Foundation - Week 2

- [ ] Google reference datatypes Java
- [x] ~~Solve HashMap exercises over the weekend~~
- [ ] Prepare lightning talk for week 3
- [ ] Solve PalindromeSearcher exercise

Method vs. function - in OOP context

A **function** is a piece of code that is called by name. It can be passed data to operate on (i.e. the parameters) and can optionally return data (the return value). All data that is passed to a function is explicitly passed.

A **method** is a piece of code that is called by a name that is associated with an object. In most respects it is identical to a function except for two key differences:

1. A method is implicitly passed the object on which it was called.
2. A method is able to operate on data that is contained within the class (remembering that an object is an instance of a class - the class is the definition, the object is an instance of that data).

(In Java context, only methods exist because every method is called on an object)

Data Structures

String Methods

- string.toLowerCase()
- string.toUpperCase()
- string.startsWith(prefix) - checks if element starts with the given parameter
- string.endsWith(suffix) - checks if element ends with the given parameter

- string.equals[IgnoreCase](anotherString) - checks if strings are equal, possibly with ignoring the casing
- string.concat(anotherString) - concatenates two string elements
- string.charAt(index) - gets the character in a String at the given index
- string.indexOf('char' or "substring") - returns the index of the first instance of the given character or substring
- string.indexOf('char', index) - will start searching for the char at the given index
- string.substring(startingindex, endingindex) - can cut out parts of Strings
- string.replace('char1', 'char2') - replace a char with another one in String
- string.trim() - trims whitespace in String (not spaces between words)

ArrayLists

- Collections are objects which group multiple elements into a single unit
- The ArrayList class is a type of a collection
- Collections contain methods for object management
- like an Array, Collections hold more than one object
- but they are not of a fixed size
- Collections cannot use primitive types (int, double)
    - so int > Integer
    - and double > Double

```
import java.util.ArrayList; int[] grades = new int[20]
ArrayList<Integer> grades = new ArrayList<>(); // Add
element to Collection grades.add(int index, Integer
element) // Remove element grades.remove(int index)
```

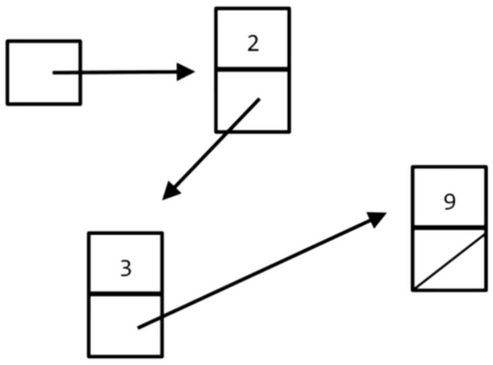
JavaScript

- Instead of .length, collections have .size
- Indexes can be called with collection.get(index) method
- collection.set(index, setvalue) - can change an element

Arrays vs ArrayLists

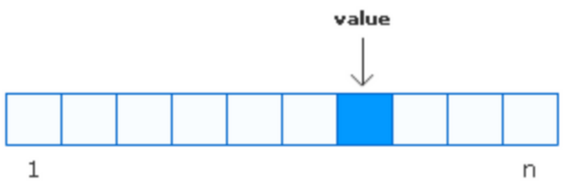
-

- An array is basic functionality provided by Java. ArrayList is part of collection framework in Java. Therefore array members are accessed using [], while ArrayList has a set of methods to access elements and modify them.
- Array is a fixed size data structure while ArrayList is not. One need not to mention the size of Arraylist while creating its object. Even if we specify some initial capacity, we can add more elements.
- Array can contain both primitive data types as well as objects of a class depending on the definition of the array. However, ArrayList only supports object entries, not the primitive data types. Note: When we do arraylist.add(1); : it converts the primitive int data type into an Integer object.
- Since ArrayList can't be created for primitive data types, members of ArrayList are always references to objects at different memory locations. Therefore in ArrayList, the actual objects are never stored at contiguous locations. References of the actual objects are stored at contiguous locations. In array, it depends whether the arrays is of primitive type or object type. In case of primitive types, actual values are contiguous locations, but in case of objects, allocation is similar to ArrayList.



Arrays vs Hashtables

- On the surface, they both allow you to perform common tasks such as adding data, accessing data, and removing data. What sets them apart is how they work behind the scenes, and this difference has a significant impact in how and when you would use them.
- In an array, if you are looking for an element you have to basically loop through all the array items to find the one with the matching value. Potentially, this could be the last item of the array.

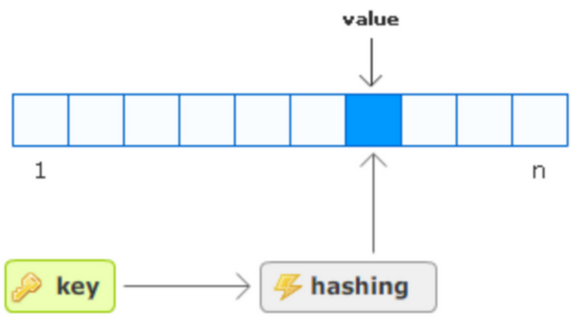


- Hashtables work by finding the exact location your value is stored and returning that value instantaneously or nearly instantaneously.
- When you add data to a hashtable, you specify both a **key** and a **value** to go with it:

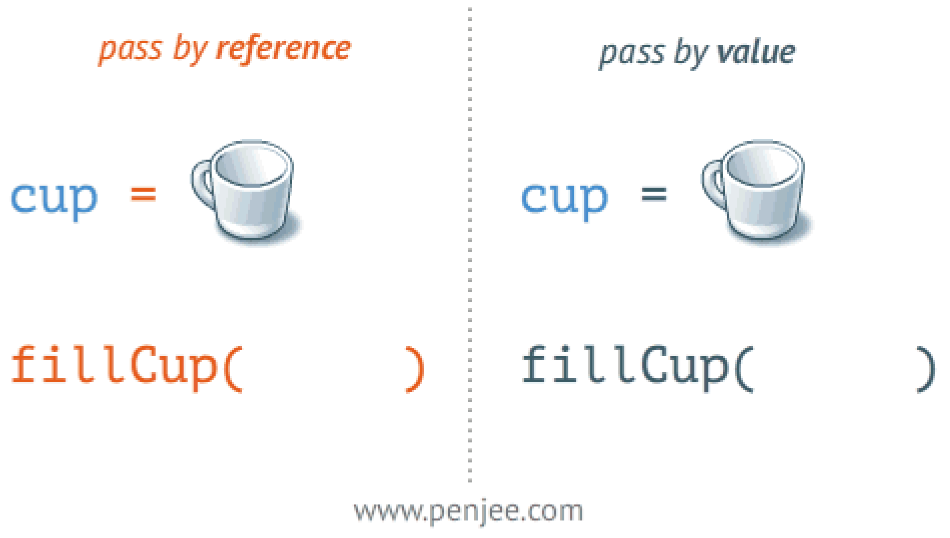
```javascript
myData["Pink Floyd"] ="110.1";
```

JavaScript

- When your hashtable sees this key, it passes it through what sounds like a painful process called **hashing**, and the output of this hashing function is a number indicating where in the hashtable our value needs to go.



- If you want to retrieve the value from the hashtable, all you have to do is provide the same key. When your hashtable sees this key, the hashing function returns ***the same number*** of the location inside the hashtable containing your value. Once you have this number, the hashtable has no difficulty retrieving the exact value.



```
cup =
fillCup(      )
```

```
cup =
fillCup(      )
```