

PROGRAMMING ASSIGNMENT 2

Ordered Linked List

In this programming exercise you are going to design, implement and test an *ordered linked list* based on the generic List class introduced in the notes. The purpose of this exercise is to understand in detail the use of inheritance through generic methods that are provided in superclasses and can be used in diverse subclasses, resulting in reusability and extensibility. In this specific case, generic *find*, *insert* and *remove* methods will be added to class `OrderedList` (demonstrating reusability) and then could be used by specific subclasses, e.g. `IntegerList`, `StringList`, etc., by introducing type-specific *compare* methods (demonstrating extensibility).

The class `List` introduced in the notes supports linked list functionality in terms of inserting and retrieving elements to/from the front and back of the list only. But in some applications, e.g. time-driven simulation of a real system, elements need to be inserted in the list keeping it ordered, e.g. representing the time when an event will happen. In such a use of an *ordered* linked list, elements are only retrieved from the front of the list, e.g. as time advances and the first item in the list becomes the “current event”, and are inserted in the right place in the list keeping it ordered. Another example is a list of student records ordered based on name.

So the first part of this exercise is to create an abstract class `OrderedList` class by extending `List` and adding the following methods:

```
protected abstract int      compare(Object obj1, Object obj2);
public                  ListNode find    (Object newData) { /* to impl */ }
public                  boolean insert  (Object newData) { /* to impl */ }
public                  ListNode remove (Object remData) { /* to impl */ }
```

The *compare* method does not know what types of objects to compare, as such it is abstract making also `OrderedList` an abstract class. It is also protected as it is only used by this class and derived classes.

The *find* method should “walk-through” the list, check to see if a `ListNode` with the same data as `newData` exists by using *compare* and, if so, return it. The *remove* method should also “walk” through the list to find the sought element, but if it exists it should remove and return it. In order to do this we need to keep a reference to the previous element so that we are able to link it with the element after the one to remove.

The *insert* method should also use *compare* to insert a new object in the list while keeping it in order. It should check first using *find* if the element already exists to avoid duplicates. In order to insert an element in-between two others, we need to have a reference to the previous one and check if the element to insert is smaller than the next one, in which case it should be inserted in-between the two. So a special “walk-through” the list is also required.

Note that by implementing the `OrderedList` through inheritance, we have the problem that the `List` *insertAtFront* and *insertAtBack* methods can still be used while they should not! You should redefine them to do nothing and just print a relevant error message.

Having now implemented `OrderedList`, you should implement a subclass `IntegerOrderedList` which should simply redefine the *compare* method. The redefined *compare* method should also be protected and should now assume that the two objects are of type `Integer`, so the arguments of type `Object` should be “cast back” to `Integer`, for example `((Integer) obj1).intValue()` returns the actual integer value in argument `obj1`. A negative result should mean `obj1 < obj2`, zero should mean `obj1 = obj2` and a positive result should mean `obj1 > obj2`. You should also provide a constructor which should initialise the list name.

Having finished with the `IntegerOrderedList`, you should also implement a `StringOrderedList` which should be a “ten minute job” given that all the work is effectively done by `OrderedList` and you have already implemented `IntegerOrderedList` which is very similar (only the 1-line compare method will be different).

Having implemented `IntegerOrderedList` and `StringOrderedList`, you should write an `OrderedListTest` program that should demonstrate that these classes work correctly. This should be done by getting a series of numbers or strings from the standard input, inserting them and printing any of the two lists when required. It should also support removal of an element the user specifies. You should also implement a small menu to guide the user through activities to enter numbers/strings, print the appropriate list, remove a number/string, etc. If you find it difficult to keep state for two lists in your program (which gets full marks), you may ask the user to choose one of the two lists in the beginning of the program and then the rest of it deals with that type of list.

The second part of this assignment will use exactly the same generic class infrastructure (`List` and `OrderedList`) to implement a student record database in a similar fashion to last year’s C assignment but without saving data to and retrieving them from a file. You should start by introducing a `StudentRecord` class as follows:

```
public class StudentRecord {
    public String surname;
    public String name;
    public int     studentNo;
    public float   averageMark;

    public String toString ()           // toString, implement
    public StudentRecord (<parms>)     // constructor, implement
}
```

You should also implement a `StudentRecordOrderedList` which should be a very easy job given that all the work is effectively done by `OrderedList` and you have already implemented `IntegerOrderedList/StringOrderedList` which is very similar (only the 1-line compare method will be different). For the compare method, you should concatenate *surname* and *name* which together make the student record unique.

Having implemented the `StudentRecordOrderedList`, you should write a small program that allows the user to introduce a student record (surname, name, number), remove a student record (based on surname, name), include the average mark in a student’s record, find and print a student’s record and finally print all students’ records. For the last feature, you may re-implement `StudentRecordOrderedList toString` to print a record per line – the `List toString` prints all elements in a single line. (Note: you should treat the `List` as a “library class” for which you do not have access to its source code, hence you cannot change `List toString`).

You will realise that doing all this is very easy based on the generic object-oriented infrastructure you created (i.e. `List`, `OrderedList`) which enables us to implement lists of any content quickly and efficiently through reusability (we are reusing `List` and `OrderedList`) and extensibility (we use inheritance to implement very easily `StudentRecordOrderedList`).

You should also submit by the end of Sunday 5 April the following 3 files: *OrderedList.java*, *IntStrOrderedListTest.java* (includes `IntegerOrderedList` & `StringOrderedList` NOT as public classes and the `IntStringOrderedListTest` class with the test program) and *StudentRecordOrderedListTest.java* (includes `StudentRecord` & `StudentRecordOrderedList` NOT as public classes and the `StudentRecordOrderedListTest` class with the test program).