

CS249A Final Exam Review

Anant Bhardwaj

anantb@cs.stanford.edu

Details

- December 12, 2011 (3:30 – 6:30PM).
- Alternate Finals (only after approval from the course staff) on Dec 13, 2011 (12:30-3:30PM)
- 180 Minutes in length.
- Closed notes. Closed books.
- Covers Chapter 1 – Chapter 10.
- Sample finals on the web page.
- 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

SOS

- 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)

Accessor Properties

- **Nilpotent:**
 - 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.

Accessor Properties

- **Exception-free:**
 - 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.

Mutator Properties

- **Idempotent:**
 - 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.

Mutator Properties

- **Transactional:**
 - 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.

Attribute Types

- 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

Functor Example

```
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 *);
    ...
};
```

Functor Example

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

Functor Example

```
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);
    }
};
```

Functor Example

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

State and Behavior Separation

- 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

Ref. Management, Smart Ptr

- PtrInterface, Ptr classes
 - Single Ptr class; use proxy objects
- Inter-object relationships
 - How to break circular references with notifications, config and status interfaces
- Optimizations
 - Anchoring with Ptr and pass raw pointer
 - Large-scale deletions
- C++ references - why they're dangerous

Automatic Garbage Collection

- Limitations
 - Loss of control
 - Runtime overhead
 - Reliability issues (bugs in collector)
- Usage scenarios
 - Garbage collector vs. alternatives

Sample Question

- Cheriton claims that Ptr allows you to engineer predictability into the performance and resource requirements of your software, compared to automatic garbage collection. Describe what this means, why this is hard to achieve with garbage collection, and a case to be careful of even with Ptr to achieve this.

Sample Answer

- Predictability in memory management
 - No random periods of excessive non-application-related processing
- Hard to achieve with auto. GC
 - GC kicks in at unpredictable times
 - GC processing intensive
 - GC may have bugs
- Potential issue with Ptr: Cascading deletion, i.e. a single Ptr going out of scope results in many deletions
- Deferred deletion, better structuring

Logging and Exceptions

- Logging for error reporting
- Alternative return model
- Pass-through to top-level caller
 - Reduce clutter, increase performance
 - Requirements for accessors, mutators, destructors, copy constructors
 - Transactional semantics
 - Catching at intermediate caller
- Exception types: Range, Resource, Return
- Mixed exception / return val. check

Sample Question

- Cheriton's Nemesis claims that exceptions should contain detailed information of the situation so that upper-level callers have a better chance of handling them. For instance, a disk controller should throw exceptions containing which block allocation failed. Is this a good idea? If so, why? If not, strike down this awful enemy with two reasons.

Sample Answer

- This is a bad idea of course!
- Too much info exposes implementation of module throwing the exception
- Caller can rarely understand implementation-specific information to rectify problem with the exception-throwing module
- Even if caller does know, it may need human intervention, e.g. switching disk, so logging is preferred for detailed error / exception info
- Generic error suffices, e.g. `DiskException`, caller can simply use another disk - simplicity

Naming, Introspection, Modules

- System-wide naming facility with user-meaningful names
- Introspection with `Fwk::Type`
- Directory structure, hierarchical names
 - `Fwk::Dir`
- Optimizations
- State-oriented approach
 - Mounting, system database, agents
- Generic access and scripting (untyped)

Sample Question

- After skimming through your precious CS249A course reader, Michael challenges Cheriton's approach of giving system-wide names to objects, claiming that pointers are all you need in referencing objects anyway. How would you convince him of the importance of a system-wide naming system?

Sample Answer

- Need for user-meaningful names
- GUI / CLI, e.g. a room object named “/Stanford/Gates/B1”
- Error reporting and logging, e.g. “/Stanford/CS249A/TA/Anant is giving an awesome review session”
- Relaxing referential integrity, i.e. dangling pointers e.g. “/Stanford/CS249A/Student/Abhinav” and “/Stanford/CS249A/Student/Joy” are good friends.

Sample Answer

- Need for uniform system-wide naming
 - Smoother learning curve
 - Facilitate logging, error reporting, e.g. easier parsing
- Reduce name conflicts
- Necessary for large (distributed) systems, e.g. mounting remote directory of objects

Components, Dynamic Discovery

- Component-oriented design
 - Behavioral coupling
 - Composite attributes - evolvability
 - Multi-level simulations
- Vs. Base class-oriented design
 - More flexible
- Collisions - generalization of components
 - Dynamic relationships
- Why need both?

Sample Question

- There is a major performance concern about the collision-based model. Explain what kinds of optimizations you would implement to make the model practical, preferably with short examples.

Sample Answer

- Break space into subspaces and perform collisions per subspace, e.g. quadtrees
- Components, quadratically reduce number of collision detections

Type Hierarchy, Inheritance

- Restricted inheritance
 - Basic-Extended Interface
 - Interface-Implementation
 - Basic-Extended Implementation
- Inheritance vs. composition
 - Static, singleton, visibility
- Replacing inheritance with components
- Inheritance as optimization
- Multiple inheritance
 - Problems; how we use it in class

Sample Question

- You are asked to review the following code snippet. Point out the problems with it based on what we know about inheritance.

```
class Person {  
    ...  
    private:  
    string name_;  
};  
class CEO : public Person {  
    ...  
};
```

Sample Answer

- Static relationship - Difficult to change the CEO to another person
- Singleton - CEO has to be one person.
 - Problem if we want to have a co-CEO
- Visibility - Suppose we change CEO to be filled by more than one person, then client code needs to be changed as well, because it assumed CEO is a single person

The End

- Hope, you had a wonderful quarter!
- Please send your feedback/suggestions to cs249a-staff@cs.stanford.edu
- Thanks and good luck for the final exams!