Object-oriented Software Design and Development
CCP114N

Week 5:

Class Design

- Concepts fundamental to the class design
  - Encapsulation & Information hiding
- Analyzing the quality of an interface
- Programming by contract
- Software & Unit testing

Fundamental Design Concepts 1/3

- Abstraction
  - Abstraction is an important simplifying technique enabling us to replace a complex and detailed real world situation with a model within which we can solve a problem.
  - In OO software design
    - We produce an abstraction of the software system to be built using a set of use cases, classes & their relationships, sequence diagrams and state diagrams, among others.
    - Built-in data types and programmer-defined classes are abstractions that describe data objects with states
    - Function/Method represents an action or a service to be provided
    - Control: a control mechanism without specifying internal details, e.g. a sequence diagram, or Remote Method Invocation (RMI)

Fundamental Design Concepts 2/3

- Encapsulation
  - As a process, encapsulation means the act of enclosing one or more items within a physical or logical container, as an entity
  - Encapsulation refers to the package or enclosure that holds one or more items.
  - In procedural programming
    - a function/module perform a task/functionality
    - examples?
  - In object-oriented programming
    - a class encapsulates both data and operations
    - examples?

Fundamental Design Concepts 3/3

- Information Hiding
  - Separation of interface and implementation
    - Software components can be developed in isolation from each other.
    - Also good for re-use.
  - The computer scientist David Parnas expressed these ideas in a pair of rules known as Parna's Principles
    - The developer of a software component must provide the intended user with the information needed to make effective use of the services provided by the component, and should provide no other information.
    - The developer of a software component must be provided with all the information necessary to carry out the given responsibilities assigned to the component, and should be provided with no other information.
  - **Notes**: While Encapsulation is design-related, Information hiding – implementation-related. However, some authors do not distinguish the meanings of “encapsulation” and “information hiding”. This could imply that information hiding seems to be an attribute of encapsulation.

Applying the concepts to class design

- A class can be an abstraction of a type of real world objects
  - shapes
  - cars
  - People
  - Account Payable service
- The class code encapsulates both
  - data
  - functions/methods/operation
- Good class design hides away its implementation details
  - private/protected/public
  - data members can only be accessed by Setter and Getter
  - It implies that the same class design may be implemented in more than one way, which doesn't affect the user/client of the class.

How to achieve Encapsulation and Information hiding by class design & implementation

- Quality of Class Interface
- Programming by Contract
- Use of Assessors and Mutators
- Avoiding side effects

Quality of Class Interface

- Customers/Clients are Programmers using the class
  - They are unaware of implementation
  - The class designer/programmer should make no assumption re. the customer knowing of implementation details
- Criteria:
  - Cohesion:
    - Class describes a *single* abstraction
    - Methods should be related to the single abstraction
  - Completeness
    - Support operations that are well-defined on abstraction
  - Convenience
  - Clarity
    - Design should be intuitive and following the user's mental model
  - Consistency
- Engineering activity: make tradeoffs

Programming by Contract

- Spell out responsibilities using pre- & post-conditions
  - of caller (preconditions)
  - of implementer ((postconditions)
  - That would increase reliability and efficiency
- **Preconditions**
  - Excessive error checking is costly
  - Returning dummy values can complicate testing
  - Contract metaphor
    - Service provider must *specify* preconditions
    - If precondition is fulfilled, service provider must work correctly
    - Otherwise, service provider can do *anything*
  - When precondition fails, service provider may
    - throw exception
    - return false answer
    - corrupt data
- **Postconditions**
  - Conditions that the service provider guarantees
  - Every method promises description, @return
  - Sometimes, can assert additional useful condition

Accessors and Mutators

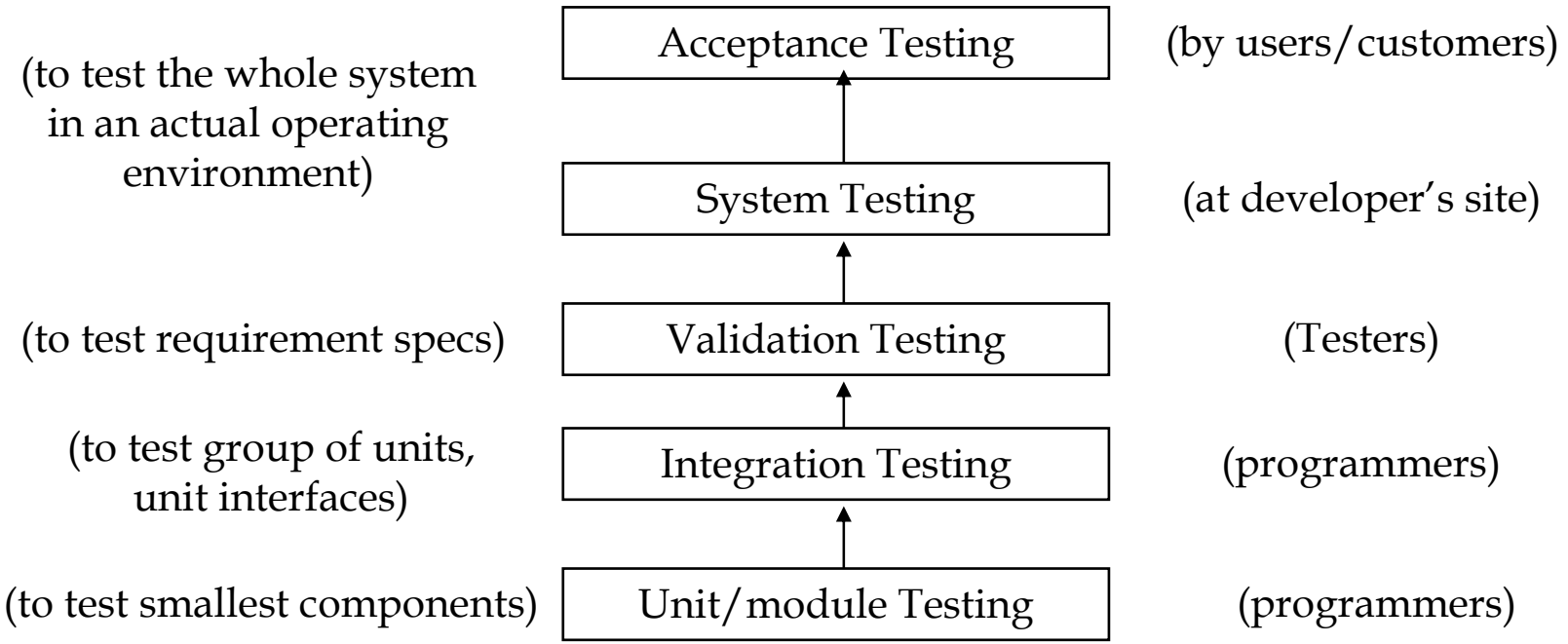
- Mutator: Changes object state
  - E.g. Setter methods of a class
- Accessor: Reads object state without changing it
  - E.g. Getter methods of a class
- Class without mutators is *immutable*
  - Java String class is immutable
  - Date and GregorianCalendar are mutable
- Sharing Mutable References
  - References to immutable objects can be freely shared
  - Don't share mutable references
    - To avoid: use **clone( )** method in Java or **final static** field in declaration

Side Effects

- Accessor: no change to object
- Mutator: changes object state
- Side effect: change to another object
  - Parameter variable
  - static object
- Avoid side effects--they confuse users
- **Law of Demeter**
  - The law: A method should only use objects that are
    - instance fields of its class
    - parameters
    - objects that it constructs with new
  - Shouldn't use an object that is returned from a method call
  - Remedy in mail system: Delegate mailbox methods to mail system
    ```
    mailSystem.getCurrentMessage(int mailboxNumber);
    mailSystem.addMessage(int mailboxNumber, Message msg);
    . . .
    ```
  - Just a rule of thumb, not a mathematical law

Software Testing Strategies and Techniques

- While the techniques of modular software design is *top-down*, the testing is *bottom-up*.



Software testing techniques 1/2

- Different testing techniques are used at the different testing stages. They can be categorised broadly into *white box* and *black box* testing.
- Back box testing
  - The software is treated like a black box
  - *Black box* testing is performed by creating test data according to the requirements specification, and the expected results, then testing the software using the prepared tests, and comparing the expected results with the actual results.
  - As well as establishing whether or not the software is correct, *black box testing* can result in changed functionality etc. to be more user friendly.

Software testing techniques 2/2

- White box testing
  - *White box* testing is the testing of the correctness of the software itself. The most important part of the testing process is to ensure that the tests have been properly designed so as to test all the paths, all the conditions rather than the same one every time.
  - The *basis path analysis*, which is based on McCabe cyclomatic complexity metric, is a well-established *white box* testing technique.
  - *Module Integration Testing* where a group of modules is tested e.g. a module and its immediate subordinates
  - *Unit testing* can be performed using ***driver programs***, or skeleton programs in which each unit would initially exist as a ***stub***, and then inserted to replace the stub to be tested within the skeleton program.

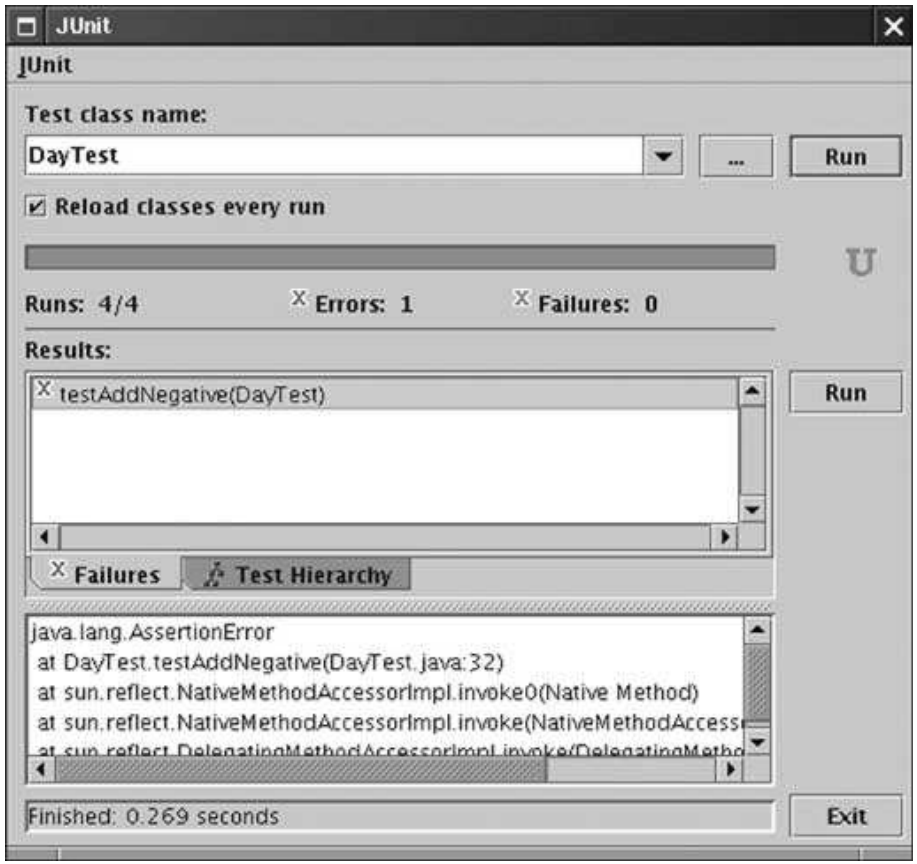
Unit testing with JUnit

- JUnit is an increasingly popular tool for unit testing
- A tutorial and more details can be found at http://junit.org
- Test class name = tested class name + Test
- Test methods start with test

```
import junit.framework.*;
public class DayTest extends TestCase
{
public void testAdd() { ... }
public void testDaysBetween() { ... }
. . .
}
```

- Each test case ends with assertion
- Test framework catches assertion failures

```
public void testAdd()
{
Day d1 = new Day(1970, 1, 1);
int n = 1000;
Day d2 = d1.addDays(n);
assert d2.daysFrom(d1) == n;
}
```



Summary

- Fundamental design concepts
  - abstraction
  - encapsulation
  - information hiding
- Class design techniques
  - Design “good” class interface
  - Programming by contract
  - Use of assessors & mutators
  - Avoid side effects
- Software & unit testing