Java Core Review

* JDK vs JRE vs JVM?
  + JDK contains the JRE and JVM
  + JRE contains the JVM
  + JDK Java Development kit
    - Needed by developers to make Java applications
    - What does the JDK contain that the JRE does not have?
      * Compiler
      * Allows you to turn something.java source code files into something.class Java byte code files
  + JRE Java Runtime Environment
    - Contains any libraries and resources necessary for the Java Application to run
  + JVM Java Virtual Machine
    - Is your virtualized registers and processors for executing your byte code
    - JIT compiler
    - JVMs are OS dependent
* Tell me about Java 8
  + Java 8 is a version of Java that came out it 2014
  + It is the most popular version of Java code today
  + Features
    - Lambda expressions to perform functional programming
    - Default keyword for interfaces allowing a default implementation in your interfaces
    - Streams API a way to functionally handle I/O operations
      * Using lambdas to say what to do with the data as you receive it
* What are the pillars of OOP?
  + Abstraction – hiding implementation
    - Coding to interfaces
    - Not worrying about the implementing class
  + Polymorphism – One object many labels
    - Refencing an object via a parent class or interface that it implements
    - Overriding (aka runtime polymorphism)
      * Same method signature different implementation ins a child class
      * Covariant returns
        + An overridden child method returns an object of a more specific type so it still satisfies the parent signature
    - Overloading (static or compile time polymorphism)
      * Same method name different parameters in the same class
  + Inheritance – reusing the code from another class
    - Inherit its fields and methods
    - Creates an is a relationship
    - Static fields and methods ARE NOT INHERITED
      * Static means specific to that class/ attached to the class itself
  + Encapsulation – Protecting your variables and methods from other parts of your code
    - Public – accessible anywhere
    - Protected – the package + any classes that inherit
    - (default) – the package
    - Private – just within the class
    - Allows you to hide implementations details so that you only interface with the public facing methods that you are supposed to
* Ways to use the final keyword
  + Use on a class to prevent the class from being inherited
    - Examples include String
  + Use on a variable to prevent the value of that variable from changing
  + Use on a method to prevent the method from being overriden

Design Patterns

Why use design patterns?

* Design patterns are a solution to a more general recurring problem in software application
  + I have a connection resource that people keep duplicating in the program. Causing the DB to eventually timeout from too many connections
  + Multiple scanners in my applications are diverting the Input Stream away from the keyboard making the interface unreliable
    - Both of these problems are solved by only have one of the class
    - A singleton design pattern is the solution
  + I know what an object needs to be able to do. But I do not know what object is best for me to use
    - Factory design pattern allows me to say what I need and it returns the object to use
* Factory
  + You have a factory method that generates objects that match a certain parent class/ interface
  + You feed the factory method what you need and it churns out an implementation class that meets your specificiations
    - Examples
      * ArrayList vs LinkedList
      * SessionFactory creates sessions using your details like url , username, password, type of database
* Front Controller
  + All request to an application go to a SPOC Single Point of Contact
  + From that SPOC they are sent to various parts of your application
    - Ex Spring uses a DispatcherServlet where all requests are sent to
    - The DispatcherServlet then uses the RequestMapper to send the request to an appropriate controller
* Singleton
  + Object that you only want one instance of in the entire application
    - Ex singletons
      * Scanner, Factories, Connection, DAOs, controllers, services
  + Creating a singleton manually
    - Private constructor
    - Public getter
    - Some objects are very large like SessionFactory and having multiples would clog up memory and reduce performance
    - Sometimes you want to enforce a shared state throughout the application
      * Ex. You are building a java desktop application to play chess between two users on the computer
      * You have an object gameboard that should have the same states for each player
* DAO (Data Access Object)
  + This is a class that is designed to perform all of your persistence operations
  + Creates a separation of features/layers
  + Decouple your code
* DAO vs Repository (Both ways of creating the persistence layer)
  + DAO is supposed to create a 1 -1 connection from an object to record
    - When you update an object in Java a record in the Database will be updated
      * If you change only name in your object only name in the database is changed
  + Repository operates like a map
    - Updating an object is replacing a record in the database
      * Changing even one thing in your object causes a complete overwrite of the record

Java Collections Framework vs Collection Interface vs Collections class

* JCF
  + A conglomeration of interfaces and classes
  + They are all data structures
  + Classes designed to hold objects in different ways
* Key interfaces
  + Iterable – top interface of the JCF. Not implemented by map
    - Maps are not Iterable
  + Collection – Interface implemented by all collections except for map
  + List
    - Can have duplicates
    - Ordered by insertion
    - Implementations
      * ArrayList – all elements are stored together in a sequential block of memory
        + Fast retrieval anywhere
        + SLOW insertion or deletion at the beginning of an arraylist
      * LinkedList – Elements are stored throughout memory with pointers linking you to the neighboring elements
        + SLOW retrieval towards the middle of the LinkedList
        + Fast insertion towards the ends of a LinkedList
  + Set
    - Unique elements only
    - Not ordered by insertion
      * Implementations
        + HashSet

“orders” you elements via their hash number

* + - * + TreeSet

Orders your elements via the natural ordering

Comparable defines the natural ordering

* + Queue
    - FIFO
  + Stack
    - LIFO

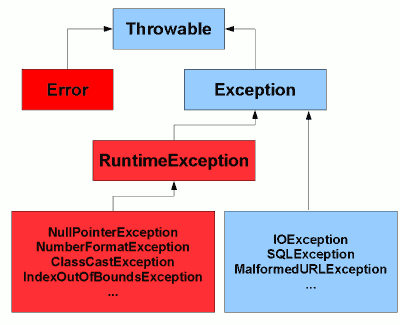
Comparable vs Comparator

* Comparable is an interface that classes can implement that defines the natural ordering for a class
  + Natural ordering as the default ordering for this class when in a sorted collection
    - You get one comparable you each class can only implement an interface once
* Comparator is a functional interface
  + Performs sorting for a class as well
  + You can write as many comparators lambdas as you want to help sort your class
    - Ex if you have a class Basketball player
      * In one TreeSet you want them ordered by height
      * In the other TreeSet you want them ordered by salary

Array vs ArrayList?

* Array is a sequential block of memory that can hold elements. Primitives or objects
  + They are created using the syntax int [] ray = new int [10]
    - Array of size 10
  + Fixed size when created
* ArrayList is a class in the JCF
  + It uses array under the hood to store its elements
  + However because it is a class is has a lot of helpful methods that a plain array in Java would not have
    - Find()
    - indexOf()
  + ArrayLists can dynamically resize themselves so you do not have to set a size for the ArrayList when you create it.
  + It will create more memory as needed

Exceptions



* Exceptions are for whenever something does not execute along the happy path of execution
  + Intended execution where nothing goes wrong
* Checked Exceptions / Compile time exceptions
  + Not because they happen at compile time
  + But because your code will not compile unless you have code to handle the exception
* Runtime Exceptions
  + You are not required to handle before your code will compile successfully
* Errors
  + Catastrophic failures which you should not be trying to handle

When to use checked vs Unchecked exceptions?

* Create checked exceptions whenever you want to enforce that there is code to handle if something goes wrong
  + For example
    - Make UserNotFound exception checked to force the application to have back ups incase a user is not found
* Unchecked exceptions can be used when it is okay to let the exception go unhandled
  + Ex UnableToParseFullNameExcpetion
    - Given “Adam\_Ranieri”
    - If left unhandled the database just puts in the full name to both firstname and lastname columns

LTI Questions

What would you show a user if there unexpected problem in the backend

“You generally want to hide http response status codes if something goes wrong. Your average user is probably not tech savy enough to understand a 503 and displaying it to the page with just black text on a white background looks unprofessional and scary.”

SQL triggers

* You can set triggers or traps on a table in a database
* When something activates that trigger a piece of SQL is called
  + Trigger can be configured to be activated whenever someone inserts to a table
  + Someone deletes from a table or tries to modify
  + You could write a trigger that whenever someone calls delete on a record it automatically save a copy of that record to a backup table

Build Triggers

* Different ways of telling Jenkins to start a build
* Github webhooks so that whenever a change in a branch happens that a build is started
* Timed polling – a new build is started every hour

Storing images in a database

* Old-school – Store the image as a BLOB in the DB. Binary Large OBject
* Newer-cooler option -save the image to an s3 bucket. In the database save a hyperlink to that image in the s3 bucket

Encrypt Data is a properties file

* I don’t know how to encrypt data in a properties file
* You can have your properties file reference environment variables so you avoid putting sensitive information into your properties file

Agile Coding standards

* Having your code changes be peer reviewed before being pulled into a branch
* Having automated tests verify the functionality of your code when you push it
* Having a static code analysis tool like SonarCloud read the code for bad practices
* (Python) recommends using against comments unless necessary
  + Focus on making the code descriptive versus a lot of comments

Index for a database

* In very large databases your tables will have indexes to help sort your queries faster
  + Imagine you had a deck of cards and you were tasked with finding the 9 of diamonds
  + The suit heart, clubs, spades, diamonds could be made an index
  + Searching for 9 of diamonds will make the query only look at cards in the diamond index increasing query efficiency