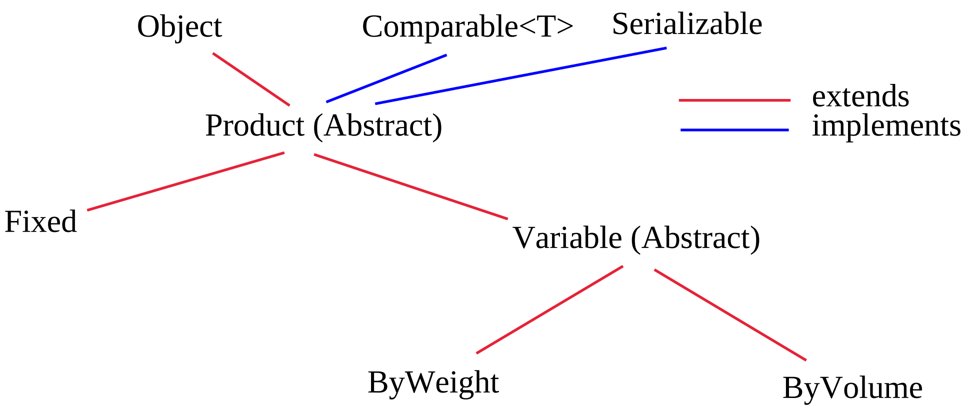
CS 0445 Spring 2024 Assignment 1

Online: Wednesday, January 17, 2024
Due: All files (see below for list) zipped into a single .zip file and submitted properly to the submission site by **11:59PM on Friday, February 2, 2024.** (Note: see the submission information page for submission details)
Late Due Date: 11:59PM on Monday, February 5, 2024

Purpose: To refresh your Java programming skills and to emphasize the object-oriented programming approach used in Java. Specifically, you will work with inheritance, polymorphism, interfaces and generics to create and test a simple array-based (enhanced) queue.

Goal: To design and implement a simple class MyBuffer<T> that will act as a buffer for accessing Java objects. To test this class you will also write a hierarchy of classes based on products found in a large multipurpose store. Your MyBuffer<T> class will primarily implement 3 interfaces – QueueInterface<T>, Reverser and SaveRestore. The details of these interfaces are explained in the files QueueInterface.java, Reverser.java and SaveRestore.java. Read these files over very carefully before implementing your MyBuffer<T> class. See also some more specific implementation requirements below.

Details: Your products must be set up using the class hierarchy shown below.



Note that the base class of your hierarchy is the Product class, which extends Object and implements Comparable<T> and Serializable. Thus, a Product reference can access any of your objects, and they can be sorted based on the compareTo() method that they all implement. They can also be written to and read from a file using the writeObject() (to an ObjectOutputStream) and readObject() (from an ObjectInputStream) methods. I have defined the Product class already in file Product.java. See that file for the code and some comments about the class. You must implement the remaining four classes correctly, each in a separate file. You should construct the hierarchy in a logical and efficient way (for example, data or operations common to subclasses should be declared in the superclass), such that the classes will work with your MyBuffer<T> and main program correctly. See file Assig1B.java and its output, A1B-out.txt to see what methods / functionality is required of your classes.

Note on Serializable: In Java, any class built from Serializable "parts" can itself also be Serializable without having to actually implement any methods. Thus, if all of your subclasses extend the Serializable class Product, and if any new data you add is also Serializable, your new classes will also be Serializable. Your additional variables should all be primitive types, which will keep the classes Serializable without any extra effort.

For the details on the functionality of your MyBuffer<T> class, carefully read over the files QueueInterface.java, SaveRestore.java, Reverser.java, Assig1A.java and Assig1B.java provided on the Web site. You must use these files as specified and **cannot remove/alter any of the code that is already written in them**.

In Lecture 4 we discussed the author's QueueInterface<T>, which is an ADT that allows for adding at the logical back (enqueue) and removing from the logical front (dequeue) of the data structure. The methods in this interface are specified in file QueueInterface.java. See the Lecture 4 Powerpoint presentation for some background and ideas about the QueueInterface<T>.

In Recitation Exercise 1 you implemented (or will implement) two simple classes called PrimQ1<T> and PrimQ2<T> that satisfy QueueInterface<T> but in an inefficient way. Specifically, they use an array that maintains one logical end or the other of the queue at index 0 and thus requires shifting for one of the enqueue() or dequeue() operations. We will discuss specific run-time analysis of these implementations a bit later in the course, but it is intuitive that there is a lot of overhead in both of these implementations.

A queue can be implemented in a more efficient way with an array if we allow both logical sides of the queue to move along the array in a circular fashion. For example, consider the array below:

0	1	2	3	4	5	6	7
		10	20	30	40	50	
		front				back	

In this queue, both front and back will move forward within the array as we enqueue() or dequeue() in the queue. For example, if we enqueue(55) in this queue, it will look as follows:

0	1	2	3	4	5	6	7
		10	20	30	40	50	55
		front					back

If we then dequeue(), the queue will look as follows:

0	1	2	3	4	5	6	7
			20	30	40	50	55
			front				back

Note that for this approach to work effectively, both back and front will need to "wrap" around the end of the array when necessary. This enables the beginning indices in the array to be reused. For example, if we now enqueue(66), the array will appear as follows:

0	1	2	3	4	5	6	7
66			20	30	40	50	55
back			front				

Note that when the queue is implemented in this way, we do not have to shift data in the array and either of the enqueue() or dequeue() methods can be implemented with just a few statements. However, there are some special cases to consider (ex: detecting a full array, handing an empty queue) so think carefully about how you would implement your class.

To implement the SaveRestore methods you will need to review Java files and their use thoroughly, in particular ObjectOutputStream and ObjectInputStream. You will also have to review exceptions, since they must be dealt with during I/O operations. You may have to look up some of this information on the Java web site using the Java API, and you may need some trial and error before completing the methods correctly. There is also information on object streams and exceptions in Chapter 12 of the Gaddis text. To help you with this, I have made a handout that demonstrates using Java ObjectOutputStream and ObjectInputStream. See OStreamDemo.java and DataType.java.

Additional Requirements for MyBuffer<T> Class

Your MyBuffer<T> class should have a constructor to set the initial capacity of the object (i.e. length of the array) but it should not be limited to that capacity. If the array is filled and an enqueue() is performed, a new array of double the size should be created and the data copied into that array (maintaining the correct queue ordering). See course notes and perhaps your CMPINF 0401 notes for resizing the array. Be careful with the resizing – the circular representation of the data makes the copying more complicated than just an index by index copy.

Your MyBuffer<T> class should also have a toString() method that produces a single String containing the size (number of items), capacity (length of the array) and contents (all items) in the MyBuffer in the relative order that they are organized (i.e. from front to back).

To see how the resizing and toString() methods are expected to work, see sample output in the file A1A-out.txt. Provided comments in the test file Assig1A.java also clarify what is expected to be done and generated.

After you have finished your coding, both the Assig1A.java file and the Assig1B.java file should compile and run correctly as given to you, and should give output similar to the output shown in the sample executions: A1A-out.txt and A1B-out.txt. I have separated the testing into 2 programs so that you can still get credit for MyBuffer<T> even if you do not get your class hierarchy working correctly.

In your single .zip file you must submit the following 12 complete, working source files for full credit:

Product.java
QueueInterface.java
EmptyQueueException.java
Reverser.java
SaveRestore.java
Assig1A.java
Assig1B.java

the **above seven files** are given to you and must not be altered in any way.

Fixed.java
Variable.java
ByVolume.java
ByWeight.java
MyBuffer.java

the **above five files** must be created / developed by you so that they work as described.

Don’t forget to **comment your code** and don’t forget to submit a completed assignment information sheet with your other files.

The idea from your submission is that your TA can compile and run your program from the command line (i.e. using the javac and java commands) WITHOUT ANY additional files, so be sure to **test it thoroughly from the command line** before submitting it. If you cannot get the programs working as given, clearly indicate any changes you made and clearly indicate why (ex: "I could not get the SaveRestore interface to work, so I eliminated code that used it") on your Assignment Information Sheet. See the CS 0445 Web site for an Assignment Information Sheet template – you do not have to use this template but your sheet should contain the same information.

Note: If you use an IDE such as Eclipse to develop your programs, make sure they will compile and run on the command line before submitting – this may require some modifications to your program (such as removing some package information). Some IDEs add code to files that produce compilation errors when compiled on the command line. **If the TA see compilation errors when compiling your program you will lose most of the credit for program execution.**

If you want to do some extra credit, here is an idea:

- Come up with an additional interface that has non-trivial functionality and also have your MyBuffer class implement this interface. To test this functionality, write an additional driver program (do NOT modify the supplied driver programs).

Alternatively, you may come up with your own idea for extra credit – if so be sure to talk to me about it before submitting. Up to 10 extra credit points (total) can be earned on this assignment.