COMP 2522
Object Oriented Programming
Lab 5

Yue Wang
`yue_wang@bcit.ca`

Due at or before 23:59:59 on Sunnday, February 26th, 2023

1 Introduction

For your next lab, you will apply what you have learned about OOP, generics, and inner classes in COMP 2522. You will implement a novel and simple data structure that we will call the **ArraySet**.

There is no ArraySet in the Java Collections Framework, but there is a Set<E> interface in the java.util package. You have all the tools and knowledge you need to implement this interface. Our ArraySet is a parameterized data structure similar the ArrayList – it uses an array "under the hood" to implement a data structure (in this case, a set instead of a list).

Recall that a set is a collection of items without duplicates and in no particular order (i.e., unordered). In Java, a set cannot contain nulls, either. A user may add items to a set, remove items from a set, and check whether an item is in a particular set. Occasionally, the elements in the set must be accessed one at a time; for instance, this is necessary to list the set elements to the standard output. Your ArraySet class must be a generic class that implements the Set interface.

2 Submission requirements

This lab must be completed and submitted to D2L at or before 23:59:59 on Sunnday, February 26th, 2023.

Don't wait until the last minute as you never know what might happen (computer issues, no Internet, power outage, ...). Submit often, and please make sure your comments are short and clear and specific.

You must not ignore Checkstyle warnings or the code style warnings produced by IntelliJ. **You must style your code so that it does not generate any warnings.**

When code is underlined in red, IntelliJ is telling us there is an error. Mouse over the error to (hopefully!) learn more.

When code is underlined in a different colour, we are being warned about something. We can click the warning triangle in the upper right hand corner of the editor window, or we can mouse over the warning to learn more. If the warning seems silly, tell me about it, and I will investigate and possibly ask the class to modify some settings to eliminate it once and for all!

3 Grading scheme

Your lab will be marked out of 10 and converted to a grade out of 5. Consult the rubric available on D2L. On a high level, expect that marks will be assigned for:

Figure 1: This lab, like all labs, will be graded out of 5

1. (7 points) Meeting the functional requirements in this lab, i.e., does the code do what it is supposed to do? We have unit tests for this!
2. (1.5 points) Meeting non-functional Java OOP design requirements such as encapsulation, information hiding, minimizing mutability, maximizing cohesion and minimizing coupling, making good choices about associations and thinking about Demeter, etc.
3. (1.5 points) Writing code that is self-documenting and well-styled that generates no warnings from IntelliJ or Checkstyle, with correct and complete Javadocs, using short and atomic methods correctly encapsulated in classes that are well defined.

4 Style Requirements

Continuing with this lab, there are code style requirements that you must observe:

1. Your code must compile.
2. Your code must not generate any IntelliJ code warnings or problems.
3. Your code must execute without crashing.
4. Your code must not generate any Checkstyle complaints (unless they are complains I have specifically said you can ignore).
5. Don't squeeze your code together. Put a blank space on either side of your operands, for example. I will be assessing the readability and clarity of your code.
6. All of your program classes must be in package ca.bcit.comp2522.xxx (replace xxx as required by the assignment, lab, quiz, etc.). For example, today's work should go in the ca.bcit.comp2522.labs.lab0X package.
7. All classes require Javadoc comments including the @author tag AND the @version tag. Class comments go after the package and import statements. There should be no blank lines between a class comment and the class it describes.
8. Public constants require Javadoc comments, private constants do not. Constants should be the first thing in your class.
9. Constants should be static and final, are often public, and should be followed by instance variables.
10. Instance variables have private or protected visibility, never public or default visibility.
11. Public and protected methods require Javadoc comments including @param tag(s)the @return tag, and the @throws tag where needed (we won't worry about throws until we talk about exceptions in depth).
12. A method's comment must begin with verbs describing what the method does, i.e., Calculates, Returns, Sets, Prints, etc. Note that we use present tense in Java – Returns, not Return. Prints, not Print.
13. The @return and @param tags go AFTER the description.
14. Private methods require non-Javadoc comments (the comments that start with a slash and a single asterisk).
15. Do comment complicated logical blocks of code inside methods with sparse, clear inline comments.

16. Do not use magic numbers (you must use constants instead). Remember that a magic number is any numeric literal. A constant can be local in a method (use the final keyword with it) or class-level (make it static and ALL_CAPS).
17. All method parameters that are object references must be made final (so we don't forget parameters are passed by value):
    (a) Nice to prevent erroneous assignments, and necessary if parameter is referenced by inner class, but that is perhaps a little advanced for now.
    (b) References made final mean that the reference, once pointing to an object, cannot be changed to point at a different object.
18. Consider making your methods final:
    (a) Making a method final prevents subclasses (those that inherit the method) from changing its meaning.
    (b) Final methods are more efficient (the methods become inline, thus avoiding the stack and generating overhead).
19. Data and methods that work together must be encapsulated in the same class.
20. Code duplication must be minimized.
21. The values of local variables that are primitives are set when they are declared, and local variables are not declared until they are needed.
22. Every class that stores state needs an equals method, a hashCode method, and a toString method.
23. In general, we enforce a fairly hard maximum method length of 20 lines of code. Aim for 10 (excluding braces).

5 Coding Exercise

The following must be added to ca.bcit.comp2522.lab05:

1. You must complete the ArraySet class for me. This class is-a Set.
2. An implementation of the Set interface can be realized in more than one way. **For your assignment, you must implement the Set by using an array whose elements are of type E.**
3. Begin by noticing that I have already created the class header for you. Modify it so your class implements the Set<E> interface and the Iterable<E> interface.
4. **You must implement the methods from the Set<E> interface and the Iterable<E> interface.** When you mouse over the red warning underline, the pop-up window suggests that you implement the methods. Do not implement spliterator( ), it already has a default implementation and I think it's too much work right now, but the rest of the interface methods must be implemented.
5. **Your implementation of the ArraySet class must support initially holding 10 items, i.e., it must have a capacity of 10.** If there is an attempt to add one more item than the current size allows, i.e., the current size has increased to be equal to the capacity, the size of the set (its capacity) must double. The recommended way to do this is to quietly create a new array that is twice the size of the current array and secretly move all of the data from the current array to the new array.
6. When an item is added to the ArraySet, I recommend that you add the item to the end of the array. Do not try to keep track of null entries in the underlying array because this will make monitoring the capacity and the load difficult. When an item is removed from the ArraySet, I suggest you replace it with the item at the end of the array to "plug" the hole. That is, move the element at the end of the array into the recently vacated slot. Remember this is okay because the ArraySet doesn't maintain sequence. In the case of removal, your implementation must not move more than one element to remain efficient.

7. **Your class must be a generic type like the ArrayList.** I encourage you to look at the ArrayList.java source file that was installed on your computer in a file called src.zip when you installed Java. Check out the implementations of the Set interface for some tips, too!

8. **You will need to create another helper class that implements the Iterator interface** in order to satisfy the iterator( ) method. I suggest you use an inner class called, oh, I don't know, how about ArraySetIterator<E> and return an instance of this from iterator( ). If ArraySetIterator implements Iterator<E>, which it should, then it is-a Iterator<E>. You must implement hasNext( ), next( ), and this is the big challenge: remove( )!

9. ArraySet provides a iterator, which is why we added Iterable<E> to the list of interfaces ArraySet implements. As long as you have an iterator( ) method, this interface is happy and we have established another is-a relationship (hooray polymorphism!).

That's it! Good luck, and have fun!