**## 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) | Application Programming Interface (API) is a list of all classes that are part of the Java development kit (JDK). It includes all Java packages, classes, and interfaces, along with their methods, fields, and constructors. These prewritten classes provide a tremendous amount of functionality to a programmer. |
| - Object-oriented | Based on the concept of "[objects](https://en.wikipedia.org/wiki/Object_(computer_science)" \o "Object (computer science))", which can contain [data](https://en.wikipedia.org/wiki/Data" \o "Data) and code: data in the form of [fields](https://en.wikipedia.org/wiki/Field_(computer_science)" \o "Field (computer science)) (often known as attributes or properties), and code, in the form of procedures (often known as [methods](https://en.wikipedia.org/wiki/Method_(computer_science)" \o "Method (computer science))). |
| - Strongly/strictly typed | In Static Typing, type checking is performed during compile time. It means that the type of a variable is known at compile time. |
| - 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 Developers 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 | **Create** a file ending in .java ie .Dance.java  **Compile** the file in the terminal window ie javac Dance.java  If there are no errors, then a Dance.class file will be created.  **Execute** the file in the terminal window ie java Dance (don’t need the .class extension) |
| - JavaDocs - What is it, what is it used for, how do you access it? | Document generator tool in [Java programming language](https://www.geeksforgeeks.org/java/) for generating standard documentation in HTML format. It generates [API](https://www.geeksforgeeks.org/introduction-to-apis/) documentation. It parses the declarations and documentation in a set of source file describing classes, methods, constructors, and fields.  JavaDoc comments /\*\*………………..\*/  To create a JavaDoc you do not need to compile the java file. To create the Java documentation API, you need to write Javadoc followed by file name.  javadoc file\_name or javadoc package\_name |
| - 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 | Java ternary operator is the only conditional operator that takes three operands. It’s a one-liner replacement for if-then-else statement. |
| - logical | “AND”, “OR” and “NOT” operations (Uses boolean values) |
| - Scopes of variables | Scope of a variable is the part of the program where the variable is accessible. |
| - static / class | Static means that the variable or method “belongs to” the class, instead of each object of the class.  Variables declared with the static keyword reside in the static or class scope.  Static variables persist throughout the lifetime of your entire program. static variables share a value across all object instances of a class. Changes to the variable value in one object will change the value in all objects. |
| - object / instance | Variables created outside of any method, or flow control statement within a class reside in the Instance or object scope. Instance scope variables persist through the lifetime of the object. |
| - method | These variables can be utilized within a method but cannot be accessed outside of the method in which they are declared.  Method scope variables are also known as local variables. |
| - block | Variables created within a flow-control statement or loop reside in the Block scope.  These variables can be utilized within the block they are declared but cannot be accessed outside it. |
| - Control flow statements | Controls the flow of execution, with if/else conditions |
| - for | For loop provides a concise way of writing the loop structure. Unlike a while loop, a for statement consumes the initialization, condition and increment/decrement in one line thereby providing a shorter, easy to debug structure of looping. |
| - enhanced for loop | Enhanced for loop provides a simpler way to iterate through the elements of a collection or array. It is inflexible and should be used only when there is a need to iterate through the elements in sequential manner without knowing the index of currently processed element. |
| - if/else | If - if statement is the most simple decision making statement, if true, the code block will execute.  Else - The second part of an If-Else statement. When the “if” code block is false, the else block will run. |
| - while | A while loop is a control flow statement that allows code to be executed repeatedly based on a given Boolean condition. The while loop can be thought of as a repeating if statement. If the condition is true, it keep running. Once the condition becomes false, it terminates. It checks before each iteration. |
| - do-while | Similar to a while loop except it will execute once before checking the parameters. It checks after each iteration. |
| - switch | The switch statement is a multiway branch stk is false, then the “else” part of the code will execuatement. It provides an easy way to dispatch execution to different parts of code based on the value of the expression. Simple way of creating an if/else if/else statement. |
| - Class members |  |
| - Fields | Variables and states |
| - Methods | Behavior  A method is a collection of statements that perform some specific task and return the result to the caller. A method can perform some specific task without returning anything. Methods allow us to reuse the code without retyping the code. |
| - Constructors | Constructors are used for initializing new objects  They have no return type and always are named with the same name as the class in which they are defined.  Every class must have at least one constructor.  If you don’t provide one, a default no-argument constructor is implicitly provided. |
| - Access modifiers | Helps to restrict the scope of a class, constructor, variable, method, or data member. There are 4 types of modifiers. |
| - public | All-access |
| - protected | Available to same class, same package (child and non-child) and different package child |
| - default | Available to same class, same package (child and non-child) |
| - private | Same class only |
| - Non-access modifiers | 7 Special keywords that can add or limit functionality to the given method. |
| - static | When a member is declared static, it can be accessed before any objects of its class are created, and without reference to any object. |
| - final | When a variable is declared with final keyword, its value can’t be modified, essentially, a constant. This also means that you must initialize a final variable.  Final Variable = To create a constant variable  Final Method = Prevent method overriding  Final Classes = Prevent inheritance  Declares a resource (class, method or field) as the last implementation, which cannot be extended, overridden, or changed. |
| - abstract | Keyword abstract is used to make a class abstract.  Allows for the declaration of a class or method without implementation. |
| - synchronized | only one thread can access the resource at a given point of time. |
| - transient | **Transient**is a variables modifier used in [serialization](http://quiz.geeksforgeeks.org/serialization-in-java/" \t "https://www.geeksforgeeks.org/transient-keyword-java/_blank). At the time of serialization, if we don’t want to save value of a particular variable in a file, then we use transient keyword. When JVM comes across transient keyword, it ignores original value of the variable and save default value of that variable data type. Prevents the resource from being serialized. |
| - Packages and import statements |  |
| - static imports |  |
| - this and super keywords | **‘this’** is a reference variable that refers to the current object.  Super - The current class’ parent class. |
| - Constructor chaining | Constructor chaining is the process of calling one constructor from another constructor with respect to current object. |
| - JRE library classes |  |
| - Object |  |
| - equals, hashCode, and toString methods | **equals()** - determines if two objects are equivalent, i.e. whether or not they have the same values in every variable. Uses the result of hashCode() by default.  **hashCode()** - generates an integer that uniquely identifies an object from all other objects of a class. By default, typically uses the memory address of the object (JVM-specific).  **toString()** - returns a String representation of the object. By default, prints the class name and the memory address of the object. |
| - System |  |
| - String |  |
| - StringBuilder | Creates a mutable sequence of characters. An alternative to String class. Not synchronized |
| - StringBuffer | Creates a mutable sequence of characters. An alternative to String class. Synchronized |
| - Collections | Collections is a utility class present in java.util.package. It defines several utility methods like sorting and searching which is used to operate on collection. It has all static methods. |
| - Arrays | An array is a group of like-typed variables that are referred to by a common name.  An array is a collection of items stored at contiguous memory locations. |
| - Wrapper classes | A Wrapper class is a class whose object wraps or contains primitive data types. Allow you to treat primitives like objects. |
| - Autoboxing / unboxing | **Autoboxing** - Converting a primitive value into an object of the corresponding [wrapper class](https://www.geeksforgeeks.org/wrapper-classes-java/).  **Unboxing** - Converting an object of a wrapper type to its corresponding primitive value |
| - String pool | String literal, created by using equal sign and quotes, in a separate part of the heap |
| - == vs .equals() | **==** makes a reference/memory comparison. This is a operator. Are the references the same?  **Equals()** makes a content or value comparison. This is a method. Are the references equivalent? |
| - Making objects immutable | Unchangeable |
| - Varargs | short-form for variable-length arguments.  A method that takes a variable number of arguments is a varargs method. |
|  |  |
| **### OOP** | |
| - Object-oriented programming principles | **A**bstraction, **P**olymorphism, **I**nheritance, **E**ncapsulation (A PIE) |
| - Abstraction | Only the essential details are displayed to the user. The trivial or the non-essentials units are not displayed to the user. |
| - Abstract classes | An abstract class is a class that is declared abstract. It may or may not include abstract methods. Abstract classes cannot be instantiated, but they can be subclassed. |
| - Interfaces | Allows for the implementation of an abstract method, particularly within interfaces |
| - Polymorphism | Having many forms;  Polymorphism allows us to perform a single action in different ways. In other words, polymorphism allows you to define one interface and have multiple implementations.  The ability for objects, classes, variables and/or methods to alter functionality while maintaining structure |
| - Overloading | Overloading allows different methods to have the same name, but different signatures where the signature can differ by the number of input parameters or type of input parameters or both. |
| - Overriding | Overriding is a feature that allows a subclass or child class to provide a specific implementation of a method that is already provided by one of its super-classes or parent classes. When a method in a subclass has the same name, same parameters or signature, and same return type(or sub-type) as a method in its super-class, then the method in the subclass is said to override the method in the super-class.  Changing the implementation of an inherited behavior |
| - Covariant return types |  |
| - Inheritance | It is the mechanism in java by which one class is allow to inherit the features (fields and methods) of a parent or super class. |
| - Inheriting from classes vs interfaces | You can only extends from one class, but you can interface with several collections. |
| - Encapsulation | It is the wrapping up of data under a single unit. It is the mechanism that binds together code and the data it manipulates. It is a protective shield that prevents the data from being accessed by the code outside this shield. |
| - Using access modifiers with getters/setters |  |
| - Object vs class | **Class** - the blueprint for an object. Classes have states (variables) and behaviors (methods) that are shared among all the objects in that class.  An **Object** is an instance of a class in memory. Accessed through a reference, not directly. |
| - Abstract classes | The keyword Abstract is used to make a class abstract. Allows for the declaration of a class or method without implementation. |
| - Difference between abstract and concrete class | Abstract methods have a declaration, but no definition (method body). whereas concrete methods have both. |
| - Interfaces | an interface can have methods and variables, but the methods declared in an interface are by default abstract (only method signature, no body) and public. |
| - 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 | **List** (Subinterface of the collection) - It is an ordered collection of objects in which duplicate values can be stored.  **Set** (Subinterface of the collection) - It is an ordered collection of objects in which duplicate values cannot be stored.  **Queue** (Subinterface of the collection) - It is an ordered list of objects with its use limited to insert elements at the end of the list and deleting elements from the start of the list. it follows the FIFO or the First-In-First-Out principle.  **Map -** The Map interface present in [java.util](https://www.geeksforgeeks.org/java-util-package-java/) package represents a mapping between a key and a value. The Map interface is not a subtype of the [Collection interface](https://www.geeksforgeeks.org/collections-in-java-2/). Therefore it behaves a bit differently from the rest of the collection types. A map contains unique keys. |
| - Concrete implementations of above interfaces and their differences |  |
| - Using enhanced for loops |  |
| - Comparable and Comparator interfaces | **Comparable Interfaces** - A comparable object is capable of comparing itself with another object. The class itself must implements the java.lang. Comparable interface to compare its instances.  **Comparator Interfaces** - Comparator interface is used to order the objects of user-defined classes. A comparator object is capable of comparing two objects of two different classes. |
| - Iterable vs Iterator interfaces | Iterable -  Iterator Interfaces - |
| - Annotations | Annotations are used to provide supplement information about a program. |
| - @Override | Declares the method must override an inherited method. If it does not, a compilation error will occur. |
| - @Deprecated | Marks a method as obsolete, resulting in a compilation warning anywhere it is used. |
| - Generics |  |
| - Generic classes |  |
| - Generic methods |  |
| - Diamond operator and type inference | <> |
| - Serializable interface | It is used to make an object eligible for saving its state into a file. |
| **### Exceptions** | |
| - Class hierarchy |  |
| - Error | An Error indicates serious problem that a reasonable application should not try to catch. A critical failure. |
| - Exception | An exception is an unwanted or unexpected event, which occurs during the execution of a program i.e at run time, that disrupts the normal flow of the program’s instructions. A problem we can fix.. |
| - RuntimeException | Unchecked errors because the code was compiled and run. |
| - Checked vs unchecked exceptions | **Checked** - are the exceptions that are checked at compile time.  **Unchecked exceptions** are exceptions that are not checked at compile time. Don’t have to handle them, but should. |
| - Handling or declaring exceptions |  |
| - try/catch/finally blocks | **Try** - contains set of statements where an exception can occur.  **Catch** - Catch block is used to handle the uncertain condition of try block. A try block is always followed by a catch block, which handles the exception that occurs in associated try block.  **Finally** - Finally blocks always run |
| - rules for multiple catch blocks |  |
| - try-with-resources |  |
| - Creating custom exceptions |  |
|  |  |