Faculty of Science and Engineering

COMP125 Fundamentals of Computer Science
Workshop Week 10

Learning outcomes

By the end of this session, you will have learnt about iterators and custom implementation of `ArrayList` class.

Iterators

1. What is the need for having iterators when you can traverse a collection without it? For example, you can traverse an array `arr` as:

   ```
   for(int i=0; i < arr.length; i++)
   ```

   and a `List (ArrayList/ LinkedList) list` as:

   ```
   for(int i=0; i < list.size(); i++)
   ```

2. Consider the following piece of code.

   ```
   ArrayList<Integer> list = new ArrayList(Arrays.asList
       (10,50,20,90,40,80));
   //list = [10, 50, 20, 90, 40, 80]
   ```

   Write a piece of code that adds the items in list and stores it in a variable `total` **using an iterator**.

3. The `remove` method removes the **last item returned** from the iterator. Iterator remains at the same place (that is, before the item after the one removed).

   What is the state of the list after the following code is executed?

```java
ArrayList<Integer> list = new ArrayList(Arrays.asList
    (10,50,20,90,40,80,70,30));
//list = [10, 50, 20, 90, 40, 80, 70, 30]
Iterator<Integer> iter = list.iterator();
while(iter.hasNext()) {
        if(iter.next() >= 50) {
                iter.remove();
        }
}
//list=[???]
```

4. `ListIterator` interface: The `ListIterator` interface addresses the limited set of methods provided by `Iterator` interface and builds on that. In addition to the three methods provided by `Iterator`, it provides the following methods:

   a. `add(<T> obj`: add item at current location of iterator. That is, before the item that will be returned by a subsequent call to `next()`.

   b. `hasPrevious()`: self-explanatory

   c. `previous()`: self-explanatory

   d. `nextIndex()`: index of item that will be returned by a subsequent call to `next()`. In other words, index of item **before** which iterator is located.

   e. `previousIndex()`: index of item that will be returned by a subsequent call to `previous()`. In other words, index of item **after** which iterator is located.

   f. `set(<T> obj)`: replaces the last element returned by next() or previous() with the passed object.

   A `ListIterator` can only be created for a `List` object. There are two ways of creating a `ListIterator`:

```java
//assuming list is either a LinkedList or ArrayList
ListIterator<T> iter1 = list.listIterator(); //cursor before first item
ListIterator<T> iter2 = list.listIterator(idx); //cursor before item at
     index idx
```

What is the state of the list after the following code is executed?

```
ArrayList<Integer> list = new ArrayList(Arrays.asList(10,20,90,40,30));
list.add(10);
list.add(20);
list.add(90);
list.add(40);
list.add(30);
//list = [10, 20, 90, 40, 30]
ListIterator<Integer> iter = list.listIterator();
while(iter.hasNext()) {
        int item = iter.next();
        if(item % 20 == 0) {
                iter.remove();
                iter.add(item - 1);
                iter.add(item + 1);
        }
}
//list=[???]
```

5. A list is `symmetric` if it remains unchanged if reversed.

   Complete the method `isSymmetric` that when passed an `ArrayList<Integer>` object, returns `true` if it is *symmetric* and `false` otherwise.

   Hint: You can create two iterators, one beginning at the front using `list.listIterator()` and the other at the end using `list.listIterator(list.size())`.

Preparation for third practical exam

Writing methods using Java's built-in ArrayList class

1. Define a method that when passed an ArrayList of Integer objects, returns the sum of all items.
2. Define a method that when passed an ArrayList of Integer objects, returns the sum of all even items.
3. Define a method that when passed an ArrayList of Integer objects, returns the sum of all items that are multiples of 5 or is even, but not both.
4. Define a method that when passed an ArrayList of String objects, returns true if there are any items with more than 10 characters, false otherwise.
5. Define a method that when passed an ArrayList of String objects and a second String, returns true if there are any items in the list containing the second String inside them, false otherwise. For example, if list = ["is", "this", "the", "best"] and the String is "his", return `true`. On the other hand, if list = ["is", "this", "his", "best", "work"] and the String is "you", return `false`.
6. Define a method that when passed two ArrayLists of Integer objects, returns true if they are identical (order important). So [10, 50, 20] and [10, 50, 20] are identical while [10, 50, 20] and [10, 20, 50] are not.
7. **(Advanced)** Define a method that when passed two ArrayLists of Integer objects, returns true if they are identical (order unimportant). So [10, 50, 20], [10, 50, 20] are identical and so are [10, 50, 20], [10, 20, 50].

Writing methods for custom-built array-based lists

All questions are based on the following custom-built array-based list class like the one we created in the lecture.

```java
class MyArrayList {
        private int[] data; //items stored here
        private int nItems; //tracker to determine current occupancy

        public MyArrayList() {
                data = new int[10];
                nItems = 0;
        }

        /**
        * @param item: item to be added at the end of the list
        */
        public void add(int item) {
                data[nItems] = item;
                nItems++;
        }
}
```

1. Find the problem with the add method. How many ways are there to fix the problem?
2. You might have already done this as a solution to the previous question. If not, define a method `grow()` that resizes the array holding the items by 5. Please go through lecture notes on how to do this.
3. Define a method `grow(int)` that resizes the array holding the items by the parameter passed. If the value passed is less than 1, nothing should happen. Otherwise, it should increase the size of the array by at most 100.
4. Define a method `add(int, int)` where the first parameter is for the index at which the item (given by the second parameter) should be inserted. Note that all subsequent items need to be moved one space towards the end of the array.

5. Define a method `remove(int)` where the first parameter is for the index whose occupant should be removed from the list. Note that all subsequent items need to be moved one space towards the front of the array.
6. Define a method `remove(int)` where the first occurrence of the item (given by the first parameter) should be removed from the list. Note that all subsequent items need to be moved one space towards the front of the array.
7. **(Advanced)** Define a method `removeAll(int)` where all occurrences of the item (given by the first parameter) should be removed from the list. You can use the previously defined method `remove(int)` to do this.
8. **(Advanced)** Define a method `removeAll(int)` where all occurrences of the item (given by the first parameter) should be removed from the list. You **CANNOT** use the previously defined method `remove(int)` to do this.