CSCI-201: Principles of Software Development

Spring 2022

Lab04: Collections

Introduction:

Generic programming involves the design and implementation of data structures and algorithms that work for multiple types. You are already familiar with the generics in Java. In this lab you are to implement a few generic methods to work with collections. In addition to dealing with bounded type parameters, you will work with predicates to see how they are useful in generic programming. The programming is not difficult (and short) but requires careful thought. Note the following two cases when you work with bounded type parameters:

1. If you need to go through a collection and do things with each item, then the collection is a producer, so you should use <? extends >.
2. If you need to add things to a collection, then the collection is a consumer, so you should use <? super >.

Implementation:

Write a generic method called concat() that takes two Iterable arguments. The method combines two iterables of the same abstract type into a single iterable. The method returns that new iterable. Note, the returning type must be Iterable, and not Iterator. Also, make sure you did not modify the original iterables in any way. You are not allowed to use Java's streams, and extra packages (like Guava).

Write a generic method called reverseMap() that takes a single Map argument of any type and returns the inverse of the given Map. The method transforms "Key to Value" to "Value to Key". You may assume that all values are unique. The method returns that new Map. Make sure you did not modify the original Map in any way.

In the following three methods you are to use the Predicate interface. You are not allowed to use Java's streams, and extra packages (like Guava).

A predicate is a boolean-valued function with one parameter:

```
interface Predicate<T> {
        public boolean test(T obj);
}
```

If an object implements this interface it knows how to "test" objects of type T in some way. Java already has some methods that use predicates (in java.util.function), however here you write a few similar methods yourself. You are not allowed to use Java's streams, and extra packages (like Guava).
Write a generic method called collect() that takes two arguments: Iterable and Predicate. The method collects from the provided iterable all elements that satisfy the provided predicate. The method returns a new iterable. Note, the returning type must be Iterable, and not Iterator. Also, make sure you did not modify the original iterable in any way.

Write a generic method called remove() that takes two arguments: Iterable and Predicate. The method removes from the provided iterable all elements that satisfy the provided predicate. The method is void; it modifies the original iterable accordingly.

Write a generic method called countIf() that takes two arguments: Iterable and Predicate. The method counts the number of all elements the provided iterable that satisfy the provided predicate. The method returns int. Make sure you did not modify the original iterable in any way.

Testing:

The testing driver is provided. Here is the expected output:

```
Concatenated iterable: [Dog:5, Dog:8, Dog:2, Terrier:1, Terrier:11, Terrier:88, Terrier:12, ]
Original Map: {Terrier:3=dandie, Dog:1=beagle, Dog:2=poodle, Dog:1=labrador, Terrier:3=irish}
Inverted Map: {dandie=Terrier:3, labrador=Dog:1, poodle=Dog:2, beagle=Dog:1, irish=Terrier:3}
Original collection: [499, 567, 311, 104, 200, 201, 570, 270, 1999]
Count evens: 4
Remove evens: [499, 567, 311, 201, 1999]
Retain evens: [104, 200, 570, 270, ]
Original collection: [499, 567, 311, 104, 200, 201, 570, 270, 1999]
Count bigger than 270: 5
Remove bigger than 270: [104, 200, 201, 270]
Collect bigger than 270: [499, 567, 311, 570, 1999, ]
Original collection: [Dog:5, Dog:8, Dog:2, Terrier:1]
Count evens: 2
Count evens: 2
```

Grading Criteria:

Labs are graded based on your understanding of the course material. To receive full credit, you will need to:

1) complete the lab following the instructions above
2) show your understanding of the lab material by answering questions upon check-off

If there is a discrepancy between your understanding of the material and your implementation (i.e. if your code is someone else's work), you will receive a grade of 0 for the lab. Please note, it is the professor's discretion to report the incident to SJACS.

Implementation: **(8 Points)**

a) collect() is implemented correctly (2 pts)

b) reverseMap() is implemented correctly (1 pts)

c) remove() is implemented correctly (1 pts)

d) countIf() is implemented correctly (1 pts)

e) concat() is implemented correctly (3 pts)

Check-off Questions: **(2 Points)**

Please randomly select one question.

Question 1: Which of the following assignments are incorrect and why?

a. List test1 = new ArrayList<Integer>();

b. List<?> test2 = new ArrayList<Integer>();

c. List<?> test3 = new ArrayList<?>();

d. List<Object> test4 = new ArrayList<Integer>();

Question 2: Explain why these code compile but not run.

```
Integer[] intArr = new Integer[5];
Number[] numArr = intArr;
numArr[0] = 1.23;
```

Question 3: The following code does not compile.

```
ArrayList<Integer> inList = new ArrayList<>();
ArrayList<Number> numList = intList;
```

How do you fix the numList type to make that assignment work for a Number type?

Question 4: The following code does not compile.

```
List<Number> numList = new ArrayList<Integer>();
```

How do you fix the numList type to make it work for Integer and Number types?

Question 5: Explain why the following code does not compile.

```
List<? extends String> strList = List.of("apple", "orange");
strList.add("fruit");
```