to call constructor: classname<generic type>(param)
to call other method: classname.<generic type>method(param)

CS2030 Programming Methodology

Semester 2 2019/2020

20 February 2020
Problem Set #5

1. The following static generic method `max3` that takes in an array of generic type `T` that such that `T` implements the `Comparable` interface.

```
static <T extends Comparable<T>> T max3(T[] arr) {
    T max = arr[0];
    if (arr[1].compareTo(max) > 0) {
        max = arr[1];
    }
    if (arr[2].compareTo(max) > 0) {
        max = arr[2];
    }
    return max;
}
```

need to extend comparable in order to call compareTo

a,b/ cannot assign arr[0] to max -> try to fix by changing max to Comparable<T>. Still got error because compareTo takes in a variable T not Comparable<T>
=> better not change T[] to Comparable<T>[]
=> still can try to fix it by:
+change max to Comparable<T>
+casting max into (T)max in compareTo()
+@SuppressWarnings("unchecked")
-> may cause runtime error

What happens if we replace the method header with each of the following:

(a) `static <T> Comparable<T> max3(Comparable<T>[] arr)`

(b) `static <T> T max3 (Comparable<T>[] arr)`

(c) `static Comparable max3(Comparable[] arr)`

might cause compile time or runtime error

2. Suppose a `Fruit` class implements the `Comparable` interface, and `Orange` is a sub-class of `Fruit`, how would you change the `max3` method header in question 1 such that the parameter type is `max3` is `List<T>` instead? You should aim to make the method as flexible as you can. static <T extends Comparable<? super T>> T max3 (List<T> list) -> T is orange or
static <T extends Comparable<T>> T max3 (List<? extends T) -> T is fruit

3. Compile and run the following program fragments and explain your observations.

java type eraser: don't create new class for each generic type, instead, create a bridge method and erase the type (during compile time)

(a)
```
import java.util.List;

class A {
    void foo(List<Integer> integerList) {}
    void foo(List<String> StringList) {}
}
```

error because both foo method's erased class is the same. Type erasure replace generic type T with object

(b)
```
class B<T> {
    T x;
    static T y;
}
```

cannot because static variables are declared before the initialization of an object.

(c)
```
class C<T> {
    static int b = 0;

    C() {
        this.b++;      better use C.b++
    }

    public static void main(String[] args) {
        C<Integer> x = new C<>();
        C<String> y = new C<>();

        System.out.println(x.b);
        System.out.println(y.b);
    }
}
```

C<Integer> and C<String> is the same class => same static variable

2
2

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

(a)
```
List<Integer> list = new ArrayList<>();
int one = 1;
Integer two = 2;

list.add(one);     (auto boxing)
list.add(two);
list.add(3);       (auto boxing)

for (Integer num : list) {
    System.out.println(num);
}
```

1 2 3

(b)
```
List<Integer> list = new ArrayList<>();
int one = 1;
Integer two = 2;

list.add(one);
list.add(two);
list.add(3);

for (int num : list) {        auto unboxing
    System.out.println(num);
}
```

(c)
```
List<Integer> list = Arrays.asList(1, 2, 3);

for (Double num : list) {
      System.out.println(num);
}
```

error, cannot box int into a Double

(d)
```
List<Integer> list = Arrays.asList(1, 2, 3);

for (double num : list) {
    System.out.println(num);
}
```
auto unboxing Integer -> int, auto typecasting int -> double

(e)
```
List<Integer> list = new LinkedList<>();
list.add(5);
list.add(4);
list.add(3);
list.add(2);
list.add(1);

Iterator<Integer> it = list.iterator();
while (it.hasNext()) {
    System.out.println(it.next());
}
```
add(E e)

auto boxing