Activity 8-2: Engineering Design Principles

Why?

Designers must have criteria to guide design generation and serve as the basis for choosing between design alternatives. Several constructive design principles are widely accepted in the development community as the essential basis for software engineering design.

Learning Objectives

- Understand the Principles of Small Modules, Information Hiding, Least Privilege, Coupling, Cohesion, Simplicity, Design With and For Reuse, and Beauty
- Use constructive principles to make engineering design decisions

Success Criteria

- Be able to explain the following concepts: module, modular, information hiding, coupling, and cohesion
- Be able to state the nine constructive design principles
- Be able to justify design decisions in terms of the nine constructive design principles

Resources

ISED sections 8.2 through 8.4

Vocabulary

Design principle, basic design principle, constructive design principle, modular, module, information hiding, coupling, cohesion, reuse, beauty (simplicity and power)

Plan

1. Review *ISED* sections 8.2 through 8.4 individually.
2. Answer the Key Questions individually, and then evaluate the answers as a team.
3. Do the Exercises as a team, and check your answers with the instructor.
4. Do the Problems (based on the Case Study) and Assessment as a team.
5. Turn in the Problems and Assessment as a team deliverable.

Key Questions

1. What is a module? What are some examples of modules?
2. How big can a small module be?
3. What is information hiding?

Exercises

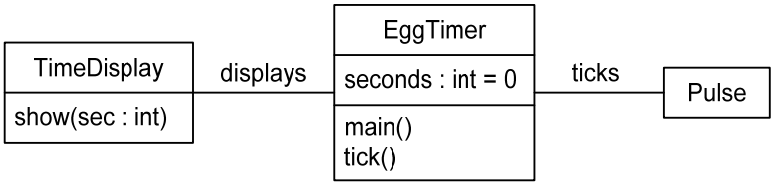
1. What are cohesion and coupling and how are they related?
2. What do power and simplicity have to do with beauty? Is that all there is to beauty?

Case Study

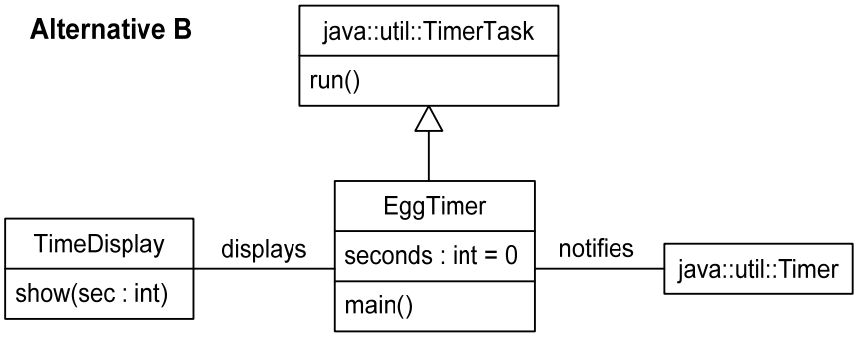
We consider three alternative designs for a simple program called *EggTimer*. This program counts down seconds given to it as a parameter and dings or beeps when time is up. More specifically, the EggTimer program must take a single integer value on the command line. It must interpret this value as the number of seconds to count down until dinging or beeping the computer. The EggTimer program must display the number of seconds left as it counts down. The first version of the program must display its output on the command line. Later versions must open a GUI window and display the countdown there.

The following figure shows three alternative designs for EggTimer. The diagrams are UML class diagrams. The connector with a triangle at one end represents the generalization or inheritance relation. The class at the triangle end of the connector is a super-class of the class at the other end of the connector.

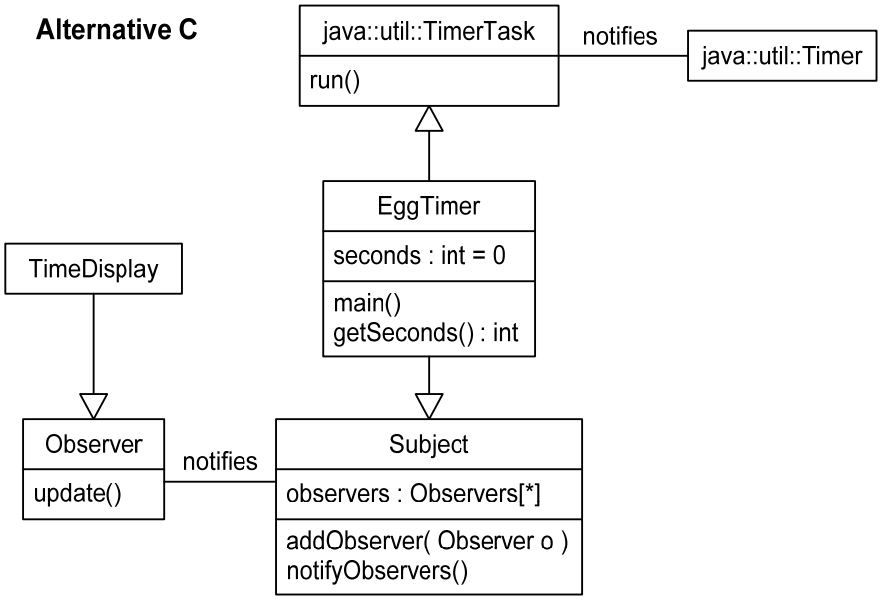
Alternative A



Alternative B



Alternative C



Alternative A uses a Pulse class to notify (or tick) the EggTimer class every second. The EggTimer calls TimeDisplay.show() to display the number of seconds left. The TimeDisplay dings or beeps the computer when seconds reach 0.

Alternative B replaces the Pulse class with the java.util.Timer class. Instances of this class call the run() operation in a java.util.TimerTask object at regular time intervals (like every second). In this design, the run() operation in EggTimer calls TimeDisplay.show() to display the time.

Alternative C retains the java.util classes from alternative B but changes the way that the TimeDisplay class works. In this alternative The EggTimer is a subclass of Subject. A Subject class keeps a list of observing objects. Every time its notifyObservers() operation is called it calls every observer's update() operation. When EggTimer begins, it creates a TimeDisplay object and adds it as an observer of EggTimer. When EggTimer.run() executes, it calls notifyObservers(), which in turn calls TimeDisplay.update(). This operation calls EggTimer.getSecond() to get the time left and display it.

Problems (Deliverable)

1. Evaluate each of the EggTimer design alternatives with respect to each of the nine constructive design principles. Rate each alternative as good, bad, or ok for each principle.
2. Which design principles seem to distinguish these designs from one another?
3. What are the relative strengths and weaknesses of these alternatives in comparison with one another?
4. On balance, which alternative do you think is the best? Justify your choice.
5. Can you think of an even better design alternative? If so, document your design with a UML diagram and a brief explanation like those above.

Assessment (Deliverable)

1. Did your team improve its performance from last week?
2. Did this activity help you achieve the learning objectives?