Engr 660: Software Engineering II
(Software Design)
Fall Semester 2005, Assignment #1
Revised Due Date: Monday, 12 September

The purpose of this assignment is to create and document a modular design using the information hiding and abstract interface principles. You are not (in this assignment) required to implement the program, but you are to create a modular design appropriate for implementation using the programming language Java. You will be working in two-person groups defined by your instructor.

You are to design a program that solves the problem described below. However, approach the design by considering this as one of a family of programs, giving attention to aspects that may change over time or from one user to another.

For this assignment you are required to design a simplified price calculation program that might be used in checkout system of a retail store (e.g., Wal-mart or Kroger). The program should do the following:

1. Read a set of items and their base prices from a file and store the data where it can be accessed by the program. The set should come from a text file with the data about each item appearing on a separate line. There are three pieces of data on the line in the following order, separated by spaces (blanks or tabs):
   (a) An item identifier, which is a nonnegative integer less than 100,000.
   (b) A price in whole cents.
   (c) A text description of the item.

2. Read a sequence of one or more "shopping carts" from a file. Each shopping cart has the following format:
   (a) The first line gives the customer identifier. This identifier is left-justified on the line and ends at the first space or the end of the line.
   (b) There is a line describing each item to be purchased. The line holds two pieces of data in the following order, separated by spaces: the item identifier and the quantity of the item being purchased.
   (c) The last line has -1 in the same position as the item number on previous lines.

3. For each item in a shopping cart, the program looks up the base price in the price data, uses a discounting policy to determine the amount of any applicable discount, and then calculates the final price of that item.

   The discounting policy supported initially is the following: if the customer is a member of the store's loyalty club (i.e., presents a loyalty card), then he or she gets a 10 percent discount on the second and subsequent items of the same kind. (For example, if a member buys one package of rice, then the member does not get a discount. But if he buys three packages, he gets a 10 percent discount on the second and third packages. Non-members do not get a discount.)

4. For each shopping cart, the program computes the total price of the items purchased and outputs the information on the items and the total to a receipt file.

5. Once all shopping carts are processed, the program will output statistics on the set of purchases.

6. The program may read a configuration file, if appropriate, in the parameterization of the program. (For example, information on tax rates might be there or the name of the file containing the loyalty club members.)

The program should be robust with respect to change. That is, a likely change should affect only one module. Give special attention to the discounting policy, which might clearly change over time and among different users.

When your design is complete, turn in *two paper copies* of the document that describes your design. The design document should:

1. Describe the overall modular structure you choose. (It may be helpful to include diagrams that show the relationships of the modules. You may use UML diagrams, but you are not required to do so.)

2. For each module:
   (a) Give its secret.
   (b) List the assumptions you make about all implementations of the module.
   (c) List the signatures of the methods (procedures and functions) that are in the abstract interface to the module.
   A signature of a method is the information in the header statement of the method – name, return value type, and the parameters, their types, and order.
   (d) Give a precise description of what each method does. (You may want to use preconditions and postconditions, but you are not required to do so.)
   Remember that the information about an abstract interface should not reveal the secret of the module.

   Note: Since Java is the target language, a module will likely consist of one or more classes and interfaces. Describe the public methods of the classes or interfaces but do not give implementations of the classes. If there is more than one class or interface that is exposed by the module, you may want to break up the specifications into a specification for each interface or class.

In future assignments, you may be called upon to critique the design of another. You may also be required to implement your design or, perhaps, the design of another. So strive for clarity, precision, and completeness. But also try to keep the design flexible and the document small.