List Methods and Strings

Dr. Andrew Rosen

September 2, 2018

In this exercise, you will learn to write static methods that use `List`s and Strings. Each of these is around the difficulty of some exam questions. If you need to make a new `List` as part of the method, implement it using an `ArrayList`. Read footnotes for hints.

Your answers for this homework will be used for your next homework.

1 Static Generic Methods

Most of the methods we write in class *won't* be static, but that's no reason not to learn how to do that. Java makes writing generic methods look intimidating, but in reality, it's not so bad. I'll walk you through how to do it by showing what we want to do, hitting an error, and then showing you how to resolve the error.

Suppose we want to write a method called `in`, which given a `List` and an item, checks to see if the `List` contains that item. [1] We don't know what type the item will, nor do we know what kind of stuff the List will be holding, as that will change from one program to another, and we want this method to be able to be used in any kind of context. That means we want it to be generic. However, if we write

```
//this is an error
public static boolean in(List<E> list,  E item){

}
```

We get "E cannot be resolved to a type" as an error. This happens because Java doesn't know that you want to use `E` as the symbol for generics for this method. We can fix this by adding a `<E>` in between `static` and our return type, like so:

```
public static <E> boolean in(List<E> list,  E item){

}
```

The big difference here between a class that uses a generic, like `ArrayList`, and the static methods you write here is that the generic type only exists for the method.

1.1 When Not to Use The Above

You do not have to have method methods generic when we know what types we are using. For instance, if we know we are dealing with a list of integers (`List<Integer>`), we **don't** have to write anything generic in. We just write:

```
public static returnType myMethod(List<Integer> list){

}
```

For each method, ask yourself "Does this method need to work on a `List` of any type, or just a single type?" If any type, you need to make a generic method. If a single type, then you don't need to use a generic in that method.

[1] This exists as an instance method under a different name within any `List`, but let's pretend it doesn't.

2 The Methods

For each of the following, create a **static** method with the appropriate inputs and outputs. Call each of them in the `main` method.

2.1 Uniqueness

Write a method called `unique()` which takes in a `List` and returns true if all the items in the `List` are unique. All the items are unique if none of them are the same.[2] Return false otherwise.

2.2 All Multiples

Write a method named `allMultiples()` which takes in a `List` of integers and an `int`. The method returns a new `List` of integers which contains all of the numbers from the input list which are multiples of the given `int`. For example, if the `List` is $[1, 25, 2, 5, 30, 19, 57, 2, 25]$ and 5 was provided, the new list should contain $[25, 5, 30, 25]$.

2.3 All Strings of Size

Write a method named `allStringsOfSize()` which takes in a `List<String>` and an `int length`. The method returns a new `List<String>` which contains all the Strings from the original list that are `length` characters long. For example, if the inputs are `["I", "like", "to", "eat", "eat", "eat", "apples", "and", "bananas"]` and `3`, the new list is `["eat", "eat", "eat", "and"]`.

2.4 isPermutation

Write a method called `isPermutaion()` which takes in two `List` objects which contain the same types. It returns `true` if the Lists are permutations of each other and false if not. Two lists are permutations if they contain the same elements, including the same number of duplicates, but they don't have to contain the elements in the same order. For example, `[1,2, 4]` and `[2,1,4]` are permutations of each other.

2.5 String To List of Words

Write a method called `stringToListOfWords()` which takes in a `String` converts it into a list of words. We assumes that each word in the input string is separated by whitespace.[3]

[2] Use the equals() method.

[3] The `split()` method of `String` can split an input up along a provided special string called a regular expression or *regex*. A regex is much like a code for pattern, and `split()` will match anything in that pattern. The regex `"\\s+"` matches any amount of whitespace.

If our input String is `"Hello, world!"`, then the output should be `["Hello,", "world!"]`. For extra credit, sanitize the String, cleaning it up so that we remove the punctuation and other extraneous characters such that the output in the above example would become `["Hello", "world"]`

This method returns `List<String>`.

2.6 Remove All Instances

Write a method called `removeAllInstances()` which takes in a `List` and item[4]. The method then proceeds to remove each item in the list that matches the given item. For example, if the method is passed the `List<Integer> [1, 4, 5, 6, 5, 5, 2]` and the `Integer 5`, the method removes all `5`'s from the `List`. The `List` then becomes `[1, 4, 6, 2]`. It should return nothing, since the changes the `List` it was provided. [5]

3 Your Grade

Turn in your assignment on Canvas and come see the Professor or a TA to demo your work for credit.

90 points Each method working properly is worth 15 points. 5 points of each method is for using generics correctly.

10 points Code is neat and properly indented.

5 points extra credit Described in the `stringToListOfWords()` section.

3.1 Demoing

Be prepared to explain how you solved `removeAllInstances()` and that it works on the test cases, as well as one other method the TA will choose.

[4] In other words, the first parameter is a list of generics and the other input is a single item of the same type the list holds.

[5] This one is extremely tricky, since removing an item shifts the indexes.