

Language, and Formal Specification

- Assertions: constraints of classes
 - Preconditions: must be true before a method is executed
 - Postconditions: must be true after a method is executed
 - Invariants: must be true both before and after all method executions in a class
 - “Design by Contract”
- Object Constraint Language (OCL)
 - A formal, textual notation for constraints in UML models
 - Optional part of UML
 - Often used to express assertions
 - Predicate logic, sets
 - Expressions without side-effects

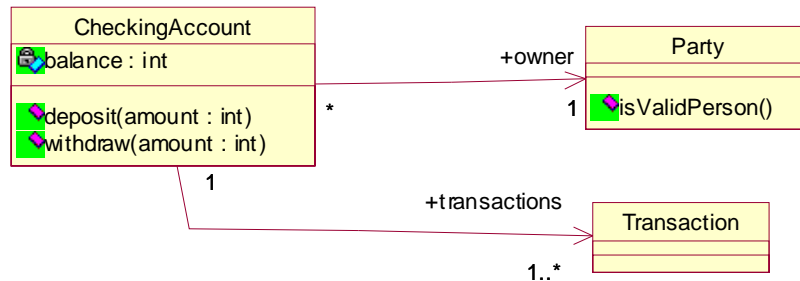
497-5-66

Design by Contract

- Contract between a method and its caller
 - If you fulfill my precondition,
I promise to fulfill my postcondition
 - using pre- and postconditions to specify a method
 - partial or complete specification
- Also useful for
 - reusing a library that someone else wrote
 - making sure it does what you think it does

497-5-67

Assertion Example: Checking Account



497-5-68

OCL Example

```
context CheckingAccount:: deposit( amount: Integer )
  pre: amount > 0
  post: balance = balance@pre + amount
context CheckingAccount:: withdraw( amount: Integer )
  pre: amount > 0 and ( amount <= balance )
  post: balance = balance@pre - amount
context CheckingAccount
  inv: balance >= 0
  inv: owner. isValidPerson()
  inv: ( transactions -> size() ) >= 1
```

- Syntax Elements: context declaration, types, expressions, attributes and operations, "@pre" postfix

497-5-69

OCL

- Context is necessary only when constraint is listed outside of a class diagram
- Dot operator (".") for accessing attributes and operations of objects
 - also: objects connected through associations
 - "self": name of the current object
`self.balance >= 0`
- Arrow operator ("->") for accessing operations of collections
 - the result of associations with multiplicity > 1 is always a set
 - collection operations can be used on objects too: each object is a set with one element (itself)
`owner -> size() = 1`

497-5-70

Some Predefined OCL Types

- Boolean
 - and, or, not, xor
- Integer
 - +, *, -, /, abs()
- Real
 - +, *, -, /, floor()
- String
 - toUpper(), concat()
- Collection
 - isEmpty(): Boolean, size(): Integer, forAll(boolean-expression): Boolean

497-5-71

Assertion Example: Stack

- Stack with a maximum size
 - pop(): Object
 - push(Object)
 - empty(): boolean
 - attributes: top, maxSize, elements
- Preconditions and postconditions?
 - for pop?
 - for push?
 - for empty?
- Class invariants?

497-5-72

Stack in Java

```
class Stack< E > {
    Stack( int maxSize );

    /** Add element to your top. */
    void push( E element );

    /** Remove and return the element at your top. */
    E pop();

    /** Return whether you are empty. */
    boolean isEmpty();

    private E[] elements;
    private int top; //index of the highest element
}
```

Preconditions? Postconditions? Invariants?

Stack Assertions

```
context Stack
  inv: (top >= 0) and (top <= elements. length)
context Stack:: pop(): E
  pre: top > 0
  post: top = top@pre - 1
context Stack:: push( element: E )
  pre: top < maxSize
  post: top = top@pre + 1
context Stack:: empty(): Boolean
  post: result = ( top = 0 )
```

- Is this specification complete?
- Is there implementation bias?

497-5-74

Pre- and Postconditions

- Can (in some cases) specify a method completely
 - CheckingAccount example
 - pre- and postconditions are a complete formal specification of these methods
 - theoretically, an implementation can automatically be generated
 - compare: logical programming (Prolog)
- Generally: incomplete specification
 - Stack example
 - property that pop() returns the parameter of the last non-popped push(): hard to specify in OCL!
 - cause: no assignment possible
 - can be specified if referring to underlying implementation (e.g. array): but that's not good (implementation bias)

497-5-75

Assertions with Inheritance

- What happens to assertions in subclasses?
- A inherits from B
 - preconditions can only become weaker
 - postconditions can only become stronger
- Remember: substitutability

497-5-76

Assertions in Design and Implementation

- Used in to design to prepare implementation
 - implementation needs to adhere to constraints
 - Some assertions may be replaced by exceptions
 - throw an exception if a precondition is not met
 - Programming language support for assertions
 - C++: `assert(int)`
 - in `assert.h`
 - Java
 - “assert” keyword
 - throws `AssertionError` if parameter is false
 - has to be enabled with a command line parameter (`-ea`)
- ```
double y = squareRoot(x);
assert(Math. abs(x - (y * y)) < 0.001) :
 "squareRoot too imprecise";
```
- An “assert()” function can easily be implemented in any language

497-5-77

## Using Assertions

- Use assertions when...
  - Your method is easily checked
    - Checking the result takes less effort than calculating it
    - Efficient: checking result of square root
    - Not efficient: checking the result of a sort is not efficient
  - You aren't sure if a condition holds in your program
    - you should probably find out!
  - You want to make sure you comply to a specification
    - low-cost formal methods
- Switch assertions off when...
  - when your program is sufficiently tested
  - when you can't tolerate the performance overhead
  - when you don't want to get embarrassed

497-5-78

## Formal Specification

- Based on formal logic or mathematics
  - propositional logic
  - predicate logic
  - set theory
- Examples
  - OCL
  - state diagrams
  - programming languages, assertions
- Advantages of formal specifications
  - precise
  - can be analyzed with tools
    - automatic checkers may be able to find inconsistencies
    - but: undecidability of predicate logic
  - claim: result in more reliable software
- Disadvantages of formal specification
  - hard to learn
  - hard to use

497-5-79

# Conclusions on Analysis and Specification

- Natural Language
  - informal semantics
  - informal syntax
- UML and similar notations
  - informal semantic
  - formal syntax
- Formal specification languages (such as OCL)
  - formal semantics
  - formal syntax
- Specification
  - saying **what** needs to be done
  - not **how** it is done
  - qualities of requirements apply

497-5-80