

# 6.170 Quiz 1 Review

Lecture 1 – Lecture 10

Quiz Date: March 6, 2002

2/28/02

1

## Quiz Info

- Wednesday, March 6, 2002, 10am
- Locations
  - 54-100 usernames a\*-j\*
  - 34-101 usernames k\* - z\*
- Open book and open notes, but it is only a 50 minute exam!

2/28/02

2

## Quiz Topics

- True/False
  - Java
  - General Concepts
- Dependences & Decoupling
  - MDDs
  - Specifications
- Testing and Debugging
  - Revealing subdomains
  - Black, Glass, Boundary
- Exceptions
- Object Models
- Representation Independence and Exposure
- Abstract Data Types
  - Representation Invariants
  - Abstraction Function
  - Inductive Reasoning

2/28/02

3

## Decoupling

- Decomposition
  - Division of Labor
  - Reuse
  - Modular Analysis
  - Localized Change
- Top Down Design vs. Modularization
- Uses Diagram: Trees, Layers, Cycles
  - Reasoning
  - Reuse
  - Construction Order
- Dependencies & Specifications; MDDs for
  - Weakened assumptions
  - Evaluating changes
  - Communication
  - Multiple implementations
  - (Dependence stops after one step)

2/28/02

4

## Decoupling

MDDs, Techniques

- MDDs
  - Specification parts
  - Implementation parts
  - Meets, depends, weak depends relationships
- Techniques
  - Façade: new implementation part between two sets of parts
  - Hiding representation: avoid mentioning how data is represented
  - Polymorphism: 'many shaped'
  - Callbacks: runtime reference to a procedure

2/28/02

5

## Decoupling

Java Namespace, Access Control, Interfaces

- Java Namespace
  - Packages → {Interfaces, Classes} → {methods, named fields}
- Access Control
  - public: accessed from anywhere
  - protected: accessed within package or by subclass outside of package
  - default: accessed within package
  - private: only within the class
- Interfaces: more flexible subtyping
  - Express pure specification
  - Allows several implementation parts to one specification part

2/28/02

6

## Specifications

- Behavioral equivalence
- Specification structure
  - Precondition: requires
  - Postcondition: effects
  - Frame condition: modifies
- Declarative specifications
  - Do not give implementation details
  - Properties of final outcome based on initial state

2/28/02

7

## Specifications

- Preconditions: engineering judgment based on cost of check of a precondition and scope of method
- Specification A is at least as strong as a specification B if
  - A's precondition is no stronger than B's
  - A's postcondition is no weaker than B's, for states that satisfy B's precondition (includes exceptions)
  - Essentially  $\rightarrow$  A is at least as useful as B if A requires less and guarantees/promises more than B

2/28/02

8

## Specifications

- Judging specifications
  - Informative
  - Strong enough
  - Weak enough
- Crucial wall between implementer and client

2/28/02

9

## Testing & Debugging (1)

- Terminology: validation, debugging, defensive programming
- Techniques and tips for successful debugging
  - Attitude
  - Scientific method
  - Binary search
- Testing and debugging are different
  - Testing: reveals existence of bugs
  - Debugging: pinpoints location of bugs

2/28/02

10

## Testing & Debugging (2)

- Unit testing
- Partitioning input space
  - Execution equivalence vs. revealing subdomain
- Revealing subdomain
  - Program has same behavior on all inputs
  - Gives incorrect result on all of those inputs
- Partitions should contain at least one revealing subdomain

2/28/02

11

## Testing & Debugging (2)

- Black box: partitioning based on specification
  - Process not influenced by component being tested
  - Robust with respect to implementation changes
  - Independent testers
- Glass box: partitioning based on code execution
  - Insight in to test cases
  - Find important class of boundaries
  - (statement coverage, decision coverage, loop coverage, condition/decision coverage, path-complete coverage)
- Boundary testing
- Regression testing

2/28/02

12

## Exceptions

- Defensive programming
- Runtime assertions
  - When: as you write the code
  - Where
    - Precondition (beginning of method)
    - Post condition (end of method)
    - Frame condition (external effect)
  - Slow execution

2/28/02

13

## Exceptions

- Responding to Failure
  - Fix: complicated, more bugs, if you know the cause → you could have avoided it anyway?
  - Execute special actions: depends on the system → hard to determine set of actions
  - Abort execution: depends on the program; compiler vs. word processor
- Exceptions
  - Non-local jumps
  - Special results
  - Checked vs. Unchecked: burden to the user; only difference is at compile time
  - General rule:
    - Checked exceptions for special results
    - Unchecked exceptions for failures
  - (Course text, p. 73)

2/28/02

14

## Object Models

- Object model: description of collection of configurations
  - Classification of objects
  - Relationships between objects
- Object Models vs. MDDs
  - Runtime vs. Static
  - “implements”/“extends” vs. “meets”

2/28/02

15

## Object Models

- Sets: subsets, disjoint subsets, exhaustive subsets
- Multiplicities (!, ?, +, \*):  $A \xrightarrow{m} \dots \xrightarrow{n} B$ 
  - $n$  B's associated with each A
  - $m$  A's mapped to each B
- Mutabilities
  - Sets
    - Static: objects in the set remain in the set for their existence
    - Fixed: contents of the set do not change
  - Fields
    - Target end: Once created, an object will have the same target(s)
    - Source end: The source of an object cannot change
- Instance diagrams
  - Representation of possible configurations
  - Graphs consisting of objects and references connecting them

2/28/02

16

## Intro to ADTs

- Data abstraction: type is characterized by the operations you can perform on it
- Built-in types vs. User defined types
- Classifying types
  - Mutable: can be changed; provide operations which when executed cause results of other operations on the same object to give different results (Vectors)
  - Immutable: cannot be changed (Strings)
- Operations ( $T$  = abstract type,  $t$  = some other type)
  - Constructors:  $t \rightarrow T$
  - Producers:  $T, t \rightarrow T$
  - Mutators:  $T, t \rightarrow void$
  - Observers:  $T, t \rightarrow t$
- List example

2/28/02

17

## Intro to ADTs

- Designing an Abstract Type
  - Few, simple operations that can be combined in powerful ways
  - Operations should have well-defined purpose, coherent behavior
  - Set of operations should be adequate
  - Type may be generic (list, set, graph) or domain specific (street map, employee db, phone book), but not both

2/28/02

18

## Intro to ADTs

- Representation Independence
  - Ensuring that use of an abstract type is independent of representation
  - Changes in representation should not affect using code
- Representation Exposure
  - Representation is passed to the client
  - Client is allowed direct access to representation
  - Need careful programming discipline
- Language Mechanisms
  - Access control: private, protected, default access
  - Interfaces (List interface and ArrayList & LinkedList implementations)

2/28/02

19

## Understanding ADTs (1)

- Data abstraction is defined by its specification
  - Collection of procedural abstractions (creators, mutators, observers)
- A rep = data structure + set of conventions
  - Rep invariant: defines set of reachable values of the data structure
    - Maps Object → Boolean
  - Abstraction function: how the data structure is interpreted
    - Maps Concrete → Abstract

2/28/02

20

## Understanding ADTs (1)

- Ordinary Induction
  - Assume property holds for arbitrary value
  - Prove it holds for the next value in the ordering
- Structural Induction (ADTs)
  - Can be used on any ADT
  - Producers and modifiers
    - Assume holds for x
    - Prove holds for x.o(...), where no assumptions made about arguments
- Reasoning about Rep Invariants
  - Show invariant holds on rep after each creator completes
  - Assume invariant holds when other operations are called

2/28/02

21

## Understanding ADTs (1)

- Example: CharSet
  - Base case: creator satisfies the invariant
  - Induction step: assume invariant holds before other operations are called
  - Prove for each operation
    - Observer (member): does not change rep, so invariant is preserved
    - Modifier (delete): invariant broken only when add elements, can never break the invariant
    - Modifier (insert): 2 cases (rep is unchanged, or rep is added only if not a duplicate)
- Benevolent side effects
  - Mutates the rep, but does not change abstract value

2/28/02

22

## Understanding ADTs (2)

- Structural Induction not always valid
  - Rep Exposure
    - Exploit immutability
    - Make a copy
- Checking the RI (checkRepInv)
- Abstraction function: relates concrete representation to the abstract value it represents
  - toCharSet(elts) = { c | c is contained in elts }
  - Abstraction function: toCharSet
  - Concrete objects: actual objects of the implementation (elts)
  - Abstract objects: mathematical objects that correspond to the way the specification of the abstract type describes its values ( { ... } )
  - Many (concrete) to one (abstract) mapping
- Combined AF & RI
  - Allows examining operators independently
  - "Correctness" a local issue by

2/28/02

23

## Understanding ADTs (2)

- Writing abstraction functions
  - Defined for all representations that satisfies the RI
  - Problem lies in denoting the range of the AF (notation for abstract values)
  - Overview section of specification is key: provides a way of writing abstract values
- Key points
  - RI
    - Which concrete values represent abstract values
    - Use induction to show that it is an invariant
  - AF
    - Which abstract value each concrete value represents
  - RI & AF combined: modularized implementation and reasoning
    - Look at: RI, AF, and specification
  - In practice,
    - RI almost always worth writing
    - AF harder to write and less useful

2/28/02

24