Java Review

Java

* WORA write once run anywhere
  + Cross platform
  + JVM
* Strongly and statically typed language
* Compiled language
  + something.java => something.class
* Automatic memory management
  + No pointers
  + Garbage collection
* OOP centric 99% (1% lambdas and primitives)
* Multithreaded

OOP (Object Oriented Programming)

* Abstraction
  + Hiding the details. Covering up the implementation
  + What it does as opposed to how it is done
  + Interfaces
  + Abstract classes
* Polymorphism
  + Many labels for the same object
  + Overloading
    - Same method name different parameters
  + Overriding
    - Same method signature in a child class. Different implementation.
* Inheritance
  + Ability of classes to derive code from other classes
  + DRY (do not repeat yourself)
* Encapsulation
  + Protecting our variables and methods from other parts of our code
  + Access modifiers

OOP constructs

* Class
  + Blue print for an object
* Object
  + Instance of a class
  + Can have state
    - Fields
  + Can have behavior
    - Methods

Scopes in Java

* Static/class
  + Attached to the class
* Instance/object
  + Attached to an object
* Method
  + Only within this method
* Local
  + Only within this set of brackets
* NO GLOBAL SCOPE

Access Modifiers

* Public
  + Accessible anywhere
* Protected
  + Accessible in the package + inheriting subclasses in other packages
* (default)
  + Only within this package
* Private
  + Only within this class
* ACCESS MODIFIERS ARE NOT SCOPES

Constructors

* A special type of method
* They create an instance of a class
* They must be named the same as the class they are in
* They have no return type
* Can be overloaded
* CANNOT be overridden
* If you do not write a constructor Java will create a default no arguments constructor for you to use
* The first line of every constructor is to call super()
  + This will create the parent object

Memory Management

* Automatic memory management
* You as the programmer do not have to worry about deallocating memory
* Garbage collection
  + Objects that are unreachable in memory \*can\* get deleted
  + The object has not reference to it
  + This cannot be enforced just requested by system.gc()
* Finalize method
  + Called when an object is garbage collected
  + Do not use

Strings

* ARE OBJECTS
* They store character information
* They are immutable (cannot be changed once created)
* Utilize a string pool
  + Special location in memory where String are stored
  + Strings with the same value will be the same object
* StringBuilder
  + Mutable version of String
* StringBuffer
  + Thread safe version of StringBuilder

Primitives

* boolean
* byte
* char
* short
* int
* float
* long
* double

Wrapper classes

* Object versions of primitives
* Wrappers have many convenient methods related to their primitives
* Autoboxing
  + Primitive is turned into a wrapper
* Unboxing
  + A wrapper is turned into a primitive

Object class

* Top class in Java
* Every object inherits from Object
* Everything is of type Object
* Methods to know
  + toString()
    - returns a string representation of the object
    - If not overridden returns the memory address of the object
  + equals()
    - method that the developer should override
    - returns true or false if the developers deems this object ‘equal’ to another object
  + hashCode()
    - returns a unique value regarding the state of an object

Abstract Classes vs Interfaces

* Abstract Class
  + Abstract classes cannot be instantiated
  + No restrictions on your variables



* + Can have abstract methods



* + You can only inherit from one abstract class
  + Creates an is -a relationship
  + Can have constructor
* Interfaces
  + All variables are public static final
  + Cannot construct an interface
  + Methods must be abstract\*
    - Technically default keyword
  + Creates is-a relationship
    - Grants the “ability to”
    - Most interfaces end in able
  + You can implement as many interfaces as you want
  + Types of interfaces
    - Marker Interfaces
      * No methods
      * Used solely to grant a type
    - Functional Interfaces
      * Only one method
      * Used for lambdas

Collections Framework

* A conglomeration of classes and interfaces that are designed for storing and sorting objects
* Key interfaces
  + Iterable
    - Top of the collections framework
  + Collection
  + Queue
    - Stores objects as FIFO
  + List
    - Stores objects via the order of insertion
    - Allows duplicates
      * ArrayList
      * LinkedList
  + Set
    - Does not allow duplicates
    - Does not maintain the order of insertion
      * TreeSet
      * HashSet
  + Map
    - Stores objects as key value pairs
    - Does not implement Iterable
    - Cannot be iterated through
* Comparable
  + Interface used to sort objects in a collection
  + Defines what is called the “natural ordering”
* Generics
  + <> diamond operator
  + Allows to specify what type belongs in this interface/class
  + Specify what type belongs in the collection

Exceptions

* Exceptions are objects
* They are thrown when a program does not execute “happily”
  + Checked/compiled exceptions
    - Required to handle these for your code to compile
  + RuntimeExceptions
    - Not required to handle for your code to compile
    - IndexOutOfBounds, InvalidInput, TypeCast
* Errors
  + Catastrophic failures
  + Should not catch them even though you can
    - OutOfMemory, StackOverflow
* Throwable is the top of the exceptions hierarchy

Design Patterns

* Factory
  + Have a method return an implementation of an interface
    - ArrayFactory
  + Avoid coding directly to classes
  + Provides a level of abstraction
* DAO
  + Data Access Object
  + Class responsible for performing CRUD operations
* Java Bean
  + A class
    - No arguments constructor
    - Private fields
    - Public getters and setters

Lambdas

* Way to achieve functional programming
* Storing code as an object
* We can pass around this code object (first class function)
* Steps to create a lambda
  + Create a functional interface
  + Write an implenetation of that interface using lambda syntax
    - (a,b)->{ code};
* Lambda passed as a parameter to another method/function is called a callback

Threads

* A thread is a path of execution through a program
* You can create as many threads as you want
* Multi-threading can increase efficiency
* However, you can run into concurrency issues
* Steps to create a thread
  + Write a Runnable lambda
  + Pass that lambda into the constructor of a thread
  + Call thread.start()