Principles of Java Language with Applications, PIC20a
O. Azencot, originally created by E. Ryu
Spring 2018

Homework 5
Due 5pm, Friday, June 8, 2018

Download the starter code `MetaCollection.java` and `Test.java`. Your code must work with `Test.java`. Put everything in the package `hw5`. Submit `PhoneUtil.java`, `ListUtil.java`, and `MetaCollection.java`.

Problem 1: Imagine you have a `Map<String, BigInteger>` that represents a phone book. Until now, you stored all (US) phone numbers as the usual 10 digit number. (These numbers do not start with a `0`.)

One day, your business partner from London gives you the phone number `44-020-1234-1234`, and you realize that you need to add a `1` to the beginning of all existing phone number entries to indicate that they are US phone numbers. (`1` is the country code for the US.)

Write the function

```
public static void prependOne(Map<String, BigInteger> m)
```

that adds a `1` to the beginning of each 10-digit `BigInteger`. Place this function within the utility `class PhoneUtil`.

Problem 2: Read the documentation of `java.util.function.Predicate<T>` and `java.util.AbstractCollection<E>`.

Problem 3: Write a utility `class ListUtil` that provides the following 2 methods.

```
public static <E> ArrayList<E> merge(
            Collection<? extends E> c1, Collection<? extends E> c2)
```

`merge` returns an `ArrayList<E>` that contains all elements of `c1` and `c1`. You will find the `addAll` method of `List<E>` useful.

```
public static <E> ArrayList<E> select(
           Collection<? extends E> coll, Predicate<? super E> pred)
```

`select` returns an `ArrayList<E>` that contains all elements of `coll` for which `pred.test(...)` evaluates to `true`.

Problem 4: Write the generic `class MetaCollection<E>`, which serves as a concatenation of many `Collection<E>`s. Make `MetaCollection<E>` inherit `AbstractCollection<E>`.

For efficiency reasons,

```
private ArrayList<Collection<E>> collectionList;
```

is the only `Collection<T>` you may use as a field. In particular, you may not create additional `Collection<E>`s.

Write the constructor

```
public MetaCollection(Collection<E>... c_arr)
```

which takes in a variable number of `Collection<E>`s, and initializes the `MetaCollection<E>` to be the concatenation of them.

Write the method

```
public void addCollection(Collection<E> coll)
```

so that it adds `coll` to the `MetaCollection<E>`. (So there are 2 ways to add a `Collection<E>` to the `MetaCollection<E>`. One is via the constructor and the other is via `addCollection(...)`.)

Implement the method

```
public int size()
```

so that it returns the sum of the sizes of the `Collection<E>`s the `MetaCollection<E>` consists of.

Implement the method

```
public Iterator<E> iterator()
```

so that it simply returns a `JoinedIter` instance.

Write the `private` inner `class`

```
private class JoinedIter implements Iterator<E>
```

A `JoinedIter` instance iterates through all `E`s of the `Collection<E>`s the `MetaCollection<E>` consists of.

Remark. Do not define the `class` as

```
private class JoinedIter<E> implements Iterator<E> //don't do this
```

If you do so, the generic type `E` of the inner `class` hides the generic type `E` of the top-level `class`, which is not what you want.

Hint. Let `JoinedIter` have a field `private int itrCounter`. Initialize `itrCounter` to `0` and increment it every time `next()` is called. Then `hasNext()` can `return (itrCounter<size())`.

Hint. `JoinedIter` must use an `Iterator<E>` when iterating through a given `Collection<E>`. `JoinedIter` can keep track of which `Collection<E>` within `collectionList` it is currently going through with an `Iterator<Collection<E>>` or an index stored as an `int`.

Remark. The two-tiered structure of this problem is initially confusing, but a correct solution can be simple. If your code gets long and complicated, try to see if you can simplify your logic. As a guideline, `JoinedIter` can be written with 20 lines of code.

Problem 5: In the last lines of `Test.java`, we have

```java
ArrayList<Complex> l3 = new ArrayList<>();
l3.add(new Complex(1,2));
ArrayList<Complex> l4 = new ArrayList<>();
l4.add(new Complex(2,3));

ArrayList<Complex> l5 = ListUtil.merge(l3, l4);
Complex cpx = l5.get(0);
cpx.real = 0;
cpx.imag = 0;
l5.set(1, new Complex(8,9));

System.out.println("Why do we have the following output?");
System.out.println(l3.get(0)); //0.0+0.0i
System.out.println(l4.get(0)); //2.0+3.0i
System.out.println(l5.get(0)); //0.0+0.0i
System.out.println(l5.get(1)); //8.0+9.0i
```

Think about why the `0`th entry of `l3` did change while the `0`th entry of `l4` did not.