CSE403 Winter 2009 Midterm
February 18, 2009

- 100 total points, 4 questions @ 25 points each.
- Open-book and open-note, but closed-any-other-sentient-being; all material not produced directly by you must be appropriately cited.
- You may work on the exams anywhere you wish.
- Your exam is due through the Catalyst drop-box two hours after your chosen starting time.
- I expect you to only need to work for one hour in the two hour period.
- I will be online during the examination period as much as possible, to answer questions.
- Submit it in PDF or Word or text or some form you are certain I can read.
- Answers that are short-and-sweet are much preferred to long-and-rambling responses for which you hope to get partial credit because you said something that is true even if it doesn't relate to the question.

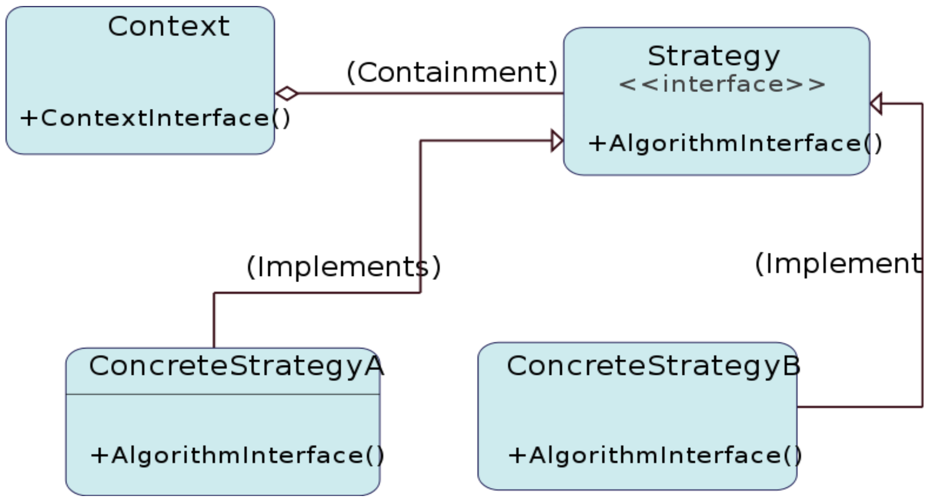
1. Consider the Manifesto for Agile Software Development. Using elements of the manifesto explicitly, and considering classes of software to include elements such as desktop applications, data processing applications, operating systems, and many others, either
   a. describe a class of software products for which Agile methods is clearly inappropriate, or
   b. state clearly why Agile methods are appropriate for developing all classes of software.

2. Consider your 403 budget game project, and consider Michael Jackson's notions of outer and inner requirements (in "Seeing More of the World"). Using examples from your project to illustrate outer from inner requirements; then describe how this distinction could affect or have affected your SRS, your SDS, and your zero feature release, if at all.

3. Consider Parnas' information hiding design approach ("On the Criteria To Be Used in Decomposing Systems into Modules") as well as Kiczales' aspect-oriented design approach as described in the video of his talk at Google. Both approaches allow some design decisions to be changed more easily later on: for example, in Parnas' KWIC example, the information hiding decomposition allows the circular shift module either to create the shifts at one time or else to create them incrementally on demand; and for aspect-oriented programming, a classic example is to allow loggers to be defined as aspects, easing the creation, deletion or modification of a given logging policy for a program.
   a. (5 points) Describe a design decision that could be changed more easily later on using an object-oriented design approach.
   b. (20 points) Assume you are the manager of software development at a major software firm. Given that each of these three design approaches (information hiding, aspect-oriented, and object-oriented) simplifies later changes, how would one reasonably and consciously decide upon one of them over the others for the company? Would it be on a company-wide basis, a per-project basis, or otherwise (and why)?

4. Consider the strategy pattern as described in Wikipedia (http://en.wikipedia.org/wiki/Strategy_pattern): "The strategy pattern is useful for situations where it is necessary to dynamically swap the algorithms used in an application. The strategy pattern is intended to provide a means to define a family of algorithms, encapsulate each one as an object, and make them interchangeable. The strategy pattern lets the algorithms vary independently from clients that use them."

   The provided UML diagram is:



   Two different design relationships are shown explicitly in this diagram: "Implements" and "Containment". However, other design relationships are implicit in the definition and use of this strategy for a specific set of clients. Discuss these other relationships, in particular those that associate the clients with the strategy.