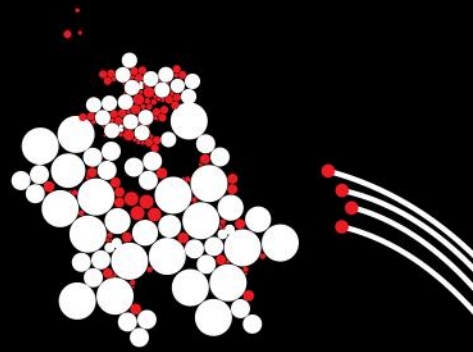
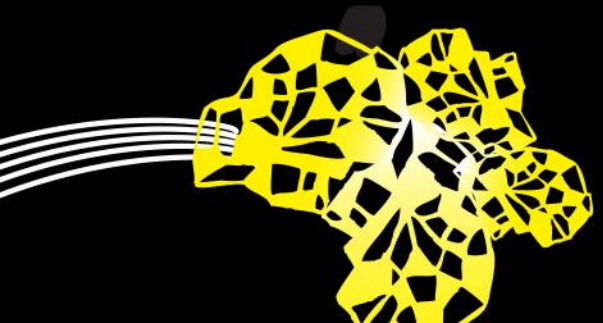
UNIVERSITY OF TWENTE.



Design by Contract

Topic of Software Systems (TCS module 2)

Lecturer: Marieke Huisman



DESIGN BY CONTRACT

PROGRAMMING DISCIPLINE WITH PRE-, POSTCONDITIONS AND INVARIANTS

- Methods have preconditions that caller (client) must respect
- Implementation can rely upon preconditions and class invariants
- If caller respects preconditions then method implementation and class invariant guarantee postconditions
- Implementation should ensure that class invariant is preserved

Enables separation of concerns
Caller only needs to look at specification

Design by Contract
First proposed by Bertrand Meyer (1986)

DESIGN BY CONTRACT

PROGRAMMING DISCIPLINE WITH PRE-, POSTCONDITIONS AND INVARIANTS

Problem: Can the caller (client) be trusted?

- What if caller does not respect the precondition?
  - Method will not guarantee postcondition and/or invariant
  - Next methods may be called while invariant is violated
  - Program does not behave properly, error hard to find

APPROACH 1: TRUST CLIENT

Client will always respect preconditions

Consequences

No special precautions necessary!

Only justified when client and object (class) are developed together

APPROACH 2: GENERATE ERROR MESSAGE

- Client will not always respect preconditions
- When this happens, program should stop, but in controlled manner

Consequences

- Implementation checks (some) preconditions
- assert precondition: stop program when precondition not respected

needs to be enabled in the JVM

APPROACH 3: DEFENSIVE PROGRAMMING

- Client will make mistakes (might be on purpose)
- Program should not fail

Consequences

- Implementation checks all preconditions, and if a precondition is not respected take appropriate emergency solution
  - Set default values, throw exceptions
  - Postcondition and invariant always respected
- Useful for critical applications

ANSWER 4: CHECK OR VERIFY

Runtime Checking

Automatically insert precondition and postcondition checks during execution

Static Checking

Construct formal proof that

- Preconditions hold at every method call
- Postconditions hold at every method exit
- Invariants are always maintained

Preferably with appropriate tooling

Not considered further in this module

DESIGN BY CONTRACT AND APIS

APIs (Application Programming Interfaces) follow the design by contract principles, sometimes in a less systematic way

return is valid double value

postcondition

s != null && proper format

precondition

valueOf

```
public static Double valueOf(String s) throws NumberFormatException
```

Returns a `Double` object holding the `double` value represented by the argument string `s`.

If `s` is `null`, then a `NullPointerException` is thrown.

Leading and trailing whitespace characters in `s` are ignored. Whitespace is removed as if by the `String.trim()` method; that is, both ASCII space and control characters are removed. The rest of `s` should constitute a *FloatValue* as described by the lexical syntax rules:

FloatValue:

$Sign_{opt}$ `NaN`

$Sign_{opt}$ `Infinity`

SUMMARY

- Behaviour of methods can be (precisely) specified
  - Precondition: what should hold when method is called
  - Postcondition: what implementation guarantees when method finishes
  - Invariant: property that holds throughout lifetime of object
- Specifications can be checked during execution
  - Insert checks manually (e.g., using `assert` statements)
  - Use dedicated tool support