Announcements

- Program 4 late turn in date: Tonight at 11:59 PM
  - Late turn in incurs a -5% penalty to the program grade
- Program 5 will be assigned sometime this weekend
- Schedule has been updated
- Zybooks ch14.1 - 14.8 all due 11/18
- Final Exam
  - Reminder this is a group exam, and it is held on a Saturday
  - If you have a conflict see blackboard announcements and follow the instructions for a make up exam

Refresher

How do we loop through the elements in a Collection?

More specifically, looking at the data structures we know about (Linked List, Array List, Stack, and Queue) how do we traverse through all of the data if we are from the perspective of **USING** the data structure?

Refresher (cont.)

The answer is by using an “**Enhanced For Loop**”, also known as a “**For Each Loop**”

Given some ArrayList strings called bookTitles, here is an example of an enhanced for loop:

```
for (String bookTitle : bookTitles) {
        System.out.println(bookTitle);
}
```

Enhanced For Loops (cont.)

Notice the unique for loop syntax:

```
for (String bookTitle : bookTitles) {
    System.out.println(bookTitle);
}
```

The enhanced for loop declares a new variable that will be assigned with each successive element of a container object, such as an array or ArrayList.

Enhanced For Loop (cont.)

An enhanced for loop increases **readability of code** as well as provides a nice level of **convenience** when working with data structures.

If you make your own data structure, does this mean anyone using your data structure can use an enhanced for loop on it?

The answer is NO.

The enhanced for loop is *syntactic sugar* for using something called an: **Iterator**

The enhanced for loop can only be used with those data structures that implement the “ **Iterable**” interface...

Iterator

Iterator is an interface which belongs to Java's collection framework.

It allows us to traverse the collection and access the data element.

The iterator interface contains three methods, we will look at two of them: "**hasNext**", and "**Next**". The third method is called "**remove**" and is rarely used.

Iterator (cont.)

- **hasNext**: It returns true if Iterator has more elements to iterate.
- **next**: It returns the next element in the collection until the hasNext method returns false. This method throws ‘**NoSuchElementException**’ if there is no next element.

Let’s look at an example...

Iterator (cont.)

Java’s Collection implements the **iterable interface** not the **iterator interface** , so what is the difference?

The **Iterable Interface** specifies a method called: “**iterator**” that returns an Iterator object. It is this interface that allows an Enhanced For Loop to be used on a Collection.

The **Iterator Interface** specifies the: “**hasNext**” and “**next**” methods, these methods are responsible for the traversal over the data in a Collection (or data structure).

Let’s look at Java’s Documentation...

Recursion

A **recursive algorithm** is an algorithm that is defined by repeating applications of the same algorithm on smaller problems.

Essentially it solves a problem by breaking that problem into smaller subproblems, solving these subproblems, and combining the solutions.

The **base case** is the point at which the algorithm stops splitting into smaller subproblems.

Recursion (cont.)

A method that calls itself is known as a **recursive method**.

For example:

```
public void countDown(int x) {
        If (x == 0) {
                System.out.println(“ZERO!”);
        } else {
                System.out.println(x);
                countDown(x - 1);                --- Notice it calls itself
        }
}
```

Recursion (cont.)

Let’s follow the algorithm step by step in the previous example...

Recursion (cont.)

Recursion is very useful in certain searching and sorting algorithms.

Here is an example of a searching algorithm:

```
public void searchForFoo(int idx, String[] strArr) {
	If (idx == strArr.length) {
		return;
	} else if (strArr[idx].compareTo(“Foo”) == 0) {
		System.out.println(“Found Foo!!”);
	} else {
		searchForFoo(idx + 1, strArr);
	}
}
```