Purpose	Practice using type parameters and Streams
What to Submit	Commit a revised coinpurse.CoinUtil with test code to your Purse project .

Generics

1. In the Purse coinpurse.CoinUtil class write a max method using a type parameter like this::

```
/**
 * Return the larger of a and b, according to the natural
 * ordering (defined by compareTo).
 */
public static <E extends Comparable<E>> E max(E a, E b) {
    //TODO write code to return the "larger" of a and b
}
```

Verify that this method works by using it on at least **2 different classes of objects**. For example:

```
    String m = CoinUtil.max( "canary", "dog" );
    System.out.println("Max of canary and dog is "+m);
    //TODO compute max using some other kind of object
```

2.1 The method only works for types E that implement Comparable<E>. It should not work for Coin. Verify for yourself that this code does not compile:

```
    Coin max = CoinUtil.max( new Coin(5), new Coin(10) );
```

2.2 Modify the type bounds for <E> so that it accepts any type E that implements Comparable<E> or Comparable<Some_Super_Class_Of_E>, like this:

```
    <E extends Comparable<? super E>>
```

Verify that now your can call "max" with Coin, String, or other types.

3. In CoinUtil there is a sortByCurrency method like this:

```
    public static void sortByCurrency(List<Valuable> money)
```

It cannot be called using a parameter like this:

```
List<Coin> coins =
     Arrays.asList( new Coin(5,"Baht"), new Coin(100,"Kip"), . . . );
CoinUtil.sortByCurrency( coins );
```

Add a wildcard (?) to the sortByCurrency method signature so it accepts List<Coin>, and the method still works correctly.

4.1 (This is more challenging) Add a type parameter to filterByCurrency so that it accepts a List of anything that implements *Valuable*, and returns a List of the same type.

This means if you invoke it with parameter List<Banknote> then it should return List<Banknote>. If Coupon implements Valuable and you call it with List<Coupon> then it should return List<Coupon>.

4.2 Write example code to prove that filterByCurrency works as specified.

5. **Variable Length Parameters**: Java allows a method to have a variable number of parmeters. The parameters are automatically put into an array using the name of the parameter. Here is an example:

```
// This method accepts any number of String parameters
public void printAll(String ... word) {
   // parameters are word[0], word[1], ...
   // Careful: the length of array may be zero!
   for(int k=0; k < word.length; k++)
       System.out.printf("param %d is %s\n", k, word[k]);
}
```

5.1 Modify the **CoinUtil.max( )** method it computes the "max" of any number of items.

```
Coin c1 = new Coin(5);
Coin c2 = new Coin(10);
Coin c3 = new Coin(0.5);
Coin cmax = max( c1, c2, c3 );
```

5.2 Does variable parameters allow us to mix objects of different subclasses of some superclass? In this case, what is the return type?

```
Coin c1 = new Coin(5);
Coin c2 = new Coin(10);
Banknote banknote = new Banknote(100,"Baht");
________?_______ cmax = max( c1, c2, banknote );
```

Streams

We can create a Stream from any collection by calling stream(). The main Stream methods are described in the PDF in the week14 folder. Here is an example that filters Strings have length less than or equal to 4.

```
List<String> words =
     Arrays.asList("Dog", "elephant", "Bird", "Zebra", "snake");
// Define a Predicate that is true fo the objects we want:
Predicate<String> shortWords = (s) -> (s.length() <= 4);
// Print the whole list
words.stream().forEach( System.out::println );
// Filter and print sublist
words.stream().filter( shortWords ).forEach(System.out::println);
```

In the above example we *consume* the stream using forEach. You can also *collect* the stream into an object using a Collector. The Java Collectors class provides many useful predefined collectors, such as Collectors.asList(). Collectiors.asList() puts the stream into a List and returns it.

```
// Create a list of words with length <= 4
List<String> result =
words.stream().filter( shortWords ).collect( Colletors.asList() );
```

6. Modify the CoinUtil.filterByCurrency method to use a stream and a Predicate as filter. You can write Predicate as a lambda or anonymous class. When you finish, the method should not have any loops. It should have 3 statements: 1) check that currency param is not null, 2) define a Predicate as filter, 3) use a Stream to filter the list and return the result.