Department of Computer Science

KATHOLIEKE UNIVERSITEIT LEUVEN

Nominal Programming

- Class invariants
- Preconditions
- Black-box Testing

Assignment

KATHOLIEKE UNIVERSITEIT LEUVEN

- Develop a class of oil tanks in which exceptional cases are handled with the paradigm of nominal programming
  - An inspector returning the capacity of an oil tank
  - An inspector returning the amount of oil stored in an oil tank
  - A constructor initializing a new oil tank with given contents and given capacity
  - Methods to add, respectively to extract a given amount of oil to, respectively from an oil tank
  - Methods to fill an oil tank completely with oil, respectively to extract all the oil from an oil tank
  - Methods to add, respectively to extract a series of successive amounts of oil to, respectively from an oil tank
  - A method to check whether an oil tank is relatively more filled than another oil tank
- The definition of the class must be complemented with a black-box test for each of its methods

Task 1

Overview

KATHOLIEKE UNIVERSITEIT LEUVEN

- Class invariants
  - Specify restrictions that must be satisfied by classes or by individual objects at stable times
    - Class invariants are both rights and duties for users and implementers
- Nominal Programming
  - Specify conditions that must hold upon entry to a method in terms of preconditions
    - Effects of methods are still expressed in terms of postconditions
- Verification
  - Work out a collection of tests to verify the correctness of the class
    - Develop a consistent set of tests for each method
- Method Definition: Advanced Topics
  - Method overloading and methods with variable number of arguments
- Epilogue

Class Invariants

KATHOLIEKE UNIVERSITEIT LEUVEN

- Class invariants impose restrictions on the state of the class itself and/or on the state of its objects
  - Class invariants are worked out in the heading of the class
    - Each class invariant starts with the non-standard tag “@invar”
  - Class invariants are specified both informally in some natural language, and formally using first-order logic and set theory
    - Formal specifications are used in this course to learn how to work out documentation of software systems in an accurate way
- The language used for the formal specification of class ingredients is not precisely defined
  - Assertions are kept as close as possible to plain Boolean Java expressions

Raw Objects

- Objects involved in the definition of methods can be annotated @Raw
  - In the raw state, an object must not satisfy all its class invariants
    - New objects may not satisfy all invariants from the very start
    - Objects may violate invariants during mutations
  - A raw annotation of an instance method indicates that the prime object may be in the raw state upon entry to the method
    - Raw annotations do not apply to static methods
  - A raw annotation of a formal argument indicates that the actual object may be in the raw state upon entry to the method
    - Raw annotations do not apply to formal arguments of primitive type
  - Upon exit from a method, raw objects must not satisfy their class invariant
    - All other objects in an application must satisfy all their invariants upon entry to a method, and again upon exit from that method

Design by Contract

- The specification of a class is a contract between its implementers and its users
  - Users must see to it that all ordinary objects involved in a method satisfy their class invariants upon entry (duty)
    - Implementers may assume that all ordinary objects satisfy their class invariants upon entry (right)
  - Implementers must see to it that all ordinary objects involved in a method satisfy their class invariants upon exit (duty)
    - Users may assume that all ordinary objects satisfy their class invariants upon exit (right)
  - Implementers must see to it that a method has achieved all its postconditions upon exit (duty)
    - Each time a user invokes a method, he/she may assume that the method has achieved all its stated effects (right)

Task 2

Overview

- Class invariants
  - Specify restrictions that must be satisfied by objects of a class at stable times
    - Class invariants are both rights and duties for users and implementers
- Nominal Programming
  - Specify conditions that must hold upon entry to a method in terms of preconditions
    - Effects of methods are still expressed in terms of postconditions
- Verification
  - Work out a collection of tests to verify the correctness of the class
    - Develop a consistent set of tests for each method
- Method Definition: Advanced Topics
  - Method overloading and methods with variable number of arguments
- Epilogue

Nominal Programming

- Impose additional conditions on the invocation of methods in dealing with exceptional cases
  - Conditions imposed on methods are referred to as preconditions
    - Argument types already express restrictions imposed on method invocations
  - Users of a class must see to it that all preconditions are satisfied upon entry to a method
    - Users of a class must not be confused with end-users
  - Implementers may assume that all stated preconditions are satisfied
    - Preconditions are duties for users, and rights for implementers

Preconditions

- In this course, preconditions are specified both formally and informally
  - The specification of a precondition is worked out in the heading of the method to which it applies
    - Each precondition starts with the non-standard tag “@pre”
    - The formal specification is separated from the informal specification by the symbol “|”
  - Preconditions for method invocations in effect-clauses and in return-clauses propagate to the specified method

Assert statements

- Since version 1.4, Java supports assert statements
  - In its simplest form, an assert statement involves a boolean expression
    - If the boolean expression is true, execution proceeds with the next statement.
    - If the evaluation yields false, the Java Virtual Machine throws AssertionError.
  - The more general form involves another expression whose value is incorporated in the thrown exception
- The Java Virtual Machine offers flags to control assertion checking
  - The flags -enableassertions (-ea), respectively -disableassertions (-da)

Task 3+4

Overview

- Class invariants
  - Specify restrictions that must be satisfied by objects of a class at stable times
    - Class invariants are both rights and duties for users and implementers
- Nominal Programming
  - Specify conditions that must hold upon entry to a method in terms of preconditions
    - Effects of methods are still expressed in terms of postconditions
- Verification
  - Work out a collection of tests to verify the correctness of the class
    - Develop a consistent set of tests for each method
- Method Definition: Advanced Topics
  - Method overloading and methods with variable number of arguments
- Epilogue

Black-Box Testing

- Develop a separate test for each case that can be distinguished in the specification of a method
  - The test program itself checks whether the results obtained from each test are correct
    - Human inspection of test results is time consuming
  - No tests can be worked out for basic inspectors nor for private methods
    - Basic inspectors have no specification
    - Private methods are inaccessible outside their class
  - No tests are worked out to check the behavior of methods under conditions that violate their preconditions

Test Structure

- Tests for different classes in a software system are typically worked out in separate test cases
  - In JUnit 4, test cases no longer inherit from any predefined class
    - Import statements reveal the ingredients used
- A test suite is set up to collect all the tests for the entire software system
  - In JUnit 4, a test suite is a class annotated with "@RunWith" and "@Suite"
- In Eclipse, classes related to tests can be assembled in a separate folder

Tests and Test Fixtures

- Test fixtures can be defined separately
  - In JUnit 4, methods to set up, respectively to tear down test configurations must be annotated "@Before", respectively "@After"
    - Use them to set up mutable test fixtures
  - JUnit 4 also offers annotations "@BeforeClass" and "@AfterClass" to work out initial, respectively final actions for all tests
    - Use them to set up immutable test fixtures
- A particular test checks the correct functioning of a method in a single case
  - In JUnit 4, test methods are annotated "@Test"
- JUnit offers methods for inspecting results obtained from testing methods
  - Examples of such methods are "assertEquals", "assertTrue", "assertNull", …

Task 5

Overview

- Class invariants
  - Specify restrictions that must be satisfied by objects of a class at stable times
    - Class invariants are both rights and duties for users and implementers
- Nominal Programming
  - Specify conditions that must hold upon entry to a method in terms of preconditions
    - Effects of methods are still expressed in terms of postconditions
- Verification
  - Work out a collection of tests to verify the correctness of the class
    - Develop a consistent set of tests for each method
- Method Definition: Advanced Topics
  - Method overloading and methods with variable number of arguments
- Epilogue

Method Overloading

- Classes may offer different methods with the same name
  - Overloaded methods must differ in the number and/or type of their arguments
    - This enables Java to associate each method invocation with the proper definition
  - Overloading is needed amongst others in each class that wants to offer several constructors

Variable Number of Arguments

- Since version 1.5, Java supports the definition of methods with a variable number of arguments
  - The type of the last formal argument in a method may be complemented with ellipsis ("…")
    - A series of values of that type may then be supplied in invocations
  - A variable argument actually stands for an array of values
    - In the body of the method, a variable argument is handled as an array
    - The method may also be invoked with an array of values of that type

Enhanced For-Statement

- Java 1.5 supports an enhanced for-loop based on iterators
  - The heading of an enhanced for-loop has the form "for (Type variable: collection)"
    - The body of an enhanced for-loop is executed once for each element returned by an iterator over the collection
      - In the body, the current element is accessible via the variable introduced in the heading
  - The enhanced for-loop is also available to iterate over the elements of an array

Quantifiers

- The universal quantifier expresses an assertion that must be satisfied by all elements in a collection
  - The universal quantifier is denoted “for each elem in set: p(elem)”
    - The predicate p(elem) must be satisfied by all elements in the given set
  - The universal quantifier may also be denoted as “for each index in begin..end: p(index)
- The existential quantifier expresses that an assertion must be satisfied by at least one element in a collection
  - The existential quantifier uses the keywords “for some”
    - In bounded versions of the existential quantifier, the keyword “some” is replaced by “one”, “two”, …

Set Comprehension

- Set comprehension offers facilities to construct sets
  - The construct “{ x in S | p(x) : e(x) }” denotes the set of all elements resulting from the evaluation of the expression e(x)
    - The expression is evaluated for all elements x in the set S that satisfy the predicate p(x)
    - “{ x in S | p(x)}” is a shorthand for “{ x in S | p(x) : x }”
- Basic sets correspond to classes, arrays, collections and ranges
  - Operators such as sum(S), product(S), max(S) and min(S) are applicable to sets of numeric type

Task 6

Overview

- Class invariants
  - Specify restrictions that must be satisfied by objects of a class at stable times
    - Class invariants are both rights and duties for users and implementers
- Nominal Programming
  - Specify conditions that must hold upon entry to a method in terms of preconditions
    - Effects of methods are still expressed in terms of postconditions
- Verification
  - Work out a collection of tests to verify the correctness of the class
    - Develop a consistent set of tests for each method
- Method Definition: Advanced Topics
  - Method overloading and methods with variable number of arguments
- Epilogue

Summary

- Class invariants
  - The heading of a class specifies restrictions imposed on properties ascribed to the class itself and to individual objects
    - Encapsulate class invariants for a property X in an instance method canHaveAsX or in a class method isValidX
  - In their standard state, objects must satisfy all their invariants
    - Objects involved in a method can be annotated "raw", meaning that they are not guaranteed to satisfy all their invariants
- Preconditions
  - Definitions of methods may be complemented with preconditions imposing additional restrictions on their invocation
    - Preconditions imply rights for the implementers of a class, and duties for the clients of a class
- Verification
  - In black-box testing, a different test is worked out for each case that can be distinguished in the specification of a method
    - The testing framework JUnit supports the development of test suites

Homework

- Add the following methods to the definition of the class of oil tanks
  - A method to initialize a new oil tank with given capacity and no contents
  - A method to initialize a new oil tank with capacity 5000 and no contents
  - A method to transfer the entire contents from a given oil tank to the oil tank to which the method is applied
  - A method returning a textual representation of the oil tank to which it is applied
  - A method checking whether two oil tanks have the same capacity and the same contents
  - A method returning a new oil tank as an exact copy of the oil tank to which it is applied