong. It accepts all correct values, but not all the possible correct values (give me an example): the precondition is *incomplete* and *sound*.

Precondition example (attempt 3)

Add pre- and postconditions to:

square_root (x: REAL): REAL

-- Returns the square root number of ‘x’.

Precondition example (attempt 3)

Add pre- and postconditions to:

square_root (x: REAL): REAL

```
        -- Returns the square root number of 'x'.
    require
        x_positive: x >= 0
    do
        -- to implement later
        ...
    end
```

Precondition example (attempt 3)

Add pre- and postconditions to:

```
square_root (x: REAL): REAL
            -- Returns the square root number of 'x'.
    require
        x_positive: x >= 0
    do
        -- to implement later
        ...
    end
```

The precondition is *complete* and *sound*.

Postcondition example (attempt 1)

Add pre- and postconditions to:

```
square_root (x: REAL): REAL
            -- Returns the square root number of 'x'.
    require
        x_positive: x >= 0
    do
        -- to implement later
    end
```

Postcondition example (attempt 1)

Add pre- and postconditions to:

square_root (x: REAL): REAL

```
        -- Returns the square root number of 'x'.
    require
        x_positive: x >= 0
    do
        -- to implement later
    end
```

The postcondition is *True*

Postcondition example (attempt 1)

Add pre- and postconditions to:

```
square_root (x: REAL): REAL
            -- Returns the square root number of 'x'.
    require
        x_positive: x >= 0
    do
        -- to implement later
    end
```

The postcondition is *True*: it is too weak. It guarantees all the correct values, but not all of the guaranteed values are correct (give me an example)

Postcondition example (attempt 1)

Add pre- and postconditions to:

```
square_root (x: REAL): REAL
        -- Returns the square root number of 'x'.
    require
        x_positive: x >= 0
    do
        -- to implement later
    end
```

The postcondition is *True*: it is too weak. It guarantees all the correct values, but not all of the guaranteed values are correct (give me an example): the postcondition is *incomplete* and *sound*.

Postcondition example (attempt 2)

Add pre- and postconditions to:

```
square_root (x: REAL): REAL
            -- Returns the square root number of 'x'.
    require
        x_positive: x >= 0
    do
        -- to implement later
    ensure
        result_ok: Result >= 10
    end
```

Postcondition example (attempt 2)

Add pre- and postconditions to:

```
square_root (x: REAL): REAL
            -- Returns the square root number of 'x'.
    require
        x_positive: x >= 0
    do
        -- to implement later
    ensure
        result_ok: Result >= 10
    end
```

The postcondition is too strong

Postcondition example (attempt 2)

Add pre- and postconditions to:

```
square_root (x: REAL): REAL
        -- Returns the square root number of 'x'.
    require
        x_positive: x >= 0
    do
        -- to implement later
    ensure
        result_ok: Result >= 10
    end
```

The postcondition is too strong. does not guarantee all the correct values (give me an example), but the ones it does guarantee are correct

Postcondition example (attempt 2)

Add pre- and postconditions to:

```
square_root (x: REAL): REAL
        -- Returns the square root number of 'x'.
    require
        x_positive: x >= 0
    do
        -- to implement later
    ensure
        result_ok: Result >= 10
    end
```

The postcondition is too strong. does not guarantee all the correct values (give me an example), but the ones it does guarantee are correct : *complete* and *unsound*.

Postcondition example (attempt 3)

Add pre- and postconditions to:

```
square_root (x: REAL): REAL
            -- Returns the square root number of 'x'.
    require
        x_positive: x >= 0
    do
        -- to implement later
    ensure
        result_ok: Result * Result  = x
    end
```

Postcondition example (attempt 3)

Add pre- and postconditions to:

```
square_root (x: REAL): REAL
            -- Returns the square root number of 'x'.
    require
        x_positive: x >= 0
    do
         -- to implement later
    ensure
        result_ok: Result * Result  = x
    end
```

The postcondition *complete* and *sound*.

(Un)sound and (in)complete pre/post-conditions

Let P and Q be two assertions.

(Un)sound and (in)complete pre/post-conditions

Let P and Q be two assertions.
If a feature needs P, then:

- if $Q \Rightarrow P$, then Q is said to be a *sound* precondition; otherwise Q is said to be an *unsound* precondition;
- if $P \Rightarrow Q$, then Q is said to be a *complete* precondition; otherwise Q is said to be an *incomplete* precondition.

(Un)sound and (in)complete pre/post-conditions

Let P and Q be two assertions.
If a feature needs P, then:

- if $Q \Rightarrow P$, then Q is said to be a *sound* precondition; otherwise Q is said to be an *unsound* precondition;
- if $P \Rightarrow Q$, then Q is said to be a *complete* precondition; otherwise Q is said to be an *incomplete* precondition.

If a feature ensures P, then:

- if $P \Rightarrow Q$, then Q is said to be a *sound* postcondition; otherwise Q is said to be an *unsound* postcondition;
- if $Q \Rightarrow P$, then Q is said to be a *complete* postcondition; otherwise Q is said to be an *incomplete* postcondition.

Example

```
feature
    f (x: REAL): REAL
        require
            x >= 0
        do
            Result := x.power (2)
        ensure
            Result >= 0
        end
```

Then:

Example

```
feature
    f (x: REAL): REAL
        require
            x >= 0
        do
            Result := x.power (2)
        ensure
            Result >= 0
        end
```

Then:

- the assertion $x > 0$ is

Example

```
feature
    f (x: REAL): REAL
        require
            x >= 0
        do
            Result := x.power (2)
        ensure
            Result >= 0
        end
```

Then:

- the assertion $x > 0$ is a *sound* and *incomplete* precondition and an *unsound* and *complete* postcondition;

Example

```
feature
    f (x: REAL): REAL
        require
            x >= 0
        do
            Result := x.power (2)
        ensure
            Result >= 0
        end
```

Then:

- the assertion $x > 0$ is a *sound* and *incomplete* precondition and an *unsound* and *complete* postcondition;
- the assertion $x > -5$ is

Example

```
feature
    f (x: REAL): REAL
        require
            x >= 0
        do
            Result := x.power (2)
        ensure
            Result >= 0
        end
```

Then:

- the assertion $x > 0$ is a *sound* and *incomplete* precondition and an *unsound* and *complete* postcondition;
- the assertion $x > -5$ is an *unsound* and *complete* precondition and a *sound* and *incomplete* postcondition;

Example

```
feature
    f (x: REAL): REAL
        require
            x >= 0
        do
            Result := x.power (2)
        ensure
            Result >= 0
        end
```

Then:

- the assertion $x > 0$ is a *sound* and *incomplete* precondition and an *unsound* and *complete* postcondition;
- the assertion $x > -5$ is an *unsound* and *complete* precondition and a *sound* and *incomplete* postcondition;
- the assertion $x < -5$ is

Example

```
feature
    f (x: REAL): REAL
        require
            x >= 0
        do
            Result := x.power (2)
        ensure
            Result >= 0
        end
```

Then:

- the assertion $x > 0$ is a *sound* and *incomplete* precondition and an *unsound* and *complete* postcondition;
- the assertion $x > -5$ is an *unsound* and *complete* precondition and a *sound* and *incomplete* postcondition;
- the assertion $x < -5$ is an *unsound* and *incomplete* precondition and an *unsound* and *incomplete* postcondition;

Example

```
feature
    f (x: REAL): REAL
        require
            x >= 0
        do
            Result := x.power (2)
        ensure
            Result >= 0
        end
```

Then:

- the assertion $x > 0$ is a *sound* and *incomplete* precondition and an *unsound* and *complete* postcondition;
- the assertion $x > -5$ is an *unsound* and *complete* precondition and a *sound* and *incomplete* postcondition;
- the assertion $x < -5$ is an *unsound* and *incomplete* precondition and an *unsound* and *incomplete* postcondition;
- the assertion $x \geq 0$ is

Example

```
feature
    f (x: REAL): REAL
        require
            x >= 0
        do
            Result := x.power (2)
        ensure
            Result >= 0
        end
```

Then:

- the assertion $x > 0$ is a *sound* and *incomplete* precondition and an *unsound* and *complete* postcondition;
- the assertion $x > -5$ is an *unsound* and *complete* precondition and a *sound* and *incomplete* postcondition;
- the assertion $x < -5$ is an *unsound* and *incomplete* precondition and an *unsound* and *incomplete* postcondition;
- the assertion $x \geq 0$ is a *sound* and *complete* precondition and a *sound* and *complete* postcondition.

Class invariant

Property that is true of the current object at any *observable* point

Class invariant

Property that is true of the current object at any *observable* point

class *BANK_ACCOUNT*

...

balance: INTEGER

-- 'balance' of the account

...

Class invariant

Property that is true of the current object at any *observable* point

class *BANK_ACCOUNT*

...

balance: INTEGER

-- 'balance' of the account

...

invariant

balance_no_negative: *balance >= 0*

Class invariant

Property that is true of the current object at any *observable* point

class *BANK_ACCOUNT*

...

balance: INTEGER

-- 'balance' of the account

...

invariant

balance_no_negative: *balance* >= *0*

A class without an **invariant** clause has a trivial invariant

invariant

always_ok: **True**

Class invariant principle

Class invariant principle

A *class invariant* must hold as soon as an object is created, then before and after the execution of any of the class features available to its clients.

Class invariant principle

Class invariant principle

A *class invariant* must hold as soon as an object is created, then before and after the execution of any of the class features available to its clients.

A class that fails to ensure its invariants is *buggy* software.

Exercises

Exercises can be found in: https://drive.google.com/open?id=0B1GMHm59JFjqY0ZFaUFJLU5IeEE.

Thank you!