

# CS249 Final Exam: Closed Book, 180 Minutes

Instructor: David Cheriton, Monday, March 18, 2002

December 3, 2007

This is a CLOSED-BOOK exam. You are expected to work on your own to complete this exam, not using other people or any materials for your assistance.

- Please answer all questions; read each question carefully. You waste time by writing more than required by the question, and you lose points by answering the wrong question.
- Please give precise, specific answers that demonstrate the depth of your knowledge of the area being examined by the question, rather than vague and general comments. Point form answers are acceptable. You can use sketch code that is not necessarily complete AS LONG AS IT CLEARLY INDICATES you understand the approach or technique we are after.
- Please put your name and student number on each exam booklet you use and sign the honor code statement.

1. **(12 Points)** Describe how separating interface from implementation benefits each aspect of the SOS methodology, i.e. for each of:
  - (a) Source code representation
  - (b) Outside-in development
  - (c) Short-cycle Development
2. **18 Points** Cheriton argued that entities should be programmed differently than values.
  - (a) Give an example of a *common description* object and describe why it is different from an entity object, from the modeling and simulation perspective.
  - (b) Describe how a common description object is similar in semantics and programming to a value object class.
  - (c) Describe the key differences in how entity classes are programmed and used compared to value classes.
3. **(18 Points)**
  - (a) Describe the key difference between when it is legal to use the C++ standard library `auto_ptr` vs. our `Ptr` class template.
  - (b) Describe one good reason to override `PtrInterface::onZeroReferences`.
  - (c) Describe why having a single notification type, namely on low memory, is adequate for the `MemAlloc` interface, given that the allocator can run out of memory, have parity errors and perhaps other problems.
4. **(18 Points)** Some frameworks and languages derive all objects from a common root type, such as `Object`. Cheriton objects to that, yet ends up with many classes derived from a common root type `NamedInterface`.
  - (a) Why have some frameworks derived all objects from a single root `Object` class, what are the problems with this, and how does C++ allow you to avoid this.

- (b) Smart-boy Fred claims that it is unnecessary to have some many interface classes derived from `NamedInterface`. What are the implications of having public interface classes not derived from a single root class like `NamedInterface`, i.e. give the argument for Fred to keep Cheriton's structure.
5. **24 Points** Cheriton tries to introduce significant structure into exceptions and exception handling. For each of the following, say why it's a good thing, illustrating with an example (or else argue it is unnecessary, if you dare!) (You can use short code snippets and can share code across each of the following, if you choose):
- All exceptions except for return exceptions derive from a common `exception` type/class.
  - Use the exception intermediate types of `ParameterException` and `ProcessingException`, with exception types derived from each.
  - Use one try block per procedure.
6. **(18 Points)** Cheriton calls for a very restricted structure for callbacks, namely a single `Notifieee::onEvent` function that takes a single parameter, with a single such type of notification per interface. Describe the rationale for each of the following these aspects.
- Distinguishing a callback as a notification from other calls, given that they are all really just function calls.
  - Providing a void return function, `onEvent`, with exactly one parameter, and only this one function.
  - Having at most one type of notifiee per interface.
7. **18 Points**
- Smart-boy Fred wants to change the `Activity` to eliminate the use of `Notifieee` and just have client software derive from `Activity`, overriding a `Activity::callBack` virtual function, to provide the same basic functionality as calling `notifieee->onEvent`. Describe the disadvantages of Fred's proposal over the current `Activity` design.
  - Fred also claims that *component-oriented design*, as discussed in this course, is unnecessary. You can design classes without consideration of them as components and then provide an *adaptor* class that connects between a random object as a component and a random other object as the super object. Describe why Fred's approach is inferior.
  - Describe the trade-offs between having a relationship between two objects explicitly represented by a (smart) pointer versus using the collision detection mechanism to discover each other.
8. **18 Points**
- C++ provides the ability to define smart pointer classes and our framework takes advantage of that with the `Ptr` class. Yet, Cheriton, that spoil-sport, argues that you should use only this one smart pointer class. Explain why that should be the case.
  - Smart-boy Fred believes you should only use the `Ptr` class template for smart pointers and thinks it is unbelievably dumb that Stroustrup didn't just incorporate the `Ptr` class as a built-in in the language. Why is it still reasonable to have it as a class defined by the framework or user?
9. **18 Points** You are provided with an `OpenFile` interface class

```
class OpenFile : public NamedInterface {
public:
    virtual Block data( int blockNo ) = 0; // read a block
    virtual void prefetchBlock( int blockNo ) = 0; //asynch read block
    virtual void data( Block b ) = 0; // write a block
};
```

Your job is to design the additional interface classes for an extensible manager class that provides multiple dynamically loadable implementations of this interface. The manager should be able to control the block caching per `OpenFile`. Sketch in pseudo-code these interface class definitions with the key attributes of each, with a comment explaining the role of each.

**10. 18 Points**

- (a) Sketch a code example illustrating a plausible coding error that static type checking would catch and relies on hierarchical typing for the intended correct solution. (Make explicit any type relationships that are relevant to the example.)
- (b) Sketch a code example in which run-time type identification is used to select among a number of possibilities yet cannot be sensibly replaced by a virtual function call, unlike what our friend Bjarne Stroustrup once thought.
- (c) Describe why it is attractive to structure the direct manipulation level as manipulating a single type.
- (d) Smart-boy Fred argues that Cheriton's *glue* layer between the GUI and the "gore" layer is unnecessary. I.e. you can just connect directly from the GUI to the statically typed layer. Describe the problems Fred will run into in trying to connect them directly.

The End, Have a good spring break!