1. Give a definition of the following terms, and include a brief example.
   a) Big Oh
   b) interface
   c) generic programming
   d) stack (also briefly describe the operations supported on Stack)
2. Give an outline of how the `Object` class can be used to create generic data structures in Java. What difficulties are involved in this approach to generic programming?

3. Attached you will find the source code for the `Comparable` interface. Write a method to perform linear search that works on arrays of `Comparable` objects. Your method should return the index at which the value you are searching for is stored in the array, or -1 if the value is not present in the array.
4. Consider the following code fragment:

```java
int x;
for (int i = 0; i < N; i = i + 2) {  // Note increment clause! Does it make a
    x = x + f();  // difference?
}  // assume that f() takes constant time.
```

What is the Big Oh running time of this fragment? Briefly explain your answer.

5. Consider the following code fragment:

```java
int x;
for (int i = 0; i < N; i++) {
    for (int j = i; j < N; j++) {
        x = x + f();  // assume that f() takes constant time.
    }
}
```

What is the Big Oh running time of this fragment? Briefly explain your answer.
6. For each of the following expressions describing the running time of a program, give the Big Oh complexity class that the algorithm falls into. Briefly explain your answer.

a) $N + \frac{1}{5}N^2 + 2000$

b) $\frac{N\log N}{\log 39}$

c) $25^2N + 100^3N$

d) 10000

7. Suppose Algorithm A has running time described by the expression $1000N\log N + 25000$ and Algorithm Zero has running time described by the expression $\frac{1}{5000}N^2$. Under what circumstances will Algorithm A be faster? Under what circumstances will Algorithm Zero be faster? Which is the “best” algorithm? (Note: you don’t have to supply a specific value of N.)
8. The Stack interface is attached to the quiz. Write a function that takes a Stack as its parameter and returns a new stack which contains the data from the original stack in reverse order.

So, for example, if your stack originally looked like this:

```
7
3
2
9
8
```

then your function should return a stack that looks like this:

```
8
9
2
3
7
```

Recall that the concrete class we have used which implements the Stack interface is called StackAr.
package Supporting;

/**
 * Protocol for Comparable objects.
 * @author Mark Allen Weiss
 */
public interface Comparable
{
    /**
     * Compare this object with rhs.
     * @param Rhs the second Comparable.
     * @return 0 if two objects are equal;
     *     less than zero if this object is smaller;
     *     greater than zero if this object is larger.
     */
    int compares( Comparable rhs );

    /**
     * Compare this object with rhs.
     * @param Rhs the second Comparable.
     * @return true if this object is smaller;
     *     false otherwise.
     */
    boolean lessThan( Comparable rhs );
}
package DataStructures;
import Exceptions.*;

// Stack interface
// ******************PUBLIC OPERATIONS**********************
// void push( x )         --> Insert x
// void pop( )            --> Remove most recently inserted item
// Object top( )          --> Return most recently inserted item
// Object topAndPop( )    --> Return and remove most recent item
// boolean isEmpty( )     --> Return true if empty; else false
// void makeEmpty( )      --> Remove all items
// ******************ERRORS********************************
// top, pop, or topAndPop on empty stack

/**
 * Protocol for stacks.
 * @author Mark Allen Weiss
 */
public interface Stack
{
    /**
     * Test if the stack is logically empty.
     * @return true if empty, false otherwise.
     */
    boolean isEmpty( );

    /**
     * Get the most recently inserted item in the stack.
     * Does not alter the stack.
     * @return the most recently inserted item in the stack.
     * @exception Underflow if the stack is empty.
     */
    Object top( ) throws Underflow;

    /**
     * Remove the most recently inserted item from the stack.
     * @exception Underflow if the stack is empty.
     */
    void pop( ) throws Underflow;

    /**
     * Return and remove the most recently inserted item from the stack.
     * @return the most recently inserted item in the stack.
     * @exception Underflow if the stack is empty.
     */
    Object topAndPop( ) throws Underflow;

    /**
     * Insert a new item into the stack.
     * @param x the item to insert.
     */
    void push( Object x );

    /**
     * Make the stack logically empty.
     */
    void makeEmpty( );
}