Purpose	Practice using type parameters and wildcards on methods. Review and fix your MoneyUtil class. Write the best code you can.
What to Submit	1. Commit a revised MoneyUtil class to the Purse project. 2. Write some tests in the main method to show that you have correctly implemented each problem. 3. **Clean up and fix code in MoneyUtil**. There should be no errors in any methods. We will evaluate methods as correct or incorrect -- no partial credit if any error. 4. Make sure code is good quality. We will also give a Code Quality score, again 0 or 1. Badly formatting code, missing Javadoc, or misleading variables names (example below) will all result in a score of 0 (incorrect). For example: `public void sortMoney(List<Valuable> coins)` "coins" is a misleading name, since the List is not required to be Coins. 5. Eliminate dead code and commented out code. This is the final submission, so there shouldn't be any unnecessary code.
Deadline	Saturday, 28 April.

Generics

1. In the Purse **`MoneyUtil`** 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).
 * @param args one or more Comparable objects to compare.
 * @throws IllegalArgumentException if no arguments given.
 */
public static <E extends Comparable<E>> E max(E ... args) {
    //TODO write code to return the "largest" argument
}
```

"E ... args" means a *variable number of parameters*. You can invoke max("a"), max("a","b"), max("a","b","c"). Inside the method, the actual argument values are args[0], args[1], args[2], ...

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

```
    String max = MoneyUtil.max( "dog", "zebra", "cat" );
```

2. Fix the above method! The method only works for types E that implement Comparable<E>.

It doesn't work for subclasses of Money because Money implements Comparable<Valuable>.

```
     Money m1 = new Banknote(100, "Baht");
     Money m2 = new Banknote(500, "Baht");
     Money m3 = new Coin(20, "Baht");
     Money max = MoneyUtil.max( m1, m2, m3 );
```

Fix the way the type parameter is specified so that max works for Money, Coin, Banknote, etc.

See the Java API for **`sort`** method in **`java.util.Collections`** for example of what to do.

3.1 Refactor: Rename MoneyUtil.sortCoins to MoneyUtil.sortMoney.

Use refactoring so that references to this method are updated.

3.2 There is a problem with this method: **`sortMoney(List<Valuable> money).`**

It *doesn't work* for cases like this:

```
     List<Banknote> list = new ArrayList<Banknote>();
     list.add( new Banknote(10.0, "USD") );
     list.add( new Banknote(500.0, "Baht") );
     MoneyUtil.sortMoney( list );
```

Use a *wildcard (?) with qualifier* so that sortMoney can sort any list of stuff that implements Valuable, such as List<BankNote>, List<Coin>, List<Money>, but not for things that *don't* implement Valuable.

Don't make this a Generic method. All you need is a wildcard (?) with qualifier, not a type parameter.

4. **`MoneyUtil.filerByCurrency`** has a similar problem. It doesn't work with List<BankNote>, etc.

If the *parameter* is List<Coin> then filterByCurrency() should *return* List<Coin>, if the paramater is List<Coupon> then it should return List<Coupon>. (Provided that Coupon implements *Valuable*).
For example:

```
List<Coin> coins = Arrays.asList( new Coin(5,"Baht"),

                   new Coin(0.1,"Ringgit"), new Coin(5,"Cent") );

List<Coin> result = MoneyUtil.filterByCurrency(coins, "Baht"); // Error
```

Add a generic type parameter to the method, so it accepts List<E> and returns List<E> for any E that implements *Valuable*.