Java to study Thursday

* Access Modifiers
  + What are the 4 access modifiers
    - public , protected, default(package-protected), private
    - These go on vars or methods
    - They restrict what other code has direct access to
  + What does each one allow access to
    - Public allows any code anywhere to use the var/method
    - Protected Allows: Our own package and our children in any package
    - Default Allows: Our own package to access
      * (Default is the one that has no keyword)
    - Private Allows: Just our own class ( and technically anonymous subclasses Don’t worry about this)
  + How can we leverage access modifiers for encapsulation
    - We can make our variables private, so no other class can change them. This is a good restriction of access to the data
    - Then we can provide public methods for getting the data or setting the data.
    - By using methods we have the capability to validate data changes and the person changing the data
* Variable Scopes
  + There is NO Global scope
  + Class level variables
    - Not in a method or a static block
    - Static and instance variables are both class level(class and instance scopes)
    - Can get access modifiers
    - Can be potentially accessed by other classes
  + Local Variables
    - Is inside a method or a static block of code
    - Method params (method scoped)
    - Things inside of for or if or while blocks (block scoped)
* StringBuilder StringBuffer
  + What are these two classes
    - These are both ways for making strings that aren’t under the string class
  + How are the related
    - String Builder is the newer and better version of String Buffer, both have nearly all the same methods
    - Buffer is older and slower because it has built in thread safety
    - Oracle recommends you use builder and if you need thread safety, write it yourself.
  + How do they relate to string
    - StringBuffer and builder are both mutable strings in memory
    - Strings are immutable
    - Both are strings of characters
    - Buffer and Builder have more useful methods than String does
  + How do they all 3 compare to each other
    - Builder Buffer - mutable
    - String - immutable
    - Builder Buffer - better methods
    - Builder - faster better than buffer
    - Buffer - built in thread safety
* Equals Hashcode
  + How are these two related to each other
    - Hashcode is supposed to take an object and generate from it a unique value that represents that object
    - Equals is supposed to compare two objects and determine if they are the same ( we decide what makes two objects the same)
    - It is very important to have hashcode return the same value for two objects that .equal each other.
  + What do we have to do with these methods on our objects
    - It means we should override these two methods in tandem from the object superclass.
    - We should use all the same fields in equals and in hashcode for the objects two methods.
* Reflections
  + What is the reflections api
    - It allows us to get access to methods classes and fields during runtime
    - These method, classes, fields are their own special objects and the have methods on them to allow us to manipulate code that we wrote while the program is running
    - Ie: I can find a private field and using reflections make that field public
    - Extraordinarily useful for the people who make tools.
    - You might end up using it for testing
  + Wait what? You can do that?
  + Dang that’s really cool!
* Maven
  + What is maven?
    - Its like npm but for java
    - It is a dependency manager
    - It is a project manager
    - It also handle build lifecycle
  + How can we use it to manage dependencies?
    - POM.xml ( like the package,json)
      * Project Object Model
      * IT is xml so it looks a lot like html
      * It has some special tags that we use to get dependencies
      * <dependencies>
        + <dependency> Specify what code for maven to get</>
      * Maven has a massive repository of code for its dependencies
      * It downloads those dependencies to a folder on your computer called .m2 (local repository)
  + What is the maven build lifecycle
    - Validate - all the files are here, make pom is correct
    - Compile - we run all the classes through javac and make sure they all compile
    - Test - run all tests using framework specified in pom and make sure they pass (unit test)
    - Package - we build a .jar( java archive file) that contains all necessary code for JRE to run the program
    - Verify - testing round two, integration testing with the larger system as a whole
    - Install - send the package into a local repository so other projects can use it as a dependency
    - Deploy - ssend package to a remote repository for other projects to use as a dependency
* Junit
  + What is Junit?
    - De facto standard for unit testing in java
    - A lot like jest
  + Of the 4 pieces of a unit testing framework which does junit supply
    - Test Runner - Junit does this
    - Assertation Lib - Junit does this too (with help from the Hamcrest Matchers)
    - Mocking - Junit DOES NOT do this
    - Coverage reports - Junit DOES NOT do that either
  + What is mockito?
    - This is a tool used to mock dependencies
    - Mock objects and we can mock method calls
  + What is JaCoCo
    - Java Code Coverage
    - It can take Junit tests and build code coverage reports to give us info on what we are actually testing.
  + Syntax
    - Junit functions through annotations (@override)
    - @test, @before @after @beforeClass @AfterClass
    - Mockito @injectMocks
    - AssertThat(this.value is(not(null())) ) (Hamcrest matchers)
* Threads
  + Java is multi Threaded
    - This allows for parallel processing which is very powerful
    - It introduces a whole new host of problems
    - Concurrency problems
      * HAving unprotected data
        + Two threads that are both using the same array of users
        + If they can both go at the same time, my users array could potentially be changed to something erratic.
        + We introduce a lock onto the resource
        + If you want to use the array you get the lock object
        + If the lock is taken you sit waiting for the lock to be released
        + Java provides synchronized keyword (goes on methods)
      * Philosophers Dining problem
        + Imagine a circle of philosophers waiting to eat dinner
        + Each has a single chopstick by their right and left hands
        + A Philosopher needs both chopsticks to eat
        + What happens if every philosopher picks up the chopstick on their right at the same time?
        + We enter a state called deadlock - where threads can’t execute code even given time by the jvm
        + Solution to this one, is try to get the first lock, then try to get the second lock … if you fail to get every lock you need, let go of all of the locks you have already acquired, try again when something changes
      * Producer Consumer
        + Go research yourself
    - Looking at the stack and the heap, when I make a new thread I am making a new stack that shares the same heap
    - The JVM has complete control over which thread is executing at which time, Thread Scheduling.
  + How can I make a thread?
    - Extend the thread class and override the run method
    - Make a new object that implements the runnable interface
    - (nowadays) make a new callable object and give to a thread pool
  + What are the recommend ways for making a thread
    - Oracle recommends using runnable
    - 1 you get an additional type from using the interface
    - 2 it allows you to build your own hierarchies for your threads for sharing behaviour and data
  + What are the different thread states
    - NEW - we built the object , haven’t called the run method
    - RUNNABLE - this means the thread is currently executing
    - WAITING - For some other thread to take a specific action, most of the time this means waiting for the thread to die
    - BLOCKED - waiting to get a lock for a resource we need to execute
    - TIMED\_WAITING - just like waiting but its only for a specific period
    - TERMINATED - no more code to execute, waiting to be garbage collected
  + How can I wait for a thread to finish
    - join() - will allow us to wait for a particular thread to reach the terminated state, we will be in waiting state until then
  + How do threads work in the stack and heap
    - One Heap per program
    - One Stack per thread
  + Can I force a thread to run?
    - NO.
    - We can suggest that a thread does not need to run by using the method yield()
    - But the jvm gets the final decision on what runs when
  + Interfaces - types
    - Marker
      * They have zero methods in them
      * They are only used to give additional typing to an object
      * Clonable, Serializable
    - Functional (java 1.8)
      * Has exactly one abstract method
      * Because as of java 8 we can write a lambda function where you would need a functional interface object
      * (param, param) -> {
      * }
      * Comparator, Consumer, Runnable, (Washable)
  + Java Bean (like our js models)
    - 1 you need a default constructor (you can have more)
    - 2 you need all private fields and public getters and setters
    - 3 implement serializable (no one really does this anymore)
  + Serialization
    - This is the process of turning a java object into byte code for a different java program to rebuild into the same java object
    - You can think of this process as like turning a js object into json for a different js program to rebuild
    - Nowadays, we mostly just turn java objects to json and back using a library like Jackson
  + Java Byte Code
    - What you get from the javac compiler, this is not machine ready code
    - The JVM interprets java bytecode and turns it to machine level code
* Extends demo as well as this