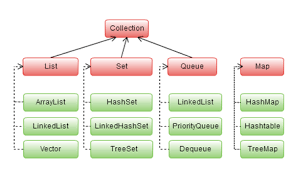
Java Review

* High level features of Java
  + Strongly and statically typed language
  + Compiled language
    - Write a .java source code (human readable) and execute a .class file which Java Byte code (machine readable)
  + WORA (Write once Run Anywhere)
    - Any Java application will run the same on any machine that has a JVM.
  + JDK vs JRE vs JVM
    - JDK Java Development Kit
    - JRE Java Runtime Environment
    - JVM Java Virtual Machine
  + Java 99% OOP language
    - Primitives
    - Lambdas
  + Extensive libraries and enterprise software
  + Designed to be very scalable
  + Automatic memory management
    - Garbage collection
  + Java is 100% pass by value
* Pillars of OOP (APIE)
  + Abstraction
    - Hiding the implementation details
    - You do not need to know HOW something works just how to use it.
      * Interfaces and Abstract classes (Code to an interface but do not need to know the implementation class)
  + Polymorphism
    - One object many labels
      * A reference variable can refer to an object by any legitimate parent class or interface
    - Runtime Polymorphism
      * Overriding
        + A child class overrides/changes the implementation of an inherited method.
        + You must have the SAME method signature but different implementation.
    - Static Polymorphism
      * Overloading
        + Same method name but different parameters.
        + Done in the same class.
  + Inheritance
    - Ability of classes to inherit the fields and methods (properties) of anther class.
    - Allows us to avoid repeating the same code.
    - Create logical hierarchies in our application.
      * Java uses Class based inheritance.
      * Only inherit from parent class in Java.
  + Encapsulation
    - Protecting our methods and fields from other parts of our code
    - Done with Access Modifiers
    - MODIFIERS ARE NOT SCOPES
      * Public
        + Accessible anywhere
      * Protected
        + The package + inherited child classes
      * Default
        + The package
      * Private
        + The class
* Abstract classes vs interfaces
  + Abstract class
    - Cannot directly instantiate
    - You must always build a child class
    - CAN have concrete methods (methods with an implementation)
    - CAN have abstract methods (Methods with no implementation)
    - Have regular fields like any class
    - Only inherit/extend a single abstract class
  + Interface
    - NEVER instantiate an interface
    - Only abstract methods
      * \* there is a default keyword.
    - All variables are public static and final.
    - Implement as many interfaces as you want.
    - Functional Interface
      * Interface with a single method signature used to create lambdas.
    - Marker Interface
      * Is empty and used solely to give a Type to a class
* Constructors
  + Special methods that create an instance of a class
  + Must be named the same as the class name.
  + Overload constructors as much as you want.
  + Only methods in Java to not have a return type.
  + Every constructor starts with a call to super which creates the parent object
* Object Class
  + Grandparent class that every other class in Java inherits from
  + Everything is of type Object
  + Important methods
    - Equals()
      * Returns Boolean if this object is “equal” to whatever you compare it to
    - toString()
      * Returns a string representation of the object
        + Default is to print the memory address.
* Wrapper Classes
  + Object versions of primitives
  + They provide the utility of an object with methods
  + Are a bit slower than primitives
  + Autoboxing
    - Primitive => Wrapper
  + Unboxing
    - Wrapper => primitive
* Exception Handling
  + An Exception is an object that holds information about a failure.
  + Exceptions are throwable.
  + Keywords
    - Try
      * Please run this block of code
    - Catch
      * If an exception is thrown please catch and deal with it here.
    - Finally
      * Always executes regardless of if the code executes correctly or not.
    - Throw
      * Sends flying and exception.
    - Throws
      * Warning on a method signature that a certain exception can be generated within this method.
  + Checked Exceptions / Compile Time Exceptions
    - Exceptions which MUST be handled in your code in order for your code to compile.
    - Design choice to FORCE developers to handle failures.
  + Unchecked Exceptions / Runtime Exceptions
    - Exceptions that you are not required to handle.
    - But you should.
  + Error
    - A catastrophic failure that you should not try to recover from
    - If you get an error it means you need to rewrite your code
* Collections Framework
  + The CF is a series of interfaces and classes whose main purpose is to hold objects
  + 
  + Iterable is the top of the collection framework
  + Key interfaces
    - List
      * Maintains the order of insertion.
      * Allows duplicates.
    - Set
      * Does not maintain the order of insertion.
      * Does not allow duplicates.
    - Queue
      * FIFO
      * First In First Out
    - Map
      * Does not implement Collection or Iterable Interface
      * Stores objects as key value pairs
* Generics
  + <> Diamond brackets/ Angular Brackets
  + Allow us to set a type for certain interfaces or classes
  + Used heavily in collections to restrict what type of object is allowed to be stored in a particular collection
  + List<String> strings =new ArrayList<String>();
  + List things = new ArrayList() // valid but you can literally store any object in it. Defeats the purpose of a strongly and statically typed language.
* Scopes of a variable
  + THERE IS NO GLOBAL SCOPE
  + Class / static scope
    - Attached to the class itself.
    - Only one copy.
  + Instance/ object scope
    - Attached to the object.
    - One copy per object.
  + Method scope
    - Variables passed in as the parameters to method
  + Local/block scope
    - Variables defined within a method
* Functional programming
  + A different paradigm of programming than OOP
    - Paradigm is a way of think about how to code
  + OOP
    - Classes and objects and methods that model the real world.
  + Lambdas
    - Store a defined piece of code in a variable.
    - Pass this lambda variable to other methods and parts of your application.
    - Lambda passed as an argument is called a callback function.
  + Creating a lambda in Java
    - Functional Interface
    - MyLambda func = (arguments) -> { code to execute};
  + Examples of lambdas
    - Comparator
    - Collections.sort(list,function)
    - Runnable
* String
  + STRING IS AN OBJECT NOT A PRIMITIVE
  + Strings use a string pool
    - Strings of the same value are the same object in memory
    - Saves on memory
    - Strings are immutable and cannot be changed once created
* StringBuilder
  + Mutable version of string
  + Directly modify its value
  + Much more efficient for any character manipulations
* StringBuffer
  + Thread-safe version of string builder.
  + Only allows one thread to work on it at any given time.
* Design patterns
  + Java Beans
    - Any class that has private fields
    - Public getters and setters
    - A no args constructor
    - (Technically implements serializable)
  + Factory
    - A method that returns an implementation of an interface
    - Allows a developer to pass in arguments for what they want and the method will choice the best option. The developer does not need to know it.
      * ArrayList vs LinkedList example
  + Dependency Injection
    - Create an object by passing in the other objects (dependencies) it needs to work.
    - BookService bserv = new BookService(new BookDaoLocal());
* JUnit
  + The most popular testing framework in Java
  + We are using JUnit 5 (latest version)
  + Annotations
    - @Test
    - @TestMethodOrder
    - @Order(1)
    - A lot more we will get to in time
* TDD
  + Test Driven Development
    - If you do not do TDD you are doing BUG driven development
  + First you design the interface.
  + Second you design the tests for that interface.
    - Makes you think about the interface methods a bit harder.
  + Third write of the implementation of that interface.
  + Pros
    - Spot bugs early.
    - Quickly find where a bug is.
    - Build upon reliable code.
* Threads
  + Thread is a path of execution through a program.
  + Java is a multi-threaded language.
    - Run multiple threads in parallel/simultaneously.
      * Can optimize performance.
    - Running multiple threads can lead to a lot consistency bugs.
    - Significantly hard to write good reliable multi-threaded code.
  + Make thread
  + Create a runnable lambda
  + Pass it into the constructor of a Thread Class
    - To start the thread
    - Thread.start();
* Gradle
  + Build tool (It builds software applicates)
    - Platform agnostic technically
    - Could be used for Java or C or Ruby
      * Java and Java based languages mostly
  + The main file for any project built with gradle is build.gradle
  + A script that build the project
    - Set the name
    - Builds in the external libraries (dependencies) needed for the application
      * Pom.xml in Maven
      * Package.json in node
  + Gradle is written in Groovy