re: discussion 9/25

compareTo, type conversions, & Map/Set methods

jump to:

compareTo

Sorry for giving that poor elaboration/example of compareTo in discussion! Hoping that this doc all about the compareTo method will help clear things up.

For more information, please see the Docs!
https://docs.oracle.com/javase/8/docs/api/java/lang/String.html#compareTo-java.lang.String-
Tutorials Point is a lil less intimidating :)
https://www.tutorialspoint.com/java/java_string_compareto.htm

Why use compareTo?

- as a tiebreaker (recall Question #1 from the 9/25 discussion)
- to specify ordering for Collections.sort, Arrays.sort, TreeSet, TreeMap, etc.c

What does it return?

- stringA.compareTo(stringB)
    - **priority for comparison: alphabetical/lexicographical** -- compareTo will go through and compare each character of each string until it finds a difference in characters (i.e. not just the first characters of each string will be compared)
        - return negative int if stringA comes before stringB alphabetically
            - e.g. stringA is “cat” and stringB is “dog”
            - e.g. stringA is “hologram” and stringB is “zoo”
            - e.g. stringA is “January” and stringB is “jump”
        - return positive int if stringA comes after stringB
        - return 0 (zero) if stringA.equals(stringB)
    - **tiebreaker: length** -- if the strings do not differ at any character up to the entire length of the shorter string, compareTo will break ties based on the strings’ lengths
        - returns negative int if stringA is longer than stringB
            - e.g. stringA is “dogma” and stringB is “dog”
        - returns positive int if stringA is shorter than string B
            - e.g. stringA is “hot” and stringB is “hottest”

What if/why would I want to make my own compareTo method?

- You may want to create your own compareTo method to define the ordering of an object you create
- All classes with a compareTo method **must implement the Comparable interface**
- Here’s a screenshotted example, based on a “Student” class I made up:

```java
class Student implements Comparable<Student>{
    // each student object will be defined by id and GPA
    int id;
    float GPA;

    // order based on GPAs
    // if GPAs are the same, return so that the student with the
        // smaller ID comes first
    public int compareTo(Student otherStudent){
        if (this.GPA != otherStudent.GPA){
            return this.GPA.compareTo(otherStudent.GPA);
        }
        return this.id.compareTo(otherStudent.id);
    }
}
```

- I now have the power to sort ArrayLists, Arrays, TreeMaps, TreeSets, etc. with my ordering, like:
    - Collections.sort(arrayListWithStudentObjects, **new Student()**);
    - Arrays.sort(arrayWithStudentObjects, **new Student()**);
    - TreeMap<Student, Integer> map = new TreeMap<Student,Integer>(**new Student()**);
    - TreeSet<Student> set = new TreeSet<Student>(**new Student()**);

type conversions

Converting between arrays, ArrayLists, and Sets was, for me, one of the greatest pains of Java. Here's a table that may help:

	to array	to ArrayList	to Set
from array	X	**For arrays containing objects (e.g. String):** ArrayList<Type> myList = new ArrayList<Type>(Arrays.asList(myArray)); ***For primitives (e.g. int):** Must loop through and add to ArrayList manually.	**For arrays containing objects:** HashSet<Type> mySet = new HashSet<Type>(Arrays.asList(myArray));
from ArrayList	Type[] myArray = myList.toArray(new Type[0]);	X	HashSet<Type> mySet = new HashSet<Type>(myList);
from Set	Type[] myArray = mySet.toArray(new Type[0]);	new ArrayList<Type>(mySet);	X

* Dealing with Object data types (e.g. String, Integer) vs. primitive data types (e.g. int) can be a challenge when you're doing conversions. Here are a few rules:

- You cannot turn an array of ints into an ArrayList or Set of Integers with a single method. (Remember that ArrayLists and Sets can only hold Object data types, so you would have to turn those ints into Integers.) Thus, you would have to loop through each int in the array of ints, and put it in the ArrayList or Set of Integers you want to create. Here's a StackOverflow post with a working example/elaboration: https://stackoverflow.com/questions/1073919/how-to-convert-int-into-listinteger-in-java
- Similarly, you cannot turn an ArrayList or Set of Integers to an array of ints, since there is no way the ArrayList or Set could have contained the primitive int values you want in the first place. If you wanted this conversion, you would have to loop through each Integer in the ArrayList or Set, and put it in an array of ints (initialized to the size of the ArrayList or Set)

- Note that Java will wrap/unwrap variables appropriately for you if you put them into an ArrayList/array one by one, e.g:

```
int[] myArray = {1, 2, 3};
ArrayList<Integer> myList = new ArrayList<Integer>();
for (int i=0; i<myArray.length; i++){
    myList.add(myArray[i]); // Java wraps int to Integer
}
```

Map/Set methods

I took the tables about important HashMap and HashSet methods from TutorialsPoint (https://www.tutorialspoint.com/java/java_hashmap_class.htm and https://www.tutorialspoint.com/java/java_hashset_class.htm), and listed the ones that I think are the most important below, as I know that the Docs can be kind of overwhelming. (Though I encourage you to try and take a look at them here: https://docs.oracle.com/javase/8/docs/api/java/util/HashMap.html and https://docs.oracle.com/javase/8/docs/api/java/util/HashSet.html)

Note: addAll was mentioned in class when we talked about HashSets but is not listed in the HashSet methods table. If you're wondering why, it's because addAll is a Collections method (meaning that both ArrayList and Sets can utilize it, yay). You can read more about addAll here, as it will definitely save you some keystrokes in the future: https://www.tutorialspoint.com/java/util/collections_addall.htm

- tl;dr thingOne.addAll(thingTwo) will add all elements of thingTwo into thingOne, and return "true" if thingOne changed as a result -- thingOne and thingTwo will probably be ArrayLists or Sets, but do not need to be of the same type

HashMap
boolean containsKey(Object key) Returns true if this map contains a mapping for the specified key.
boolean containsValue(Object value) Returns true if this map maps one or more keys to the specified value.

Set entrySet() Returns a collection view of the mappings contained in this map.
Object get(Object key) Returns the value to which the specified key is mapped in this identity hash map, or null if the map contains no mapping for this key.
boolean isEmpty() Returns true if this map contains no key-value mappings.
Set keySet() Returns a set view of the keys contained in this map.
Object put(Object key, Object value) Associates the specified value with the specified key in this map.
Object remove(Object key)

Removes the mapping for this key from this map if present.
int size() Returns the number of key-value mappings in this map.
Collection values() Returns a collection view of the values contained in this map.

HashSet
boolean add(Object o) Adds the specified element to this set if it is not already present.
boolean contains(Object o) Returns true if this set contains the specified element.
boolean isEmpty()

Returns true if this set contains no elements.
boolean remove(Object o) Removes the specified element from this set if it is present.
int size() Returns the number of elements in this set (its cardinality).