**## Java Study Guide (Week 1)**

You should be able to explain and apply the following topics:

**### Fundamentals**

-

|  |  |
| --- | --- |
| Features, benefits, and drawbacks of the Java language | |
| - WORA (write once, run anywhere) | It doesn’t matter the type of machine, as long as you have a JVM your machine can read the Java code. |
| - Backed by Oracle |  |
| - Rich APIs (e.g. Collections API) |  |
| - Object-oriented |  |
| - Strongly/strictly typed |  |
| - High level (e.g. memory handled via automatic garbage collection) |  |
| - POJO vs Bean - | **POJO - Plain Old Java Object** - An ordinary Java object, not bound by any special restrictions other than those enforced naturally by Java. They are used because of their readability and reusability. .  **Bean** - A special kind of POJO - All Java Beans are POJOs, but all POJOs are not beans. Used to represent state and data. Beans should implement a Serializable interface, All fields should be private to provide control over the fields. Fields should have getters and/or setters. Should have a no-arg constructor. |
| - Stack vs Heap | **Stack** is the temporary memory. Holds local info. Reference Variables - points to a place in the heap.  **Heap** is the larger storage. Holds most of the info of the program. |
| - Purpose and contents of the JDK, JRE, and JVM | **JDK - Java Development Kit** - Used by Java Devlopers and provides the environment and the Development tools to develop java programs. Catches errors. Compiles Java to bytecode.  **JRE - Java Runtime Environment** - Is the installation package to run the java program or applications on your machine.  **JVM - Java Virtual Machine** - compiles the bytecode (the .class file) into machine code. (JIT, Just in Time compilation). Interprets the file into machine code. |
| - Garbage collection (generally, what does the garbage collector do?) | Destroys objects not in use. Cleans up the heap memory. Always running in the background. |
| - Main method signature and significance | **Syntax**: public static void main(String[] args){ }  The execution point of most Java programs. |
| - Compiling and executing Java code on the command line |  |
| - JavaDocs - What is it, what is it used for, how do you access it? |  |
| - Primitive data types |  |
| - boolean - | Returns True and False only |
| - byte - | 1 byte/8 bits of memory |
| - char | 16 bits of memory, uses a letter within single quotes |
| - short | 16 bits of memory |
| - int | (default) 32 bits of memory, whole integers only |
| - float | 32 bits of memory, holds decimal numbers |
| - double | (default) 64 bits of memory, holds decimal numbers |
| - long | 64 bits of memory |
| - Operators | A special symbol applied to a set of variables, values or literals that can return a result. There are 3 operators (Unary, Binary and Ternary). |
| - pre / post-increment | **Pre-increment** - Increases or Decreases the expression by 1 and then returns a value.  **Post-increment** - returns a value then increases or decreases the expression by 1. |
| - ternary |  |
| - logical |  |
| - Scopes of variables |  |
| - static / class |  |
| - object / instance |  |
| - method |  |
| - block |  |
| - Control flow statements |  |
| - for |  |
| - enhanced for loop |  |
| - if/else |  |
| - while |  |
| - do-while |  |
| - switch |  |
| - Class members |  |
| - Fields |  |
| - Methods |  |
| - Constructors |  |
| - Access modifiers |  |
| - public |  |
| - protected |  |
| - default |  |
| - private |  |
| - Non-access modifiers |  |
| - static |  |
| - final |  |
| - abstract |  |
| - synchronized |  |
| - transient |  |
| - Packages and import statements |  |
| - static imports |  |
| - this and super keywords |  |
| - Constructor chaining |  |
| - JRE library classes |  |
| - Object |  |
| - equals, hashCode, and toString methods |  |
| - System |  |
| - String |  |
| - StringBuilder |  |
| - StringBuffer |  |
| - Collections |  |
| - Arrays |  |
| - Wrapper classes |  |
| - Autoboxing / unboxing |  |
| - String pool |  |
| - == vs .equals() |  |
| - Making objects immutable |  |
| - Varargs |  |
|  |  |
| **### OOP** | |
| - Object-oriented programming principles |  |
| - Abstraction |  |
| - Abstract classes |  |
| - Interfaces |  |
| - Polymorphism |  |
| - Overloading |  |
| - Overriding |  |
| - Covariant return types |  |
| - Inheritance |  |
| - Inheriting from classes vs interfaces |  |
| - Encapsulation |  |
| - Using access modifiers with getters/setters |  |
| - Object vs class |  |
| - Abstract classes |  |
| - Difference between abstract and concrete class |  |
| - Interfaces |  |
| - Implicit modifiers for variables and methods |  |
| - When to use instead of abstract class |  |
|  |  |
| **### Collections & Generics** | |
| - Collections API |  |
| - Inheritance hierarchy |  |
| - List, Set, Map, and Queue interfaces and their differences |  |
| - Concrete implementations of above interfaces and their differences |  |
| - Using enhanced for loops |  |
| - Comparable and Comparator interfaces |  |
| - Iterable vs Iterator interfaces |  |
| - Annotations |  |
| - @Override |  |
| - @Deprecated |  |
| - Generics |  |
| - Generic classes |  |
| - Generic methods |  |
| - Diamond operator and type inference |  |
| - Serializable interface |  |
| **### Exceptions** | |
| - Class hierarchy |  |
| - Error |  |
| - Exception |  |
| - RuntimeException |  |
| - Checked vs unchecked exceptions |  |
| - Handling or declaring exceptions |  |
| - try/catch/finally blocks |  |
| - rules for multiple catch blocks |  |
| - try-with-resources |  |
| - Creating custom exceptions |  |
|  |  |