CSci 4651 Spring 2006

Problem Set 8: Data Abstraction; Object-Oriented Programming.

Due Wednesday, April 19

You may (and are encouraged to) work in pairs on this assignment

Problem 1 (and only). For this problem you need to write two different implementations of multisets (sets that can have repeated elements). You also need to write two interfaces for multisets, one for the clients of these classes, and the other, more detailed one, for interaction of the two multiset implementations with each other.

Background on multisets. A multiset (also known as a bag) is a set that may contain the same element more than once. For instance, the multiset $M_1 = \{a, b, b\}$ has one copy of a and two copies of b. The order in which elements appear in a multiset does not matter.

The number of times an element appears in a multiset is called its *multiplicity*, denoted by $m()$. For instance, in the multiset M_1 above $m(a) = 1$, $m(b) = 2$. The set of all elements that appear in a multiset is called the *underlying set*. For instance, the underlying set of M_1 is $\{a, b\}$ (this is just a regular set, so each element appears once there).

The class structure. You need to write two implementations of multisets: `MultisetElements` and `MultisetMultiplicity`. The classes must implement an interface `MultisetInterface` which extends an interface `Multiset`.

As a minimum requirement for this problem, you need to implement multisets of elements of type `char`. **Extra credit:** implement generic multisets, i.e. those that can work with elements of any class. You may, if it is convenient for your implementation, restrict the class to Comparable objects.

The methods below are specified for `char`. If you are implementing generic multisets for extra credit, change `char` to the type parameter `T`.

- `MultisetElements` that uses an array or a list or a vector to store the elements. A duplicated element appears several times on this list.
- `MultisetMultiplicity` stores each element only once and stores multiplicities for each element occurring in the multiset. You may define an inner class to store an element and its multiplicity or you may store multiplicities separately from elements (in a separate array or a list), whatever is more convenient.
- `Multiset` - a Java interface that the client programs will use to refer to multisets. The interface mandates the following methods:
  1. `void add(char c)` - adds c to the multiset.
  2. `int multiplicity(char c)` - returns the multiplicity of c (i.e. the number of times c occurs in the multiset). Returns 0 if c is not in the multiset.

3. `Multiset union(Multiset ms)` - returns the union of this multiset and `ms`. The union of two multisets M_1 and M_2 is defined as a multiset M_3 with the underlying set $A_1 \cup A_2$ (where A_1 and A_2 are the underlying sets of M_1 and M_2, respectively), and $m_3(a) = max(m_1(a), m_2(a))$. For instance, $\{a, b, b, c\} \cup \{a, c\} = \{a, b, b, c\}$ (note that there is only one a in the result).

   The type of the returned multiset is the same as the type of `this` multiset.

4. `Multiset intersect(Multiset ms)` returns the intersection of this multiset and `ms`. The union of two multisets M_1 and M_2 is defined as a multiset M_3 with the underlying set $A_1 \cap A_2$ (where A_1 and A_2 are the underlying sets of M_1 and M_2, respectively), and $m_3(a) = min(m_1(a), m_2(a))$. For instance, $\{a, b, b\} \cap \{a, b, c\} = \{a, b\}$

   The type of the returned multiset is the same as the type of `this` multiset.

5. `boolean isSubset(Multiset ms)` - returns true if this multiset is a subset of `ms`, false otherwise.

- `MultisetInterface` - extends the `Multiset` interface by including a method

  ```
  char [] getUnderlyingSet()
  ```

  that returns the underlying set of the multiset as an array of `char`. Alternatively, you can use a pre-defined Set collection as the return type.

  If you are implementing the extra-credit generic version, and you are returning an array, you should return an array of type Comparable if T extends Comparable or an array of Objects if there is no bound on T.

  This method is used in the methods `union`, `intersect`, and `isSubset`. Note that you need to typecast a `Multiset` to `MultisetInterface` in order to get its underlying set.

In addition the two implementation classes must provide a constructor to create an empty multiset.

Use and testing. Write a main method or a JUnit test class to test that the methods `union`, `intersect`, and `isSubset` work correctly in all 4 possible cases:

1. When called on `MultisetElements` with a `MultisetElements` parameter,
2. When called on `MultisetElements` with a `MultisetMultiplicity` parameter,
3. When called on `MultisetMultiplicity` with a `MultisetElements` parameter, and
4. When called on `MultisetMultiplicity` with a `MultisetMultiplicity` parameter.

Note that the constructors cannot be called through the interface, so they have to be called in the testing code explicitly, thus breaking encapsulation. There are ways to hide constructors from a client, but they are not perfect and not required here.