CS2030 Programming Methodology
Semester 2 2022/2023

15 & 16 February 2023
Problem Set #4
Java Generics

1. Consider the following JShell program fragment.

```
jshell> ImList<Integer> list = new ImList<Integer>()
list ==> []

jshell> int one = 1
one ==> 1

jshell> Integer two = 2
two ==> 2

jshell> list = list.add(one).add(two).add(3)
list ==> [1, 2, 3]
```

Which of the following code fragments will compile? If so, what is printed?

autoboxing and auto-unboxing

(a) `for (Integer num : list) { System.out.print(num + " "); }` yes: 1 2 3

(b) `for (int num : list) { System.out.print(num + " "); }` yes: 1 2 3

(c) `for (Double num : list) { System.out.print(num + " "); }` no: Integer cannot be converted to Double

(d) `for (double num : list) { System.out.print(num + " "); }` yes: 1.0 2.0 3.0

auto unboxing then int subtype of double

2. For each of the code fragments below, indicate and explain the source of the error(s).

(a)
```
List<? extends Object> list = new ArrayList<Object>()

list.add(new Object())
```
<? extends Object> type will not allow us to add elements as exact type of elements in the list is unknown, compiler cannot guarantee elements into the list

(b)
```
List<? extends Object> list = List.of("abc");

list.add("def");
String s = list.get(0);
```
<? extends Object> type will not allow us to add elements as exact type of elements in the list is unknown, compiler cannot guarantee elements into the list
Can retrieve elements from the list and assign them to a variable,, BUT can only assign it to an object, not a String

also unable to get

(c)
```
List<? super Integer> list = new List<Object>();

list.add(new Object())
```
Unable to create new List, as List is an interface and we cannot create an instance of an interface directly. Need to create an instance of a class that implements the list interface e.g. ArrayList or LinkedList

(d) `List<? super Integer> list = new ArrayList<int>();`
Need to be <Integer> instead of <int> as type parameter must be a reference type instead of a primitive type

(e) `List<? super Integer> list = new ArrayList();`
Did not declare type of ArrayList

(f)
```
List<?> list = new ArrayList<String>();
list.add("abc");
```
List<?> is the same as List<? extends Object>

Unable to add elements to the list because the type of the elements is unknown due to the usage of the unbounded wildcard

3. In the lecture, we have seen the use of the `Comparator<T>` interface with the method specification `int compare(T t1, T t2)` that returns zero if `t1` and `t2` are equal, a negative integer if `t1` is less than `t2`, or a positive integer if `t2` is less than `t1`.

```
public interface Comparator<T> { // <T> declared with class scope
    int compare(T o1, T o2);
    ... o1 - o2
}
```

A generic method `T max3(T a, T b, T c, Comparator<T> comp)` can be defined in JShell as shown below. The method takes in three values of type `T` as well as a `Comparator<T>`, and returns the maximum among the values.

```
jshell> <T> T max3(T a, T b, T c, Comparator<T> comp) { // <T> declared with
   ...>     T max = a;                                   // method scope
   ...>     if (comp.compare(b, max) > 0) {
   ...>         max = b;
   ...>     }
   ...>     if (comp.compare(c, max) > 0) {
   ...>         max = c;
   ...>     }
   ...>     return max;
   ...> }
|  created method max3(T,T,T,Comparator<T>)
```

int answer = max3(1,2,3, new Comparator<Integer>())

(a) Demonstrate how the `max3` method can be called so as to return the maximum of three integers -1, 2 and -3.

(b) Other than `Comparator<T>`, there is a similar `Comparable<T>` interface with the method specification `int compareTo(T o)`. This allows one `Comparable` object to compare itself against another `Comparable` object.

```
public interface Comparable<T> {
    int compareTo(T o);
}
```

As an example, since `Integer` class implements `Comparable<Integer>`,

```
jshell> Integer i = 1 // 1 autoboxed to an Integer and assigned to i
i ==> 1


jshell> i.compareTo(2) // 2 autoboxed to an Integer and passed to compareTo
$.. ==> -1
```

Let's redefine the `max3` method to make use of the `Comparable` interface instead.

```
<T> T max3(T a, T b, T c) {
    T max = a;
    if (b.compareTo(max) > 0) {
        max = b;
    }
    if (c.compareTo(max) > 0) {
        max = c;
    }
    return max;
}
```

Integer class implements Comparable<Integer> , but since <T> is generic, there will be other classes that do not implement the Comparable interface and will be unable run this.
Thus, fail to compile

Parameters a, b, and c are of type Comparable<T>. This means that the compareTo method is available for these parameters, and the compiler will know that they are comparable.

However, you must cast the results of the comparison to type T e.g. max = (T) b

Does the above method work? What is the compilation error?

(c) Does the following declaration of `max3` work?

```
<T> T max3 (Comparable<T> a, Comparable<T> b, Comparable<T> c)
```

(d) To restrict `T` to have the `compareTo` method, i.e. any class that binds to `T` must implement `Comparable`, we redefine the type parameter `<T>` to be `<T extends Comparable<T>>`.

```
<T extends Comparable<T>> T max3(T a, T b, T c) {
    T max = a;
    if (b.compareTo(max) > 0) {
        max = b;
    }
    if (c.compareTo(max) > 0) {
        max = c;
    }
    return max;
}
```

Demonstrate how the method `max3` can be used to find the maximum of three values -1, 2 and -3. Explain how it works now.

The syntax <T extends Comparable<T>> is a type parameter constraint in Java generics. It specifies that the type parameter T must be a subtype of the Comparable interface, which is a generic interface that specifies a single method compareTo for comparing objects of the same type.

This means that any class or interface used as the type argument for T must implement the Comparable interface with itself as the type parameter.

With this constraint in place, the method will be able to use compareTo from the Comparable interface and return the largest value