CSc 210 Homework 7 Written Portion

1. What is the purpose of Generics? Why do we need them?

A. There are three important reasons why Generics are used:
- In order to reduce the amount of casting that is done while writing code.
- Generics help in making code type safe.
- It also makes programs look better stylistically

2. Why don't we just use Object as the parameterized type for our generic classes? Then we could put absolutely any object into our list i.e. When we made MyLinkedList generic, why don't we always declare it like:
MyLinkedList<Object> list = new MyLinkedList<Object>();

A. This declaration would not be useful since although this would mean that we could add any type of object to our list now, doing this would get rid of one of the most important reasons that we use Generics, i.e. type safety. For example,

```
list.add("Hello");
list.add("new Integer(3));
String s = list.get(2);
```

Although this piece of code would not produce a compile-time error, this would produce an exception during run-time. Therefore, this would get rid of the type safety that Generics assures us.

3. In class we make MyLinkedList generic and not MyArrayList. Why did we choose to make MyLinkedList generic first? Are there any issues when combining generics and arrays?

A. If we assume that the following code is allowed:
`MyLinkedList<String>[] array = new MyLinkedList<Integer>[2];`

Then this would mean that we can store the following instances of MyLinkedList in the array we just created.

```
array[0] = new MyLinkedList<Integer>();
array[1] = new MyLinkedList<String>();
```

The first statement, although wrong will not raise an ArrayStoreException during runtime. This is why we can't create arrays of parameterized types.

4. How many type parameters are you allowed to use in a generic class?

A. We can use any number of type parameters in a generic class.

5. Explain Type Erasure. How does it work?

A. Type Erasure refers to when the JAVA compiler erases all the type parameters and replaces all of them with Object (in case they are unbounded) or the first bound type (in case the class is bounded). This is done to ensure backwards compatibility across versions.

For an unbounded Generic class:
```
public GClass<E> {
  E data;
  public E getData(){
        return data;
  }
}
```

After the JAVA compiler runs type erasure, this turns into:
```
public GClass{
  Object data;
  public Object getData(){
        return data;
  }
}
```

This is the same for an bounded class, except that instead of Object, the type parameters are changed to the first bound.