

# CS249A Midterm Review

Anant Bhardwaj

[anantb@cs.stanford.edu](mailto:anantb@cs.stanford.edu)

# Details

- November 1, 2011 (7:30 – 8:15PM).
- 75 Minutes in length.
- Closed notes. Closed books.
- Sample midterms on web page.
- Through this weeks material.
- These slides will be posted after the review session.

# Chapter 1 Introduction: Software Husbandry and Software Development

- Software Development Cycle – “Husbandry”
- Maintaining Software Quality
- Framework Pyramid
- Methodology: SOS

# SOS

- Source Code Representation
  - Compare to specification languages
- Outside In Development
- Short Cycle Development

# Sample Question

- Cheriton makes a big deal of the fact that software development is iterative and claims the course techniques help with that. Describe three key techniques presented in the course that aid with iterative development and describe how they do so.

# Answer

- Source code representation
  - Less to learn
  - Efficient iterative development
  - Less error-prone
- Outside-in development
  - Test scenario
  - I/F design
  - (Multiple) Implementation
- Short cycle development
  - Avoid uncertainties of long cycle
  - Quick feedback

# Chapter 2: Attribute-Based Interface Design

- Attributes
  - Data type, name, and operations.
- Accessor Semantics
  - Const member function
  - Takes no parameters unless in a collection.
  - Nilpotent (Think reading)
- Mutator
  - Non-const (Think writing)
  - Side-effects allowed
  - Transactional (occurs fully or not at all)
  - Idempotent (no effect to writing the current value)

# Sample Question

- Cheriton's attribute-only approach, in going beyond the basic virtual variable model, calls for further constrained semantics for accessors and mutators, justified in part based on distributed implementation. List three such semantic restrictions and describe how they are justified (if not required) by a distributed implementation.

# Answer

- **Nilpotent [accessors]:**
  - Calling the accessor has no effect on the external state of the object.
  - It allows reads to be satisfied from cached object state, and allows a value to be read repeatedly without concern for some unintended side effect.

# Answer (contd...)

- **Idempotent [mutators]:**
  - Writing the current attribute value has no effect; i.e. calling the mutator with the same value multiple times has the same effect as calling it once.
  - A write operation over a network may be lost, causing the write to be reissued. Without idempotency, this situation could cause subtle bugs. Idempotency also simplifies the event notification model.

# Answer (contd...)

- **Transactional [mutators]:**
  - An accessor returns a consistent value at a given instance in time.
  - Otherwise, particularly in a distributed and concurrent environment, an external state that does not fulfill the object's invariant may be revealed, causing subtle bugs.

# Answer [contd...]

- **Exception-free [accessors]:**
  - An accessor never throws an exception but returns a default value instead.
  - Trying to propagate an exception in a distributed environment is challenging, and clients may not be able to recover from exceptions, recovering the state of objects, if accessors are allowed to throw an exception.

# Chapter 2 Continued

- Instantiating Attribute
- Collection Attributes
- Benefits of these interfaces
  - Clear semantics
  - Behavior encapsulation
  - Less documentation
  - Sorted directory example

# Chapter 3 Events Notifications and Callbacks

- Notification Model
  - Client defines a reactor type derived from notifiee
  - Override “on” methods.
- Reactors and multiple inheritance
- Activities
- Designing with Notifications
  - Call up/notify down versus call down/notify up
- Type Dispatch and the Visitor Pattern

# Question

- You join a new company and sing the praises of separating processing logic from state. Consequently you are handed the challenge to demonstrate how to do so with the following code.
- In the following, print is called to the cause sensors to print themselves and onReconfig is called when the system configuration is changed.

# Question (contd...)

```
class Sensor: public NamedInterface {
    virtual void print(Printer *);
    virtual void onReconfig(Config *);
    ...
};
class GPS: public Sensor {
    void print(Printer *);
    void onReconfig(Config *);
    ...
};
class Imu : public Sensor {
    void print(Printer *);
    void onReconfig(Config *);
    ...
};
```

# Solution

- Notifications for onRecording
- Double-dispatch/visitor pattern for print.

# Solution (contd...)

```
class Sensor {  
    class Functor : PtrInterface<Sensor::Functor> {  
    public:  
        virtual void operator()( Sensor* );  
        virtual void operator()( GPS * );  
        virtual void operator()( Imu * );  
    };  
    void operator()( Functor *f ) {  
        f->operator()(this);  
    }  
};
```

# Solution (contd...)

```
class PrintFunctor : public Sensor::Functor {
public:
    void operator()( Sensor *sensor) {
        // Print sensor
    }
    void operator()( GPS *gps) {
        // Print gps
    }
    void operator()( Imu *imu) {
        // Print Imu
    }
};
```

# Solution (contd...)

- Notifications for onReconfig() to separate event-processing from object model
- For instance, put onReconfig() in a SystemReactor which then changes state of sensors

# Chapter 4: Objects: Entities, Values and Descriptions

- Understand the distinctions between them.
- Entities
  - Objects of referent system
  - No copy constructor, no equality, no assignment
  - Referenced by (smart) pointers
  - No value operators (++ , -- , etc.)
- Micro-Objects

# Chapter 4

- Value Types
  - Equality, assignment
  - May be viewed as abstract collections
  - Pass by value
- Named Descriptions
  - Shared description of multiple objects
  - Similar to entity because of identity and referenced by (smart) pointer, but different
  - not really a real object
  - Similar to value types because of equality, has a set of values

# Question

- In the simulation of a swamp full of snails, one approach (the Entity approach) is to create an instance of an Entity type for each snail and each bacterium while another (the value approach) is to just represent the swamp as a matrix (or matrices) of values corresponding to a grid of the swamp area, each value a count of instances of the species in that portion of the swamp (grid).
- Describe how these two approaches can be related according to the relationship between entity types and value types described in the course.

# Answer

- In the entity approach, each entity represents a single snail or bacteria. In other words, an entity refers to an actual object in the referent system that one is simulating.
- The value approach is abstracting the collection of snails and bacteria into an abstract collection that only saves the cardinality of snails and bacteria in a grid block.

Questions?