Java exercises 3

Please see the separate sheet about running Eclipse if you are not familiar with the Eclipse IDE.

Objectives

- Additional practice applying Java concepts from the slides
- Practice overriding equals(), toString(), hashCode()
- Practice with Collections objects and the Comparable interface

Task 1

Download the text file **words.txt**,[1] which contains one word per line, and save it somewhere in your file space. Your programming task is as follows: for each letter of the alphabet, your code needs to be able to find the **longest** word that begins with that letter, and the word that has the **most occurrences** of that letter. For example, for the letter b, the longest word in **words.txt** is *beautifications*; for letter a, the word that contains the most 'a' characters is *adiabatically*. If there is more than one word that is the longest or has the most occurrences, you can return any of the possible matches. The input to your method should be a lower-case letter, and you should consider upper- and lower-case letters the same when searching for matches; that is, the word *Acadia* has three occurrences of the letter 'a'.

To read the file, you can use code like the following (assuming that you have downloaded **words.txt** to the directory **/path/to/words.txt**):[2]

```
List<String> words =
          Files.readAllLines(Paths.get("/path/to/words.txt");
```

For the above line of code to work, you will need to import **java.util.List**, **java.nio.file.Files**, and **java.nio.file.Paths**, and you will need to either catch or re-throw **java.io.IOException**.

You should implement methods to implement the above two queries, as follows:

```
public String wordWithMostOccurrencesOf(char c)
public String longestWordStartingWith(char c)
```

Each of these methods should throw an **IllegalArgumentException** if it is given a parameter that is not a lower-case letter (see the static methods of the **Character** class for a way to verify this).

Task 2

Create a full class description for a brick-based building toy – call the class **BrickSet**. A **BrickSet** should have the following properties:

- A **number** (represented as an integer)
- A **name** (represented as a String)
- A **theme** (e.g., Ninjas, Space – represented as a String)
- The **number of pieces** (represented as an integer)
- A **retail price** (represented as a double)

---

[1] This file was downloaded from https://users.cs.duke.edu/~ola/ap/linuxwords.

[2] Don't forget that on Windows, you need to write two backslash characters "\\" for every "\" in the path; for example, to refer to a file in **M:\Java\words.txt** you need to write it as **"M:\\Java\\words.txt"** in the above method call.

You must also define a constructor that initialises the above five fields, a set of **get()** methods for all properties, as well as a **set()** method for the **retail price** only. Also define appropriate overridden implementations of **equals()**, **toString()**, and **hashCode()**.[3]

Your BrickSet implementation should also have one additional method, **getPricePerPiece()**, that returns a **double** value indicating the price for an individual piece.

Task 3

Using the **BrickSet** class implemented above, your task is to use this class to implement a **WishList** – that is, a list of **BrickSet** objects representing the sets that a particular person is planning to purchase. The **WishList** class will provide methods for accessing the list of sets and for adding and removing items from the list.

The internal details of the **WishList** class are up to you; the following sections describe the behaviour of the methods that you must implement. Note that, depending on your implementation, you will almost certainly end up implementing additional methods and/or properties in **BrickSet** – feel free to do so if it makes sense for your overall design, as long as you make sure to implement everything below.

Your WishList class must have a getSets() method with the following signature:

```
public Collection<BrickSet> getSets()
```

The return value of this method should consist of all **BrickSet** objects stored in the wish list. The **BrickSet** objects in the returned list should be **sorted by set number, in increasing order**. You can use the **Comparable** interface to indicate what the sorting order should be, and **Collections.sort()** or similar to implement the sorting.

Your **WishList** class should also provide two instance methods for adding and removing **BrickSet** objects from the list, as follows:

- **`public boolean`** `addSet(BrickSet set)` – if the given **BrickSet** is not already in the list, this method should add it to the list and return true. If the set is already in the list, this method should not change the list and should return false.
- **`public boolean`** `removeSet(BrickSet set)` – if the given **BrickSet** is in the list, this method should remove it from the list and return true. If the set is not in the list, the list should not be changed and the method should return false.

[3] Note that Eclipse will automatically define all of these methods for you if you want it to – right click on the class, go to **Source**, and then look at all of the menu items that begin with **Generate …** If you do this, be sure to inspect the automatically generated methods to be sure that they function properly.