|  |  |
| --- | --- |
| Java: Variables | |
| Primitive Data Types | \*\*The most basic data types of Java and are different from objects because variables of primitive data types store an actual value in memory.  \*\*The 8 primitive data types are: int, long, float, double, byte, short, Boolean, char |
| Objects | Objects are the “custom data types” of Java and are defined by classes. They are different from primitive data types in that they are references/pointers to a location in memory. Besides the primitive data types, everything in Java is an object. |
| Modifiers | There are two categories of access modifiers in java: access modifiers & non-access modifiers. Access modifiers specify the scope of entities (including data members, methods, constructors or classes).  Access Modifiers   * **public** - variable is anywhere. “default” (package level) - variable is only accessible within the package. * **protected** - variable is only access it able in subclasses and the class scope it is defined in. * **private** - variable is only accessible in the class.   Non-Access Modifiers   * **static** - variable belongs to the class itself. * **final** - variable cannot be changed. **NOTE:** final objects (meaning arrays too) can change its instance variables. * **Abstract** * **Synchronized** * **Transient** * **Volatile** * **Native** |
| Variable and Method Scopes | * Instance Scope   + scope where variables/methods accessible to an instance is created.   + They are initialized to default values in which primitives are set to zero casts to for each and objects are set to null. * Class Scope   + scope where variables/methods belonging to the class that it was defined in.   + This is typically done with the static modifier. In this scope, instance variables are not accessible because they are not known until an instance has been created. * Block Scope   + scope where variables belonging to a block of code within {} brackets and are only accessible within the block.   + This scope can access the enclosing scope or lexical scope. * Method Scope   + scope where variables are only accessible throughout the same method.   + Method scope can access anything within the enclosing class but cannot access static variables. |
| Variable Arguments “varargs” | Variable Arguments “varargs”   * A parameter that is capable of taking in multiple inputs of the same type as an array. * Java does this by using the triple dot operator. * This must be the last parameter in a method signature.   + ie. someMethod(Object...objs) * Also implies that only one varargs allowed per method.   + ie. someMethod(int n, boolean b, String...strs) * Fun Fact: Can be used in main method.   + ie. public static void main (String...Cheese) { /\* does something \*/ }   Naming Conventions   * Camel Case * Underscore for Spaces * Underscore beginning   What can be used as a variable name?   * Letters, numbers, underscores * Cannot lead variable name with numbers. |
| Classes | Class are the “blueprint” for building a particular object. They define the fields and behavior of an object.  A type that defines the implementation of an object - in other words, it’s sort of a “blueprint”  Constructors   * Default Constructors - no argument constructors given to classes that have no explicitly defined constructors. (This is not the same as no-args constructors which can be explicitly defined.) * super() - a keyword method that calls the immediate super class’ no-args constructor. * this() - a keyword method that calls the no-args constructor of the this method was called in.   Instance   * Instances are “example” occurrences of an class. Objects are instances of classes. |
| Java: Static Classes | |
| Wrapper Classes | Auto-Boxing   * The automatic conversion of primitives into their corresponding wrapper class, which is performed by the compiler. When the process happens the “other way” it is called unboxing. |
| Arrays | * Grouping of entities of the same type with a fixed size and zero-based index. * Arrays are defined with [] brackets and are static (in this case, it means unchanging/fixed length). |
| Arrays static class | This class contains various methods for manipulating arrays (such as sorting and searching). This class also contains a static factory that allows arrays to be viewed as lists.  Here is a list of handy methods to keep in mind:   * binarySearch() * copyOf() * equals() * fill() * sort() |
| Java: Control Statements | |
| Control Flow Statements | Statements are generally executed in a top-to-bottom manner, but control flow statements break up the flow of execution via decision making, looping, and branching.   * Controls the order of execution   + Selection     - If, if else, else     - If something is true, do that     - Otherwise do something else   + Switch     - FallThrough - if you don’t have a break statement, it will just run everything     - String, enum, ints, any primitive that can be casted to int     - Checks the variable     - Executes code based on the variable’s value * Loop   + While     - Executes a block of code while condition is true     - Checks condition at start   + Do while     - Same as while loop, but executes code at least once     - Checks condition at end   + For     - Executes code a number of times     - Initialization     - Condition     - Increment   + For each/ enhanced for loop     - Checks every instance of a list * Transfer   + Continue     - Ends a specific iteration of a block.     - Does not end the loop.     - while, do or for   + Break     - Breaks out of a block.     - Will end a loop.     - for, while, do or switch statement   + Return     - Ends the loop, breaks out of function/method call and returns a value |
| Java: Architecture Overview | |
| JRE/JVM/JDK | * JRE (Java Runtime Environment):   + The software layer that runs on top of the operating system, providing the environment which Java programs run in.   + Contains the jvm and libraries needed to run the app * JVM (Java Virtual Machine):   + Essentially, a program that runs over programs.   + Converts Java files to bytecode to be read by any OS   + It allows Java programs to run on any OS - Write once, run anywhere   + Manages and optimizes program memory * JDK (Java Development Kit):   + Has the JRE, plus the java compiler and development tools like debuggers |
| OOP | * What is object-oriented programming   + Data modeling, logical organization of real-world objects   + Classes = blueprints * Inheritance   + Deriving a class from another class and keeping the parent class’ attributes   + IS-A / HAS-A relationship * Encapsulation   + Hiding of implementation details internally within a class   + defined as the restriction of data access.   + Accessors and modifiers * Polymorphism   + Ability to process objects differently based on their data type   + Overriding - @Override     - Dynamic     - Same function name   + Overloading -     - static     - same function name and different parameters   + Covariant types - instantiating a class based on its super type     - Animal a = new Dog(); * Abstraction   + Manage complex systems with smaller bits of pieces   + the hiding of implementation details. |
| Naming conventions | * Packages: must begin with a lowercase letter * Classes: must begin with a uppercase letter * Variables: must begin with a lowercase letter, camelcase recommended for long variable names * Static Final Variables: all uppercase letters |
| Casting | Changing an object and changing its object type  Enclosing a type in parentheses before using the object tells the compiler that the “this” object is an instance of “B” instead of “A”.  For example:  double d = 2.99d;  int n = (int) d;  Keep in mind that a ClassCastException may occur if you try casting objects that belong to different hierarchies. |
| Method Overloading | Method Overloading is intended to increase readability in programs; method names must be the same but the parameters must be different. |
| Method Overriding | Method Overriding is intended to provide a specific-implementation for a method already provided by a superclass, the method name and parameter must be the same. Overridden methods may not have its access narrowed. |
| Generics | Generics are used to specify what type of object will be in your collection, and has three advantages:   * Type-safety * Type casting not necessary * Compile time checking |
| Java: Memory Model | |
| Heap | In the context of a data structure, a heap is a tree-based data structure that organizes its nodes by priority. In the context of Java, heaps are where objects live and contains the string pool. |
| Stack | In the context of a data structure, stacks are data structures that follow a First-In Last-Out (FIFO) order. In the context of Java, stack is where method invocations and local variables are stored. It’s worth noting that each thread gets a single execution stack. |
|  | |
| String | String Literal  String str = “GeeksForGeeks”; |
| String Pool | The string pool allows string constants to be reused, which is possible because strings in Java are immutable. If you repeat the same string constant all over the place in your Java code, you can actually have only one copy of that string in your system, which is one of the advantages of this mechanism.  String str = new String(“GeeksForGeeks”); |
| Interface | If classes are blueprints for objects, then interfaces are blueprints for classes. They contain constants and methods (which are implicitly implied to be public and abstract).   * Functional interfaces: interfaces with only one abstract method and allows for lambda expressions * Marker interfaces: empty interfaces intended to designate a class with special properties |
| Multithreading and Concurrency | What is multi - Running multiple threads at the same time   * + What is a thread?     - A path of execution in an application   + States of a thread     - New     - Runnable     - Waiting     - Blocked     - terminated   + Thread priority     - setPriority()       * Thread.MIN\_PRIORITY       * Thread.MAX\_PRIORITY   + Important methods of the Thread class     - start()     - sleep()   + Runnable interface     - Implement runnable     - Is a functional interface   + How do we spin up a thread?     - Call start(), don’t call run()   + Synchronized     - Only 1 thread at a time   + Deadlock   https://lh3.googleusercontent.com/jyA-KcUNG_5dDX6FaQfOxHkaGyJv6iVJa0gIc_G2RMZIb-OjF-O6CQFaK2rfH-d24gtZAp9Jp7VUqzTCpN3KLkunSsEJ6IZ96TwZtQoLgBAlNUHvAlG2VuCSqabNF1XW5e9qZXQe   * + - When two processes are waiting for each other to give up a resource that the other is holding     - Some resources on deadlock       * <https://www.javahelps.com/2015/06/thread-deadlock.html>       * https://www.geeksforgeeks.org/deadlock-in-java-multithreading/   + Starvation     - A thread is unable to gain regular access to a shared resource, and is unable to finish     - This is different from deadlock because it’s not another thread holding the resources necessary to complete.     - This is usually caused by a priority level of a thread to complete.   + Producer Consumer problem     - Producer is trying to add stuff to a shared buffer and Consumer is trying to remove stuff from the buffer at the same time     - Problem is producer adding to a full buffer, and consumer taking from an empty buffer |
| Week 2: SQL, PL/SQL, JDBC | |
| SQL - Structured Query Language | **SQL - Structured Query Language**   * Language that we use to communicate with databases * Various dialects dependent upon particular db vendor you use. * Scripting language * Commands are divided into sublanguages   + **DDL** - Data Definition Language - used to define the *schema* of the DB.     - *CREATE, ALTER, TRUNCATE, DROP, REPLACE*   + **DML** - Data Manipulation Language - interaction with the data and also contains our *CRUD* commands.     - *INSERT, SELECT, UPDATE, DELETE*   + **DCL** - Data Control Language - specify user permissions within a DB.     - *GRANT, REVOKE*   + **TCL** - Transaction Control Language - manage transactions (units of work done on the database)     - *COMMIT, ROLLBACK, SAVEPOINT* * **DQL** - Data Query Language - used to query the DB.   1. *SELECT*   + *What is the difference between DROP, TRUNCATE, and DELETE?*     - *DROP* removes the *entire* DB object from the schema. Can be used on tables, constraints, views, sequences, and essentially any other object made with *CREATE*.     - *TRUNCATE* is used to remove all rows from a table, but keeps the schema of the table.     - *DELETE* is used to delete specific rows from a table using the *WHERE* clause (ie. DELETE FROM Users WHERE UserId = 1). If the *WHERE* clause is not present, (ie. DELETE FROM Users), all rows from the table are removed.     - *DELETE*, unlike *TRUNCATE* or *DROP*, can be rolled back.     - *ROLLBACK can only* be applied to *DML* commands, *NOT DCL* commands. |
| System-defined functions - functions natively available via Oracle | •Aggregate functions - used to do operations from the values of a column or subset of a column and returns a single value ie avg(), sum(), max(), min(), count()  •Scalar functions - one to one relationship between input: output values, used to alter every row in a column and gives single output based on each row value. Ie. length(), upper(), lower(), round() |
| RDBMS - Relational Database Management System | •Software used to interact with relational databases  •Relational databases are a particular kind of db that store data in tables with relationships ie. Oracle SQL, MySQL, PostgreSQL  •Non-Relational DB aka NoSQL store data in many forms like documents or graphs depending on the type ie. MongoDB, Apache Cassandra |
| Important Commands and Queries | •Set - UNION, UNION ALL, INTERSECT, MINUS  •Joins  •What is the difference between a union and a join?  •A JOIN is a means for combining columns from one or more tables by using values common to each whereas a UNION, one of the set operators, compares two or more similar result sets (unique values of result set A + result set B)  •WHERE - clause gives a condition to regular sql queries.   * delete from [table] where [condition] * update [table] set [change] where [condition]   •HAVING - used with aggregate functions and group by in order to give further specifications |
| DB Normalization | The process of organizing a database to increase efficiency and reduce redundancy.  Normalization is done in steps called the normal forms. There are many normal forms, we try to be in the third (3NF).  •1NF - Every table must have a primary key. Atomic data (columns must each hold smallest units of data. Ie “Name”--> “First Name”, “Last Name”). No repeat columns (customer1, customer2...)  •2NF - is 1NF and has no partial dependencies (all columns depend on the entirety of the PK)  •3NF - is 2NF and has no transitive dependencies (all columns are dependent directly on the PK, not on a column that depends on the PK) |
| Index | a distinct structure in the db used to speed up searching in the database. Indexes can be applied to columns or sets of columns that are commonly queried. They should not be used in excess, though, because they slow down insertion and removal. They are automatically created for primary key columns.  CREATE INDEX [name]  ON [TableName](Column name) [ASC|DESC];  } |
| PL/SQL - Procedural Language Extension of SQL | •What is it? Oracle’s flavor of SQL.  •Sequences (aka “Autonumbers”) - Objects in oracle used to generate a number sequence  •Triggers - stored procedure that fires when an event occurs  •Stored Procedures  •Block of executable code that has full DML and TCL capabilities. Can take any amount (0 to Many) of IN and OUT parameters.  CREATE OR REPLACE PROCEDURE (name)  [list of params]  {IS/AS}  [declare params]  BEGIN  [code to execute (use null; for empty execution)]  EXCEPTION  [exception handling]  END;  /  •Functions  •Blocks of executable code which RETURN 1 value (which can be a cursor, representing a result set FYI). Can also have 0 or many both in and out parameters. Use DQL only! No insert, update, or delete capabilities.  •Invoke via execute functionName();  •Cursors  •Oracle creates a “context area” with all information necessary for processing a SQL statement  •A cursor is a pointer to that context area  •There are implicit and explicit cursors  •Implicit cursors are created by oracle whenever we execute a DML statement (and no explicit cursor exists). We have no control over implicit cursors. (examples needed)  •Explicit cursors are programmer defined, used to obtain more control over a context area (examples needed)   * DECLARE cursor to initialize memory * OPEN cursor to allocate memory * FETCH cursor to retrieve data * CLOSE cursor to deallocate memory |
| Week 3: Html, Css, Etc… | |
| HTML - HyperText Markup Language | * DOM - Document Object Model   + Tree-like structure containing all the data of a particular document. In HTML, it is the entire thing. * Bootstrap   + An HTML framework used to “beautify” your HTML page. (Almost a must have) * Semantic tags * HTML5 * Doctype * Attributes * Elements * Forms |
| CSS - Cascading Style Sheets | * A style sheet language used to describe the presentation of HTML pages. * Layout modes * Box model * Media queries * Selectors |
| JavaScript | * What is it?   + A programming language used for scripting behind websites usually for interactions, animations, etc.   + JavaScript is a lightweight, interpreted programming language with object-oriented capabilities that allows you to build interactivity into otherwise static HTML pages * Data Types   1. Number - used for both integers and floating point variables.   2. Boolean   3. Strings   4. Object   5. Function   6. Symbol * Variable Scopes   1. Global - variable can be used everywhere within the script.      + see hoisting...   2. Local - variable can be used anywhere in the function they were defined in.      + The *var* keyword indicates a variable will be limited to the local scope.   3. Block - variable can only be used within the block it was defined in.      + The *const* and *let* keyword indicates that a variable is of block scope. * Functions   + Arrow notation     - var someVar = () => {};     - The *this* keyword refers to the lexical context the function is called in. * Hoisting   + A “hidden” behavior JavaScript uses to bring a particular variable up to the global scope. This is done to all variables declared without a *var*, *const*, or *let* keyword. * Semicolon Injection   + Semicolons not required everywhere because javascript injects it for you, but generally not good practice to leave them out. * jQuery * A lightweight javascript library that simplifies javascript features, as well as adding more utility. |
| Bubbling vs Capturing | Event handling   * Bubbling: Event is first captured and handled by the innermost element and then propagated to outer elements. * Capturing: The other way around |
| HTTP - Hypertext Transfer Protocol | * General means of communication in the World Wide Web * Not the only transfer protocol used, *but* it is the most widely used for a multitude of reasons including its ability to create dynamic applications through the use of the HTTP verbs.   https://lh5.googleusercontent.com/qw5fx5TbVUagGT4b-bAbf1443613eBF4fdYTDgQs9oTM8sXWmycEkN2LHZF0SsHyuLyU26SiwCYhFPH_DgXyG6CZA-cDvL6xs4kKx-f81YxcCm5TRvIBWtvIyegSLzHHjHNTkIQc  **Image**: An HTTP request.   * HTTP Verbs * (CRUD - Creation, Retrieval, Update, and Deletion)   1. *POST*      + Good for sending large amounts of data      + Typically used to add data to a server      + Includes information in the request body, not URL   2. *GET*      + Generally used for retrieving information from the server.      + Append parameters to the URL(is a limit on how much data you can send with a GET request, but this depends on your browser and the max length of a query) in key-value pairs.   ie. url?key1=val1&key2=val2   * + - No request body!!!!!!     - Do not use for sensitive data.   1. *PUT*      + Used to replace all current representations of the target resource with the request body.   2. *DELETE*      + Used to delete entities.   + *HEAD*     - The same thing as “*GET”*, but does not return with a response body. Essentially just returns the metadata.   + *OPTIONS*     - Used to describe the communication options for the target resource.   + *TRACE*     - Used for debugging which echoes input back to the user.   + *PATCH*      - Used to apply partial modifications to a resource.   + [*CONNECT*](https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods/CONNECT)     - Establishes a tunnel to the server identified by the target resource     - **Tunneling** is a protocol that allows for the secure movement of data from one *network* to another. Tunneling involves allowing private *network* communications to be sent across a public *network*, such as the Internet, through a process called **encapsulation**. * HTTP status codes * 100-199 - informational * 200-299 - success   + 200 - OK   + 201 - Created * 300-399 - redirect * 400-499 - client-side errors/bad request   + 400   + 404 - Not Found   + 405   + 409 - Conflict   + 418 - I’m a Teapot * 500-599 - server errors |
| JSON - JavaScript Object Notation | * JSON.parse * JSON.stringify |
| Ajax | AJAX stands for Asynchronous JavaScript and XML. It is a group of related technologies used to display data asynchronously. In other words, it sends and retrieves data without reloading the web page. |
| Async request steps | * Create XHR variable for HTTP REQUEST * define onreadystatechange * open request * send request. |
| Type Coercion | Type coercion is the process of converting a value from one type to another (such as string to a number, object to boolean, and so on). Any type, be it primitive or an object, is a valid subject for type coercion |
| Truthy and Falsy | In JavaScript, a truthy value is a value that is considered true when encountered in a Boolean context. All values are truthy unless they are defined as falsy (i.e., except for false, 0, "", null, undefined, and NaN). |
| Guard and default operators | We can use the logical ‘AND’ (&&) and the logical ‘OR’ (||) when evaluating values. This can be useful for writing simple and concise return statements and variable assignments. The logical AND operator works like a guard |
| Closure |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |