Java to Study Wednesday

* Exceptions and errors
  + What are exceptions
    - Is some problem that could go wrong, that your program could reasonably recover from
    - ArrayIndexOutOfBoundsException
    - IOException - if you have a problem getting input or output from a file
    - FileNotFoundExcpetion
  + What kinds exist
    - Runtime - unchecked
      * Are generally attributed to Programmer error
      * Things like ArrayIndexOutOfBOunds, ArithmaticException, NullReferenceException
      * Basically if you try to do something that the program can’t handle you get a runtime exception
      * Solution? Write better code
    - Compile-Time(I hate the phrase compile time) Checked Exceptions
      * THESE ARE THROWN DURING RUNTIME
      * THERE IS NEVER AN EXCEPTION OR AN ERROR THAT HAPPENS DURING COMPILATION
      * YOU CAN GET COMPILER WARNINGS OR ERRORS BUT THOSE ARE NOT THE SAME
      * These exceptions you are required to have a try catch or to “throws” them to the method that called you
      * Public void randomMethod() throws FileNotFoundExcpetion{
      * You shouldn’t have the main method throws errors, because there is nothing above the main method to catch those errors/exceptions
  + Try Catch Finally
    - Almost exactly the same as JS
    - In our catch statements we put the type of the error
    - This way we can have different catches for different errors
    - Make sure to put more general exceptions later or the might prevent your catch statements for specific exceptions
    - Ie Catch(IOExcpetion e) before catch (Exception e)
  + What are errors
    - IS something that has gone wrong, that we Shouldn’t recover from.
    - Is error has some effect on data that it shouldn’t potentially corrupting it
    - StackOverflow, OutOfMemoryError, VirtualMachineError
  + What are the difference
    - We can save ourselves from exceptions, but errors we should let crash the program
* Abstract Method?
  + Method signature, that has no implementation
* Interfaces
  + What can go in an interface
    - We can put public abstract methods in an interface
    - In Java 1.8 we also added public static final variables (these are also known as constant values)
    - In java 1.8 we are allowed to use the default keyword with an abstract method to write a default implementation
  + How to use them
    - To use an interface you need to implement on a class
    - The class then needs to provide implementations for every method on that interface
    - If that interface has default methods, the class can choose to override them or not
    - Class can implement any number of interfaces
  + Why to use them?
    - They are important because they allow us to achieve abstraction between different parts of our program.
    - Imagine you have a dao for postgres, and you decide to swap to using mongo
    - Your service layer needs to get some version of a dao
    - By giving that service layer an interface that defines those methods and by having my daos implement those interfaces, I can guarantee that my service layer will continue to function even if I change the implementation
    - By defining our dependencies on their behaviour, we can change implementation so long as the behaviour stays the same
* Abstract Classes
  + Is like a normal class with a few exceptions
  + 1 it can contain abstract methods, by putting the word abstract on the class
  + 2 It cannot be instantiated
  + We can have concrete methods ( is a method with an implementation)
  + We can have any number of variables of any types
  + We can have constructors
  + When we use it, Imagine express endpoint. That endpoint has all sorts of functions and variables for the req and res and managing them. But you need to write the method about how to process the req. So they make an abstract class with a bunch of finished and leave on abstract method for you to override
  + ExpressEndpoint class GetAllUsers extends ExpressEndpoint
* Wrapper classes
  + What are they
    - The are special object classes that correspond with the primitive types
    - Integer Double Short Character Byte Boolean Float Long
  + Boxing unboxing
    - So if I ever use an int where I needed to put an Integer, java will automatically convert my int to an Integer AutoBoxing
    - Vice versa is called AutoUnboxing
  + Purpose?
    - 1 we can put helpful methods and values on the wrapper class. Things like converting a string to an int or an int to string
    - 2 They are some times when we can’t use a primitive for a data type. Most commonly is when I am using generics
* Generics
  + How to use them
    - We can specify the use of Generics with <T > and some capital letter most of the time. T. K ,V
    - Then that Letter acts as a placeholder for a type that will be determined when the object is instantiated
    - Class OurSpecialArray<T> {
    - Only work with Object Types (look at wrapper classes)
  + Where are they used in java code
    - They are used very extensively in collections, You also see them alot in classes with static methods for manipulating data like the Arrays class.
  + Whats their purpose
    - They allow us to write reusable code, but still keep strict typing in that code.
    - Imagine making an array that could take in any kind of object. But we want it to have all of the same kind of object.
* Collections
  + Hiearchy of interfaces
    - Iterable (I) says it must have a function that builds an iterator ( an object that has a next function and can move through a grouping of data)
    - From iterable we have Collection (i) has many common methods for all of the different data structures, Size(), add, contains, remove
    - From collection we have 3 interfaces, List, Set, Queue. These are the three main different types of data strucutres
    - All Collections can only store Object types
    - List (i)
      * It has an index List.get(4). The index is insert order
      * Dynamically sizing (like a js array)
      * It allows for duplicate values
      * Allows null as a value
    - Set(I)
      * Single value elements
      * Have no guaranteed order, unlike list
      * Dynamically sizing
      * Do not allow for duplicate values
      * One null value is allowed
      * Have no index, to find a specific value you can use contains if you know exactly what the object looks like, or you can iterate over every element in the set
    - Queue(i)
      * FIFO data structure
      * It works like waiting in a line
      * Different methods like Poll (retreive first in queue) or peek(look at front of queue)
      * It does have order like a list but we generally cant access by index
      * Dynamically sizing
      * Allows for duplicates
    - Child interface of Queue Deque(i) ( double ended Queue)
      * This is just like a queue
      * Except we can add to the front or back and retrieve from the front or back
      * This means we can use the Deque as a stack or a queue if we want
      * Use case: if you want a stack, make a deque
    - Map(i) DOES NOT EXTEND COLLECTION INTERFACE
      * Is a dictionary data structure
      * It has Key value pairs
      * Keys in a map follow the rules of a set
      * Values are allowed to be duplicated if they have different keys
  + Know the implementations
    - ArrayList
      * Imagine a ts array
      * Behind the scenes uses an array for storage
    - LinkedList
      * Will bahave just like an ArrayList
      * Behind the scenes it has the node objects that point to the next node object in the list
      * {1 V:’A”, next:2} -> {2, :v”B” next: null}
    - Vector and Stack
      * These are old datastructures that predate the Collections Api and got grandfathered in.
      * Oracle highly recommends you never use these data structures unless your code is already using them.
      * They are slower because they have thread safety built in and they don’t follow the modern design principles of java.
    - Priority Queue
      * Its a queue but elements also have a certain priority
      * When we add an element we choose it’s priority and it gets to skip the line up to that priority
    - Array Deque
      * This implements deque and behind the scenes it uses a circular array ( look it up very cool)
    - HashSet
      * This is the default implementation of a set
      * No other differences then the set rules
      * Behind the scenes it uses a hashmap to store the set data
    - TreeSet - SortedSet(i)
      * This is a set but it has a strict ordering
      * That ordering is based on a implementing the comparable interface on the elements
* 4 pillar Examples
  + Abstraction - Interfaces and abstract classes those allow us to work with behaviours instead of implementation
  + PolyMoprhism - Method overriding and overloading upcasting and downcasting
    - Upcasting is my ability to treat a child as the type of its parent I can call A Cat object an animal
  + Encapsulation - restricting access to the data in our classes with the private and public keywords and with getter and setter methods
  + Inheritance - Classes can extends other classes to share behaviour and data, Object super class