Design by contract

• Object-Oriented Software Construction by Bertrand Meyer, Prentice Hall
• The presence of a precondition or postcondition in a routine is viewed as a contract.

Rights and obligations

• Parties in the contract: class and clients
• require pre, ensure post with method r: If you promise to call r with pre satisfied then I, in return, promise to deliver a final state in which post is satisfied.
• Contract: entails benefits and obligations for both parties
Rights and obligations

- Precondition binds clients
- Postcondition binds class

Example

<table>
<thead>
<tr>
<th>Contract for push of class Stack</th>
<th>Obligations</th>
<th>Benefits</th>
</tr>
</thead>
<tbody>
<tr>
<td>Client Programmer</td>
<td>Only call push(x) on a non-full stack</td>
<td>Get x added as new stack top on return (top yields x, nb_elements increased by 1)</td>
</tr>
<tr>
<td>Class Implementor</td>
<td>Make sure that x is put on top of stack</td>
<td>No need to treat cases in which the stack is already full</td>
</tr>
</tbody>
</table>
If precondition is not satisfied

- If client’s part of the contract is not fulfilled, class can do what it pleases: return any value, loop indefinitely, terminate in some wild way.
- Advantage of convention: simplifies significantly the programming style.

Source of complexity

- Does data passed to a method satisfy requirement for correct processing?
- Problem: no checking at all or: multiple checking.
- Multiple checking: conceptual pollution: redundancy; complicates maintenance
- Recommended approach: use preconditions
Class invariants and class correctness

- Preconditions and postconditions describe properties of individual routines
- Need for global properties of instances which must be preserved by all routines
- \(0 \leq \text{nb\_elements}; \text{nb\_elements} \leq \text{max\_size}\)
- \(\text{empty} = (\text{nb\_elements} = 0)\)

Class invariants and class correctness

- A class invariant is an assertion appearing in the invariant clause of the class.
- Must be satisfied by all instances of the class at all “stable” times (instance in stable state):
  - on instance creation
  - before and after every remote call to a routine (may be violated during call)
Class invariants and class correctness

- A class invariant only applies to public methods; private methods are not required to maintain the invariant.

Invariant Rule

- An assertion I is a correct class invariant for a class C iff the following two conditions hold:
  - The constructor of C, when applied to arguments satisfying the constructor’s precondition in a state where the attributes have their default values, yields a state satisfying I.
  - Every public method of the class, when applied to arguments and a state satisfying both I and the method’s precondition, yields a state satisfying I.
Invariant Rule

- Precondition of a method may involve the initial state and the arguments
- Postcondition of a method may only involve the final state, the initial state (through old) and in the case of a function, the returned value.
- The class invariant may only involve the state

Invariant Rule

- The class invariant is implicitly added (anded) to both the precondition and postcondition of every exported routine
- Could do, in principle, without class invariants. But they give valuable information.
- Class invariant acts as control on evolution of class
- A class invariant applies to all contracts between a method of the class and a client
Definitions

- Class C
- INV class invariant
- method r: pre_r(x_r) precondition; post_r postcondition
- x_r: possible arguments of r
- B_r: body of method r
- Default_C: attributes have default values

Correctness of a class

- A class C is said to be correct with respect to its assertions if and only if
  - For every public method r other than the constructor and any set of valid arguments x_r:
    {INV and pre_r(x_r)} B_r {INV and post_r}
  - For any valid set of arguments x_C to the constructor:
    {Default_C and pre_C(x_C)} B_C {INV}
How to prove correctness

• A complex story

Verifiable Programming

• Reason about imperative sequential programs such as Java programs
• Imperative program
  – defines state space
  • defined by collection of typed program variables
  • are coordinate axis of state space
  – pattern of actions operating in state space

The End