CSci 4651 Fall 2003
Problem Set 10: Object-oriented languages II (Java).

Due Wednesday, December 10th

Problem 1 (Subtype Polymorphism). In this problem you are asked to implement and study a generic queue (i.e. a queue that works on all descendants of class `Object`). The `Queue` class must have the following methods:

- the constructor `public Queue()` to create an empty queue,
- `public boolean empty()` to check if the queue is empty,
- `public void enqueue(Object o)` to add an object to the queue,
- `public Object dequeue()` to remove the first object from the queue and return it. Recall that a queue obeys FIFO discipline (First In, First Out)

You may use the example of generic Stack in the textbook on pp. 401-402. However, please keep in mind the following:

1. You might want to use a doubly linked list (a list with references to the next and the previous element) in order to find the "oldest" element easier. However, you may use linked lists instead. Using an array is not allowed because of the restriction on the number of elements.
2. You might find the class `Node` in problem 1 of problem set 7 to be helpful (you will need to make a couple of changes to it, though).
3. The code in the textbook contains the following typos:
   - The constructor for node on p. 401 uses an `int` as the second parameter, it should be `Object`,
   - The `pop()` method of the stack (p. 402) returns `-1` if top is null, it should be `null` instead, otherwise the method does not typecheck.

Question 1. Write the `Queue class` according to the specification above. In a separate file `Test.java` write `main` to test the methods of the queue on strings (elements of class `String`). Don't forget to typecast the strings when you remove them from the queue. Submit the printout of your test code (in addition to the printout of code for `Queue`) and describe the results.
Question 2. Test the queue on a mix of elements of class `Integer` and `Double`. More specifically, write a loop such that each iteration of the loop creates one `Integer` and one `Double` based on the value of the loop counter (for instance, the integers may start at 1 and be incremented by 1, and doubles may start at 0.5 and be incremented by 0.1). As a reminder, you may create an `Integer` from an `int` like this:

```
Integer n = new Integer(2);
```

The `Integer` and the `Double` are added to the queue. As the result, Integers and Doubles are alternating on the queue.
Write another loop that removes the elements from the queue and prints them out. Make sure to do the correct alternating typecasting.
Question 3 (Arrays as objects). Since arrays are objects, the queue will work for arrays as well. Write a test program that creates 3 arrays of `int` and adds them to the queue. Then remove the arrays and print out their elements. When you remove an array from a queue, the typecasting will look like this:

```
int [] arr = (int []) q.dequeue();
```

Problem 2, THE LAST ONE. Java interfaces. Suppose we want to implement a generic priority queue. A priority queue contains elements where each element is assigned a weight. The method `dequeue` returns the element that has the highest weight.

Since the class `Object` does not contain any information about weight, we can't make a queue of items of class `Object`. Instead, we can create an interface `Weighted` which requires a single method `public double weight()` and write a `PriorityQueue` class which stores items that satisfy the `Weighted` interface. Note that `PriorityQueue` does not inherit from `Queue`, you have to write a separate class.
Question 1. Write an interface `Weighted` as described above. Then write a class `WeightedPoint` which extend the Java class `Point` and implements the interface `Weighted`. The `weight()` method of a point should return the square root of the sum of squares of the coordinates (i.e. the distance from (0,0)). Don't forget to import `java.awt.*` where `Point` is defined. Test your code.
Question 2. Write the class `PriorityQueue` that stores elements of class `Weighted`. **Important, please read carefully:** for this question you don't need to change the behavior of the queue to that of a the priority queue, just change the types. The only change in behavior from `Queue` is that `enqueue` method should print out the weight of the item. The weights can be ignored after that, the priority queue will behave just as a regular queue (First In, First Out).

Test your code on a few instances of the class `WeightedPoint` from Question 1. Submit all your code and the test code.
Question 3, Extra credit (only if you have time!). Change the methods of `PriorityQueue` to implement the correct behavior (i.e. return element with the highest weight first). Submit your code and the test code.

THAT'S ALL!