Design by contract

CSCU9P5 - Software Engineering I



CSCU9P5 - Software Engineering I

Pre- and post-conditions

- We know the basic principle that a class has
 - a public interface
 - and a hidden implementation.
- A client can see the offered operations:
 - their required parameters
 - and the type of the returned value (if any)
 - ie. the public interface of the class
 - but not how the operations are carried out, i.e. the implementation.
- Ideally, the interface should also define what each operation does:
 - ie. provide information about the semantics of the offered operation
 - but not how it does it.

Computing Science & Mathematics University of Stirling

CSCU9P5 - Software Engineering I

Pre- and post-conditions

- An abstract way of expressing the expected behaviour/results of a given operation is to describe
 - the hypothesis which the execution of the operation relies upon, and
 - the effects that the operation causes without providing details on the actual implementation of the operation.
- We associate with each operation a pre-condition and a postcondition:
 - If the pre-condition is true when an operation is called then, after execution of the operation, the post-condition must be true.
 - If an operation is called when its pre-condition is false, the
 post-condition says nothing about the effect of the operation.

Computing Science & Mathematics University of Stirling

CSCU9P5 - Software Engineering I

.

Pre- and post-conditions

- We will just consider simple Boolean expressions to represent preand post-condition constraints
- UML has an associated language called OCL (Object Constraint Language) in which constraints can be represented formally.
- An example of a pre-condition is that before we call a withdraw operation on Account to withdraw amount, the pre-condition:

pre: amount <= balance</pre>

should be true.

Computing Science & Mathematics
University of Stirling

CSCU9P5 - Software Engineering I

Pre- and post-conditions

 An example of a pre-condition is that before we call a withdraw operation on Account to withdraw amount, the pre-condition:

```
pre: amount <= balance</pre>
```

should be true.

• If that is the case, then after execution of the operation, the following post-condition must be true:

```
post: balance == old balance - amount
```

- If the pre-condition is false then the withdraw operation is "undefined".
- No information about implementation of withdraw operation has been given.



CSCU9P5 - Software Engineering I

.

Class invariant

- As well as pre- and post-conditions, we can have class invariants.
- A class invariant is a property that must hold throughout the life line of (an object of) a class - by design
- The property must be true when the class is instantiated
- If the property is true **before** an operation is called then it must be true **after** the operation is called
 - During the call the property might *not* be **true**, but must become **true** by the end of the operation.
- Therefore the property will be true whenever no operation is in progress
 - (If there is no interference)



CSCU9P5 - Software Engineering I

Class invariant

- Pre- and post-conditions and class invariants are very useful at giving extra information to our models
 - mainly clarification of design decisions and
 - constraints on admissible behaviour.



CSCU9P5 - Software Engineering I

Class invariant

A possible class invariant on Account is:

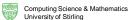
```
balance >= 0
```

That is: throughout the life of an Account object, the balance is never allowed to go negative.

- · Check:
 - withdraw OK?

Yes: if invariant is true, and precondition is true, and post-condition is true, then invariant is true

- depositCash OK?
Yes, provided we have pre-cond amount > 0

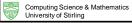


CSCU9P5 - Software Engineering I

.

Class invariant

- Remember: it may be the case that a class invariant will become false during the execution of an operation
 - But the operation must ensure that it becomes **true** again before the operation terminates.
- For instance, a possible (may be not ideal) implementation of the withdraw operation could:
 - Subtract the requested amount,
 - Only subsequently check that the balance is not negative ...
 - ...and, if it is, cancel the operation and restore the original balance.
 - This would fulfill the class invariant



CSCU9P5 - Software Engineering I

.

Class invariant

- For invariants to be really useful, of course, operations need to be checked to ensure that the invariants are satisfied.
- That is not yet part of UML tools and is not part of most programming languages.
- Some programming languages, e.g. Java and Eiffel, have assertions that allow these conditions to be checked at run time.
- The question of course remains:
 What do you do if an assertion fails?



 ${\sf CSCU9P5-Software\ Engineering\ I}$

Inheritance and conditions

• Remember that one principle of the object-oriented approach is

An object of a subclass can be used anywhere

an object of its superclass is expected.

- That means that the subclass must be able to fulfill the contract entered into by the superclass
 - The form of the contract is:

If a client calls an operation when its pre-condition is true, then the operation guarantees to satisfy its post-condition

- Potentially a problem if subclass overrides a method
- The catchy (if rather trite) requirement for a subclass is:

 Demand no more; promise no less.



CSCU9P5 - Software Engineering I

11

Inheritance and conditions

- **Demand no more** means that an operation offered by a subclass must be willing to accept all parameter values that the superclass would allow.
 - What is this in terms of pre-conditions?
- The pre-condition in the subclass must be **no stronger** (= **no more restrictive**) than the pre-condition in the superclass, e.g.

```
pre: 10 <= amount <= balance
```

- for a withdraw operation in a SavingsAccount ...
- ... is not ok, as it restricts the range of values for which the operation is defined.
- A client of Account could not safely call withdraw using its
 (apparent) pre-condition, as the actual object might be a
 SavingsAccount and the amount withdrawn might violate the
 overriding method's pre-condition.



CSCU9P5 - Software Engineering I

Inheritance and conditions

- Promise no less means that any assumption about the result that
 was valid when the superclass implementation was being used must
 still be valid when the subclass implementation is used.
 - What is this in terms of post-conditions?
- The post-condition in the subclass must be at least as strong as
 (= as or more restrictive than) the post-condition in the
 superclass, e.g.

post: balance >= old balance - amount

- for a withdraw operation in a SavingsAccount is not ok, as it expands the range of possible values in the account after the execution of the operation.
- Note that pre- and post-conditions give us a way of specifying that a subclass offers the same behaviour as the superclass, i.e. that we have behavioural inheritance.



CSCU9P5 - Software Engineering I

13

Design by contract

(what do you do if an assertion fails?)

- Pre- and post-conditions can be used to support the idea of design by contract
- Consider the situation where a withdraw method is called:
 - Whose responsibility is it that the account has sufficient funds?
 - Should the method have to check that it has been called correctly? Or should the *client* check?
- Design by contract is an approach where it is considered an error to call a method when its pre-condition is false:
 - It is the *responsibility* of the **client** to check the pre-condition
 - The operation need not check
- The opposite is defensive programming:
 - Each operation's implementation explicitly checks that its precondition is satisfied



CSCU9P5 - Software Engineering I

Design by contract

- To summarise, a class makes a **contract** with its clients that guarantees that when one of its operations is called with the pre-condition true then, after execution of the operation, the post-condition is guaranteed to be **true**.
- If an operation is called when its pre-condition is false, the class guarantees nothing about the effect of the operation.
- The idea is that one side will make the check, not neither or both.



CSCU9P5 - Software Engineering I

15

Design by contract

- You might not be happy with the use of design by contract:
 - We are effectively saying that a class definition can ignore difficult situations
 - (and e.g. Integer.parseInt does not assume its parameter is digits)
- It is of most use when you specify or design a system as a set of collaborating classes:
 - You are then making a decision within the collaboration about which class is responsible for each check
- But there are then dangers if one of the classes is used in a different environment in which the designer does not fully realise
 - the implication of the pre-condition and
 - that the client must perform the necessary checks.



CSCU9P5 - Software Engineering I

