SENG2200/6220 –Programming Languages & Paradigms
Self-Quiz Solutions for Week 5, Semester 1, 2020

True/False Questions.

1. The code below is correct?

```
LinkedList<Integer> myList = new LinkedList<Integer>();
myList.add(new Integer(10));
myList.add(new Double(1.0));
```

False

Double object cannot be added.

2. The code below is correct?

```
LinkedList<Integer> myList = new LinkedList<Integer>();
LinkedList<Object> oList = myList;
```

False

`LinkedList<Integer>` is not a subtype of `LinkedList<Object>`.

3. Generics can convert run-time errors to compile-time errors.
   True

4. Wildcard (?) can provide type safety control, or “write protection”.
   True

5. The result of the following **if** statement is true?

```
LinkedList<Integer> intList = new LinkedList<>();
LinkedList<Double> doubleList = new LinkedList<>();
if(intList.getClass() != doubleList.getClass()) …
```

False

Due to the type erasure, the raw type of intList and doubleList will be Object. The method getClass() returns the raw type of the class.

6. In bounded wildcards, supertype bounds allow read to a generic type T.
   False

7. Assume we have a generic/template class Box in Java and C++, respectively. The declaration below is allowed in both C++ and Java.

```
Box<int> yourList;
```

False

In Java, only the object types can be parameterised.

In C++, both object types and primitive types can be parameterised.

Short-Answer Questions

8. How would the Type Erasure mechanism translate the code below?

```
public class Sack<T> {
      void insert(T x) {…}
      T getRandom() {…}
}
```

```
public class Sack {
      void insert(Object x) {…}
      Object getRandom() {…}
}
```

9. What is the difference(s) between Java Generics and C++ Template during the compilation?
   - *C++ has the keyword "template".*
   - *C++ cannot restrict the types of template parameters.*
   - *The use of Java Generics results in a single compiled copy of the methods and classes.*
   - *C++ templates actually invoke a search-replace on the formal type parameter for each parameterised type used in the code at compile time - resulting in multiple compiled methods and classes.*