**JAVA**

* Java is an object oriented programming language (OOP)
  + In OOP, computer programs are designed by making them out of objects that interact with one another

**Operators**

* Operators are built in Java to manipulate variables
* Arithmetic: -, +, \*, /, %(remainder)
* Relational: <, >, ==, <=, >=, !=(not equal to)
* Bitwise: &, |, ^, ~
* Logical: &&(and), ||(or), !(not)
* Assignment: +=(add and assign), =(assign), \*=(multiply and assign)

**If else statement**

* If(s1 == s2){A} else {B} \*(s1 == s2) is the condition statement
* If condition statement is true, execute A; otherwise execute B.

**If ElseIf Else statement**

* We use this when there are more than two possible outcomes
* If(s1 == s2){A} else if (s1 == s3){B} else {C}
* If condition is true, execute A. If it’s not true, check else if condition. If true, execute B, if not, execute C.

**Ternary conditional statement**

* If else statement in a single line of code, called Ternary conditional statement
* int pointsScored = 21;
* char gameResult = (pointsScored > 20) ? ‘W’ : ‘L’;
* System.out.println(gameResult);
* Above example, boolean expression is (pointsScored > 20), which evaluates to true. This will return the value of ‘W’, which is assigned to the variable gameResult. The value ‘W’ is printed to the console.

**Switch statement**

* Java provides a way to execute code blocks based on whether a block is equal to a specific value
* int restaurantRating = 3;
* switch (restaurantRating) {
* case 1: System.out.println(“This restaurant is not my favorite.”);
* break;
* case 2: System.out.println(“This restaurant is good.”);
* break;
* case 3: System.out.println(“This restaurant is fantastic!”);
* break;
* default: System.out.println(“I’ve never dined at this restaurant.”);
* break;
* }
* In example above, we assigned the int variable restaurantRating a value of 3. The code will print a message to console based on the value of restaurantRating
* The break statement will exit the switch statement after a condition is met. Without the break statement, Java will continue to check whether the value of restaurantRating matches any other cases.
* The default case is printed only if restaurantRating is not equal to an int with the value of 1, 2, or 3.

**Variable Key Points**

* To initialize a variable
  + Must define the type and the name
  + Optionally, you can assign value to the variable, you would need name of the variable, the equals sign, and the value to be assigned
  + int i = 3+4; (in this case, we ***execute the right hand side***, results 7, ***THEN we assign*** to variable i)

**Variables and Types**

* To initialize a variable with Primitive Type
  + First Way : Combining initialization and setting value

int num = 1;

* + Second Way : Initialize first, and setting value later

int num; (first computer assigns “null” value)

num = 1; (then we assign “1” value)

* To initialize a variable with Object Type
  + First Way : Combing initialization and setting value

Object obj = new Object(“Name”, 6);

* + Second Way : Initialize first, and setting value later

Object obj; (“null” value)

obj = new Object(“Name”, 6);

**Function key points**

* How do you define a function, you have to first define a function signature, for example “public String getName()”
  + Define **access** **level**
  + Define whether it’s **static** or **instance** function
  + Define the **return type** (the type for the output)
  + Define the **name** of the function
  + Define the **input** that function needs
  + Define the **body**, which is what the function actually do
* Differences between Static and Instance functions
  + Static function is when you use without the Object
  + Instance functions is when you use it with the Object
* In Java, functions are **always** passed by Value
  + What is **Pass by Reference (NEVER IN JAVA)**
    - The inputs of the function are references of the original, so when you change the inputs in the function, it changes the original as well
  + What is **Pass by Value**
    - The inputs of the function are clones of the original input, when you change things in the function, it changes the clone instead of the original
* compareTo() function is VERY IMPORTANT!
  + It is a default function for most objects
  + For example: object1.compareTo(object2);
    - if by comparison object1 is smaller than object2, then return will be -1
    - if by comparison object1 is equal to object2, then return will be 0
    - if by comparison object 1 is greater than object 2, then return will be 1

**Object vs Primitive**

* Objects are created by developers
  + Objects are assigned by address
* Primitives are built-in in Java
  + Primitives are assigned by value
* Because primitives have predefined frame of rules (ie: int has 32 bits, whole number etc), Jave has specific place for them. Therefore, Java can assign them by values because they are always consistent
* On the other hand, objects do not have predefined frame of rules (ie: objects are created by developers. Objects are always different), Jave cannot predefine a space to allocate. Therefore, Jave created a pattern to use objects. They are assigned by addresses

**String vs Object (How object comparison works)**

* String is an object, and it is also predefined by Java
* Example1: String s1 = “hello”; String s2 = “hello”; s1==s2 is **TURE**
* Example2: String s2 = “hello”; String s3 = new String(“hello”); s2==s3 is **FALSE**
* Example3: Bike b1 = new Bike(5); Bike b2 = new Bike(5); b1==b2 is **FALSE**
* Example4: Bike b1 = new Bike(5); Bike b3 = b1; b1==b3 is **TRUE**
* Why? Because String has its own memory pool in Java. When you try to create a String not using the word new, it will go to its pool to find if there’s a same exact word hence addresses are the same. Therefore, example1 returns TRUE
* Example2 on the other hand, s3 is created with new String(“hello”), therefore, new space is assigned and address are different. Returning FALUSE
* Example3, Objects are always created with new, therefore, address are different, returning FALSE
* Example4, b1 is assigned to b3, therefore share the same address, returning TRUE

**Object key points**

* How do you define an object
  + Define properties
  + Define constructors
    - Constructor is a function to create an object
    - Constructor has NO return type
    - Constructor name has be the SAME as the class
    - Constructor is usually used to assign initial values to the properties
  + Define methods
* Instance of an object is **the** object created at that time and space
* The function used by that instance of the object is called instance function
* Key word this references the current object
* Wrapper object is an object that wrap around something else…
  + For example: Integer is a wrapper of int… Integer has value of int but it includes functions etc…
* How to create an new object
  + Object ob1 = new Object();
* What happens when you create an object
  + Java will reserve a space in the memory and store any properties or functions you define
  + But when you assign the new object into a variable name, it’s the address of where it’s assigned
* Usually, Objects will have the following functions
  + Constructor()
    - Function that constructs the object
  + Getters() and Setters()
    - Gets properties or sets the properties of the object
  + toString()
    - Function that changes turns this object in string representation
  + compareTo()
    - Function that compares this object with another object
    - Current compares to 2nd object
      * Current < 2nd object… return negative 1
      * Current > 2nd object… return positive 1
      * Current = 2nd object… return 0
      * What rule to define <, >, or =…is up to the programmer
  + equals() discussed on 03/03, make your own notes
    - Method is used to compare this string with the passed object as argument
  + hashCode()
    - Function that returns the unique identifier of this object
* In Java every variable name is just a representative of something else
  + For example:
    - int n = 1; (n is a representative of 1)
    - String str = “Hello”; (str is a representative of “Hello”)
    - BubbleTea[] youngList = new BubbleTea[2]; (youngList is a representative of array BubbleTea with 2 spaces)
    - youngList[0] = new BubbleTea(7, “Crazy”); (youngList[0] is a representative variable name for a BubbleTea)
    - Since youngList[0] is just a representative of a BubbleTea object, then we can use the instance function directly… youngList[0].getPrice();
    - List object has a function to return array… For example…list1.getBubbleTeaArray(); This this example, list1.getBubbleTeaArray() is just a representative of a BubbleTea array… list1.getBubbleTeaArray()[0];
    - Expanding on this, we can chain functions like so
    - list1.getBubbleTeaArray()[0].getPrice();

**Array key points**

* Initializing an array
  + First Way : Initializing the variable and set the values

int[] array1 = {1, 7, 12};

* + Second Way :

int[] array1; (Initializing the variable)

array1 = new int[3]; (create new array and reserve 3 spaces)

array1[0] = 1; (setting value into individual indexes)

array1[1] = 7;

array1[2] = 12;

**Loop Key points (for or while loops)**

* Why do you want to write a loop?
  + Perform the same calculation over a large population
* For loops has a variable to store the current index
* Two types of for loop
  + For loop: for(int i = 0; i < someList2.size(); i++)
    - In this case, i is the index of the arrayList, and that’s the iterator
  + For each loop: for(int x = 0; x < someList2)
    - In this case, x is the item of the arrayList, and that’s the iterator
  + How to sort in array HW 03/03
    - Int [] array = {1, 3, 11, 2, 5}; int temp; for (int i = 1; i < array.length; i++) { for (int j = i; j>0; j--) { if (array[j] < array[j-1]) {temp = array[j]; array[j] = array[j-1]; array[j-1]=temp;

**Collection Key Points**

* Collections : Specific term to describe Objects that implements the Collection Interface
* Queue and Stack are one variety of Data Structures
  + Queue is the same as waiting in line (first in first out)
  + Stack is the same as building Lego (first in last out)
* Collection in general means a group of objects
  + Example**:**
    - **ArrayList (Linear Data Structure) – more important in CapOne**
      * Just like an Array, however, use it when size need to be changed, can automatically resize…
    - **Vector (Linear Data Structure)**
      * Just like an ArrayList, however, it is synchronized…
    - **LinkedList (Linear Data Structure)**
      * Almost like an Array, however, it has no indexes…so in the get() function, we will have to look through whatever elements I want to get the value…the worst case would be ALL the elements…linear time (CON)
      * In insert(), Array is in linear time, because when it’s full, we need to create a new Array in double the size then copy each old indexes to new one…HOWEVER…insert() in LinkedList is constant time because when we want to insert() we just need to link a new element to the end or in the middle of the list. In the later we need to break the link then add the new one in between…(PRO)
    - **HashSet –**
    - **HashMap - more important in CapOne**
      * It’s a table where each item in HashMap is a key value pair
    - **Hashtable** 
      * Just like hashMap except it’s synchronized
    - **Queue**
    - **Stack**

**Casting Key Points HW 03/03**

* Taking one object of one type into another object type, and this process is called casting a variable

**Iterators**

* Iterators are temp variable that you use to iterate through a list or a collection…
* In other words, iterator is a mechanism to loop through a collection
* For example:
  + For ( int i = 0; i < length; i++ ) { }… in this case, **“i”** is an iterator

**Extends**

* The sub class does not need to redefine the properties and functions.
  + For example: sub class extends parent class
* It **cannot** extends parent class **private** properties and functions…
* Sub class can only extends **ONE** parent class…

**Interface**

* Interface is used to create rules (like fields and or signatures) that the class that implements the interface must follow.
* For example: if you have public String cry() in the interface, then your implemented class must have public String cry() as well.

**JSON object**

* Syntax is
  + {propertyName: propertyValue, secondPropertyName: secondPropertyValue}

**Input and Output**

* Inputstream is a way for Java to convert files or keyboard inputs into a format that java understands
* OuputStream is a way for java to convert data into a file or to console
* Properties class, it’s java built in utility class, this is used to load properties file into java
* Throw exceptions
  + Whenever Java needs to deal with input or outputs from connecting to a file or connecting to another server (database or anything else), we need to write a throw Exception statement at the method level, so java knows how to take care of issues.
* Surround with Try/Catch
  + The same thing as throwing exceptions, except the throw is right next to the code itself.

**Enumeration**

* Enum is a special Java type used to define collections of constants. More precisely, a Java enum type is a special kind of Java class. An enum can contain constants, methods etc.

**Data Structure**

A data structure is a generic term for an object that represents some sort of data, so a linked list, array etc are all data structures. A collection in the Java sense refers to any class that implements the Collection interface. A collection in a generic sense is just a group of liked objects.

* It is a definition of how data is stored and organized in computer so it can be used more efficiently
* Array is a type of data structure, it has a list data structure
* List of important data structures
  + List (index) – it’s organized linearly and used by calling the index
  + Tree
  + Map – it’s table which has items that are key value pairs
  + Set – in this set, all the items are unique, there cannot be duplicates
* Generics : ??
* Search function in array
* .Contains in arraylist
* .sort
* Comparator class

**Thread**

* Single sequential flow of control within a program
* Two or more tasks executing simultaneously
* Java.lang.thread class
* Must have a run function
* When you initialize it, that’s when the run started
* For example: go reference digitalClock project in netbeam

**Stub in Java**

* Partial code