Lesson 4

* The String.concat method “concatenates the specified string to the end of [the current] string”.
  + Just like the addition + operator concatenates Strings
  + Why then, if we create a String and use the concat method, then print out the String, does it not have the concatenated addition?
  + Strings are immutable. That means that once the object is created, it cannot be changed.
  + But, we have been concatenating Strings with the addition operator and it has been working??!?
    - Each time we do that we have assigned the result to another String.
    - Each concatenation creates a new String object in memory. This is not good.
  + Java offers a different class to remove this unnecessary memory suck, StringBuilder.
  + StringBuilder is mutable, and allows us to build up and manipulate parts of strings and then convert the final product to a string without the overhead of creating multiple new strings.
  + Most used constructors StringBuilder() and StringBuilder(String str)
  + Some useful methods:
    - Append
    - charAt(int index)
    - delete(int start, int end)
    - indexOf(String str)
    - length
    - replace(int start, int end, String str)
    - reverse()
    - toString()
  + Use a StringBuilder when you will be dynamically creating a String. Use String when you have a constant, or value that is only set once and does not change additively.
* In a previous lesson, arrays helped us group similar data together, however arrays have there limitations.
  + Set the size up front and they don’t shrink or grow
  + You always have to refer to the index you want to read from or write to (no method that just adds a new element to the next available index)
* A List is another data type that we can use, that works like an array, but overcomes some of the limitations of an array. First, a few things to note about Lists:
  + List is an interface, we will discuss this more when we talk about object oriented programming, but for now, just know that means we cannot create a List object, instead we have to create an object that implements List. To do this, we will use ArrayList.
  + Lists make use of a Java construct known as Generics. What that means is that you can have a List of any generic type. List is written like `List<E>` and that reads in English as `list of E`. When we declare a List, we replace the E with the type that List will contain. `List<String>` reads `list of String`.
  + Like arrays, Lists can only hold one data type at a time
  + Lists cannot be of a primitive data type. Luckily, each primitive data type has a class equivalent, i.e. int has Integer.
  + When we declare a new arraylist, we will do it like this `List<String> myList = new ArrayList<String>();`
  + We need to import both List and ArrayList from java.util
  + There are a lot of very useful methods on List, here are five of the most used methods
    - Add
    - Get
    - Size
    - Removes
    - And isEmpty
  + Let’s create a list of names and look at how much easier it is to manage than an array.
* List is a type of collection, which is another interface. Again, don’t worry about what interfaces are just yet, just know that a List is a Collection just like a Dog is an Animal, but just as Dogs are not the only animals, Lists are not the only Collections. While it is one of the most commonly used, there are others that have their own distinct, valuable attributes.
  + Three subsets of Collection are List, Set, and Map. Each is a different data structure with unique attributes, but each is still a collection of objects, or data.
  + List
    - Allows duplicates
    - Keeps everything ordered by index
    - Allows null values
    - Common implementations: ArrayList, LinkedList, Vector
  + Set
    - No duplicates
    - Unordered, cannot guarantee order of elements
    - Only allows one null, because of duplicate constraint
    - Common implementations: HashSet, LinkedHashSet, TreeSet
  + Map
    - Key value pairs (dictionary)
    - Values can be duplicate, but not keys
    - Common implementations: HashMap, LinkedHashMap, Hashtable, and TreeMap
  + Each implementation varies slightly and can be used for very specific use cases related to its attributes and how it handles the objects in its collection.
  + We’ve already seen examples of ArrayList, let’s look at examples HashSet, and HashMap.
  + Set methods
    - Add
    - Contains(Object)
    - isEmpty
    - remove(object)
    - size
  + map is written Map<K,V>, K and V are both generics and can be any class. K is the keys type and V is the type of the values.
    - Map<String, String> would be like an English dictionary. Text values (definitions) accessible by text keys (the word the is related to the definition). With a map, you look up a value by it’s key.
  + Map methods
    - Get(object key)
    - isEmpty
    - keyset – returns a set of the keys Set<K>
    - put(k key, v value)
    - remove(object key)
    - size()
    - values() – returns a collection of the values Collection<V>