Announcements

- Tophat Re-Enroll...
- TA office hours schedule is now released
- Program 2 is due 09/20 (If you haven’t started I suggest you start now)
- Midterm 1 Date: 10/02
- After today’s lecture, review Chapter 19 sections 1,3, and 4
    - This is for you knowledge and not assigned as part of your grade

Quiz Time ;)

ArrayList

An **ArrayList** is an ordered list of reference type items. You can think of it as an array, but with more methods associated with it.

- Items in the ArrayList are known as: **Elements**
- An ArrayList will not hold primitive types, so for example you cannot have an ArrayList of booleans
- While you can think of an ArrayList functioning (somewhat) like an Array, you **CANNOT** index into it
    - i.e. in order to access an element the following syntax will NOT work: array[3] = 5;
- ArrayLists will dynamically grow

ArrayList (cont.)

- While an **ArrayList** cannot hold primitive types, we can however use Wrapper classes to accomplish the same functionality
- While we cannot index into an ArrayList to access its' elements, it does provide convenient methods to accomplish the same functionality:
  - add(element) - adds an element to the end of the list
  - get(index) - retrieves the element at the specified index
  - set(index, element) - sets the specified index with the specified element
  - size() - returns the number of element currently in the ArrayList

Let's look at an example...

ArrayList(cont.)

Where to find ArrayList documentation?? Google it of course…

Remember that you need to know your JDK / SE version so you can know which version of the ArrayList to google.

Example: To find information on ArrayList in SE Version 9 , just Google:

Java ArrayList JDK 9

And click on the link the begins with:

https://docs.oracle.com …

Static Fields and Methods

- The keyword **static** indicates a variable is allocated in memory **only once** during a program's execution.
- Static variables reside in the program's static memory region and have a **global scope**
  - Global Scope means the variables can be accessed anywhere in the program
- A **static field** is a field of the class instead of a field of each class object
  - This means a static field can be accessed without creating an instance of a class (i.e. an object)

Let's look at an example...

Static Fields and Methods (cont.)

- Static fields can also be called **class variables**, and non-static fields can be called **instance variables**
- A **static member method** is a class method that is independent of class objects.
    - Typically used to access and mutate private static fields
- Since static methods are independent of class objects, the **implicit parameter** “this” is not passed to a static member method. This means a **static member method can only access a class’s static fields.**

Let’s look at an example...

Memory regions

Your program’s memory is separated into four main regions:

- **Code** (Method Area) - The region where the program instructions are stored
- **Static Memory** (Method Area) - The region where static fields are allocated. The name “Static” comes from these variables not changing (static means not changing); they are allocated only once for the duration of the program, their addresses staying the same.

Memory Regions (cont.)

- **The Stack** - The region where a method's local variables are allocated during a method call. A method called adds local variables to the stack, and a *return* statement removes them.
- **The Heap** - The region where the "new" operator allocates memory for objects.

Let's work through an example...

Garbage Collection

- Programs only have a finite amount of memory available to them.
- Because of this **memory management** is important.
- **Objects** allocated on the heap take up alot of memory, so it is important that we get rid of objects when we are no longer using them
- Conveniently, Java has something called,**Garbage Collection**, that will do memory management for us!

Garbage Collection (cont.)

- How does garbage collection do this? Well the **JVM** (Java Virtual Machine) keeps a count, known as a **reference count**, of all reference variables that are currently referring to an object.
- If a reference count is zero, then the object is considered an **unreachable object**, then it is marked for garbage collection.