COSC 1P03 Assignment 3

(Due date Oct. 31st 16:00est, Late date Nov. 3 16:00est)

Objective

To provide an implementation of an Abstract Data Type (ADT), and test it.

Background

Trying to arrange pizzas for groups of any notable size can be a monumental chore. The more people you have, the more special requests you get. Typically it ends up requiring that people make some concessions, find common ground, etc.

A *Pizza Configuration* (shortened to a **`Pizza`**) is simply the selection of which *toppings* to include. Configurations may be combined, consolidated, compared, etc., to help everyone decide what to eat!

Specifically, a Pizza constitutes an ADT with the following supported behaviours:

- You may *combine* two Pizzas
  - The result is a *new* Pizza configuration, with *all* toppings present in either (or both) Pizza
- You may *consolidate* two Pizzas
  - The result is a new Pizza, with *only* those toppings common to *both* original Pizzas
- You may exclude one Pizza from another
  - If some other Pizza is excluded (or subtracted) from the current, the result is only the *remaining* toppings unique to the original (i.e. just those that were unique to the current)
- You may ask if the current Pizza *covers* another specified Pizza
  - The Boolean result indicates if 'this' Pizza includes *every* ingredient of the other
  - This includes the case where 'this' Pizza has the entirety of the other, plus additional toppings
- You may ask if the current Pizza *is covered* (`isCovered`) by another specified Pizza
  - This is the complement to *covers*
- You may ask if the current Pizza *equals* another specified Pizza; meaning it contains precisely the same toppings
  - The result is a boolean
  - Note that sequence is immaterial. i.e. [pepperoni|olive] equals [olive|pepperoni]
- You may *add* any individual topping to the current Pizza
  - Individual toppings are listed as `String`s
  - If the topping was already present, ignore it
  - If the topping is an empty string, throw a ToppingException
- You may *remove* a requested topping from the current Pizza
  - If the topping wasn't originally present, throw a ToppingException
- You may query if a Pizza *contains* a specified topping
  - The result should be a boolean
- You may *get an array* of all toppings on this pizza (`getAllToppings`)
  - The result is a String array
- You may *represent* the Pizza as a string (`toString`)
  - The format is a square bracket ([), each topping separated by vertical bars (|), and a square bracket (])
  - Do not include a | before the first topping, or after the final topping

Include two *constructors*:

- The *default* constructor, for an 'empty' Pizza
- One to receive an array of Strings, to specify desired toppings
  - Remember that an empty string should trigger an exception
  - Duplicate toppings should simply be filtered out (i.e. included only once)

The Assignment

You have been provided with an interface, `Pizza.java`.

- This is part of the `pizzatime` package, and **you may <u>not</u> modify it at all**
    - Recall that interfaces act as specifications; as contracts. What's more, other projects rely on the same specification. Anything breaking that contract renders the entire assignment absolutely worthless.

You must write an implementation, `PizzaImpl.java`, also part of the `pizzatime` package

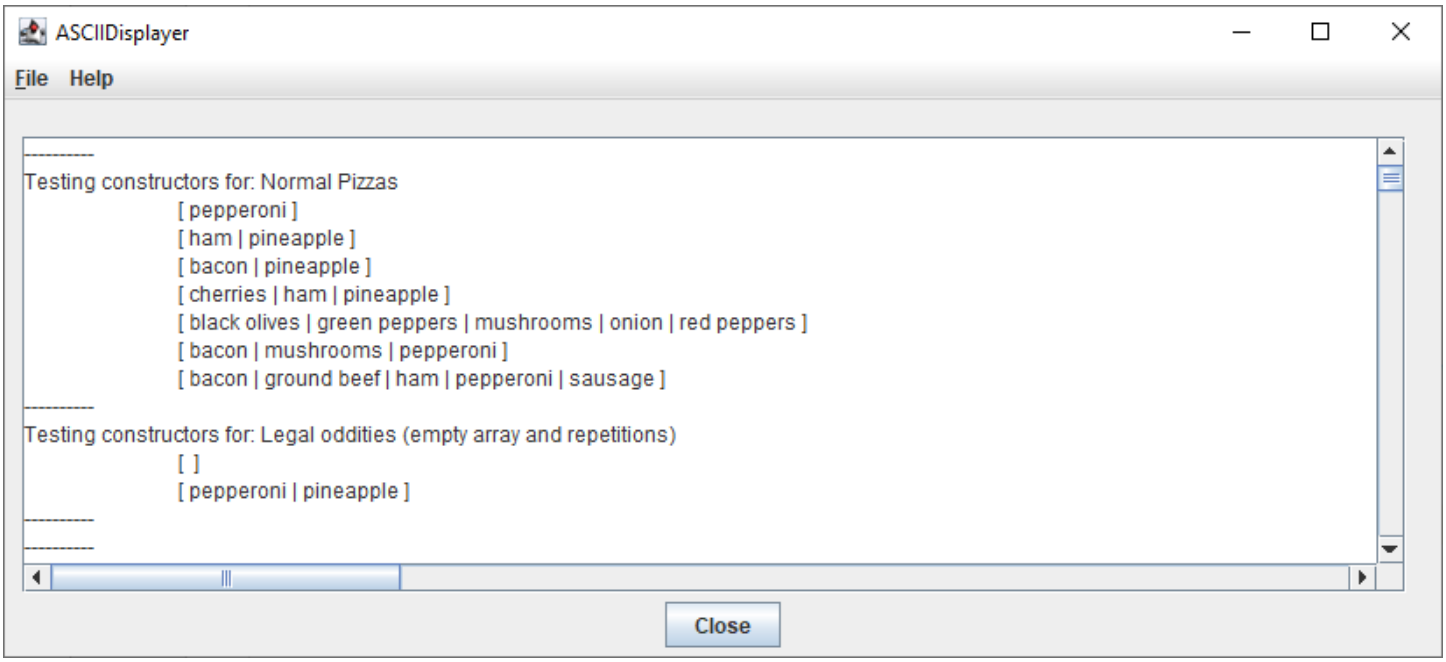
- In addition to fulfilling the promised methods in the interface, you're also providing `toString`

You must write an exception, `ToppingException.java`, as a subtype of `RuntimeException`

- There are two different types of issue to trigger these, so include a *detail message*!

In a separate package (`pizzatest`), write a basic *test harness* program that:

- Defines multiple potential Pizza configurations to test
- Demonstrates that *all* operations are completely supported
- Demonstrates that exceptions are properly thrown
    - Obviously your test program can't *crash*, so catch them, and report on them
- Tests atypical, but legal, behaviour (like trying to add the same topping multiple times)
- Writes output to both an ASCIIDisplayer *and* an ASCIIOutputFile
- Is actually thorough, and organized



An extremely simplified sample output is included as an attachment, but try to make yours better!

How to approach this assignment

First, recall the *point* of any ADT assignment: to adhere to a specification, and fulfill a contract.
This means that any code breaking the contract, in even the smallest way, is objectively **wrong**.

Beyond that:

- Though you're using two packages, you're still using a *single* IntelliJ project!
    - (Right-click src→New→package to create the second package)

- This is already included above, but again, **you may <u>not</u> modify the interface**
    - If you do, and receive a zero for the entire assignment, please to not pretend to be surprised
- It's advisable to add toppings in *sorted* sequence, simply because it makes it all easier to code
- As a friendly reminder: exceptions are how a class can communication an issue *outside of regular execution*. i.e. the exception acts as a notification to *some* earlier calling context
    - This means the ADT *throws*, and the client code (in this case, the test harness) *catches*
    - If the marker can successfully ctrl+F your PizzaImpl.java and find *catch (ToppingException* anywhere inside your concrete class… that won't end well.
- One of the two constructors accepts a String array, and the `getAllToppings()` returns one as well, but that doesn't mean you're obligated to internally represent the configuration as an array!
    - Considering the add/remove operations dynamically grow/shrink it, you might find a linked implementation simpler
- Recall that ADTs are designed to fulfill the requirements *of some other program*. As such, the Pizza configuration does not 'do the work' itself. Instead, it only responds to requests from classes outside it
    - This means it does not have a main method
    - There is to be *no* use of ASCIIOutputFile or ASCIIDisplayer anywhere within the `pizzatime` package! Similarly, you should not see a `System.out.println` anywhere!
        - The Pizza is storing data for some other class to use; if you instead output it directly, that means the class that needs it can't have it!
    - You should not even import the BasicIO library at all within the `pizzatime` package

Submission

Include in your submission, the IntelliJ project directory, zipped. Be sure your name and student number are included in your source files. Also, provide good commenting per JavaDoc standards.

- Proper commenting matters more than ever, as it's part of defining standards for usage

Assignment submission is via Sakai.