ASSIGNMENT 4 – Implementing and Using Iterators

100 pts. (updated 2-29)

Due Monday, March 7th (by the start of class)

PROBLEM

You are to design and implement a Menu class, and associated iterator classes, that maintains a collection of MenuItem objects. A MenuItem object contains the following information about each menu item of a particular restaurant.

- Item name (e.g., "prime rib", "key lime pie")
- Category (appetizer, main dish, dessert)
- Heart healthy (yes or no)
- Price

The Menu class must provide getter (factory) methods for producing the following types of iterators:

- AllItemsIterator
  Iterates over all of the items on the menu

- ItemIterator
  Iterates over a specified item type
  (appetizer, main dish, or dessert)

- HeartHealthyIterator
  Iterates over the heart healthy items on the menu

- PriceIterator
  Iterates over the main dishes that are under a specified price

In addition, the Menu class should provide the following static constants:

```
public static final int APPETIZERS = 1;
public static final int MAIN_DISH = 2;
public static final int DESSERT = 3;
```

These are to be passed as an argument when requesting an ItemIterator (see below). You might also include,

```
public static final boolean HEART_HEALTHY = true;
public static final boolean NOT_HEART_HEALTHY = false;
```

passed when adding new items to the menu (through use of the add method).

APPROACH

You may implement the Menu class any way that you wish (array, ArrayList, LinkedList, etc.). The class should also provide methods for adding and deleting items on the menu. The add method is to be passed a MenuItem object to *append* to the end of the list of menu items. The delete method is to be passed an iterator pointing to the MenuItem to be deleted. (We do not care about adding menu items other than at the end.)

A natural way to handle the deletion of menu items is to use request an AllItemsIterator from the menu and display each menu item one-by-one as the use hits return. When the menu item is displayed that is to be deleted, a response of 'd' (instead of hitting just the return key) can indicate to delete the current item displayed. The iterator object can then be passed to the delete method of the menu object, since it would be pointing to the menu item to delete.

Your client code should be written to first populate a menu with a number of menu items,

```
Menu eatAtJoesMenu = new Menu();

eatAtJoesMenu.add("Lobster Dinner", Menu.MAIN_DISH, Menu.NOT_HEART_HEALTHY, "24.99");
eatAtJoesMenu.add("Rice Pudding", Menu.DESSERT, Menu.NOT_HEART_HEALTHY, "3.50");
etc.
```

The Menu class must provide "getter" methods (factory methods) for each of the types of iterators. For example, the getter for obtaining a menu iterator that iterates over all of the menu items would have the following function signature,

```
public MenuIterator getAllItemsIterator()
```

And the function signature for obtaining an iterator that iterates over a specific item type would be,

```
public MenuIterator getMenuItemIterator(int item_type)
```

which would be requested from the client code as follows (for Menu object menu),

```
MenuIterator itr = eatAtJoesMenu.getMenuItemIterator(Menu.APPETIZERS)
```

So for example, when the client code of the Menu class wants to iterate over all of the menu items, the iterator would be retrieved and used as follows,

```
MenuItem item;
MenuIterator itr = eatAtJoesMenu.getAllItemsIterator();

System.out.println("ALL MENU ITEMS");
while (itr.hasNext())
{
  item = itr.next();
  System.out.println(item.getName() + "  $" + item.getPrice());
}
```

When the client code wants to iterate over just the main dishes, the iterator would be retrieved and used as follows,

```
MenuIterator itr = eatAtJoesMenu. getItemIterator(Menu.MAIN_DISH);
System.out.println("MAIN DISHES");

while (itr.hasNext())
{
        item = itr.next();
        System.out.println(item.getName() + "  $" + item.getPrice());
}
```

When client code wants to iterate only over heart healthy items,

```
MenuIterator itr = eatAtJoesMenu. getHeartHealthyIterator();
System.out.println("ALL HEART HEALTHY MENU ITEMS");

while (itr.hasNext())

{
    item = itr.next();
    System.out.println(item.getName() + "  $" + item.getPrice());
}
```

When the client code wants to iterate over only dessert items,

```
MenuIterator itr = eatAtJoesMenu. getItermIterator(Menu.DESSERTS);
System.out.println("ALL DESSERT MENU ITEMS");

while (itr.hasNext())

{
    item = itr.next();
    System.out.println(item.getName() + "  $" + item.getPrice());
}
```

When the client code wants to iterate over only items under a certain price,

```
MenuIterator itr = eatAtJoesMenu. getPriceterator("15.00");
System.out.println("ALL ITEMS UNDER $15.00");

while (itr.hasNext())

{
    item = itr.next();
    System.out.println(item.getName() + "  $" + item.getPrice());
}
```

Note that each of the while loops above are identical. The particular iterator that each is using is what is different.

Each of the Menu iterators should implement the following interface, which should also be defined,

```
public interface MenuIterator
{
        // returns true if items of appropriate type left in list
        public boolean hasNext();

        // returns current item and advances to next item
        public void next();
}
```

USE OF PRIVATE (INNER) CLASSES

The client code should not be given the ability to create MenuIterator objects on its own (but instead obtain such objects by use of the provided factory methods of the Menu class). One way to control access is by making the iterator classes private classes of the Menu class. (To do this, just declare each iterator class within the Menu class, with modifier private.) Such inner classes have access to the private data of the surrouding class (the Menu class). The surrounding class also has access to the private members of each of the inner classes.

PROGRAM TO CREATE

Create a program that will perform each of the following. Note that you may hard code menu items in the program so that you do not need to add menu items to an empty menu each time you execute the program.

1 – Display all menu items
2 – Display all appetizers
3 – Display all main dishes
4 – Display all desserts
5 – Display all hearty healthy items
6 – Display all main dishes under a specified price
7 – Add menu item
8 – Remove menu item

WHAT TO TURN IN: Each of the source files, submitted as one zipped file.

DUE: Monday, March 7th by class time. **Late assignments will be marked off 10% for each 24-hours past the time due.**