SoftEng 281: Object-Oriented Programming
Assignment 3 –The Set Calculator (10% of final grade)
Due: 9:00pm, 12th May 2021 (Week 9)

Learning outcomes

The purpose of this assignment is to target the following learning outcomes:

- Modeling *Discrete Structures*, *Sets, Relations*, and *Equivalence Classes* as object-oriented concepts.
- Practicing *unit testing and test driven development (TDD)*.
- Practicing the use of `Lists` and building confidence in defining String operations.
- Practicing the use of the Model View Controller (MVC) design pattern.

1 Introduction

Discrete structures such as **sets** are foundational for the design of any system involving a computer. Hence, they are the main basis of **discrete maths**, a discipline of maths that is essential for formalising concepts used in computer systems engineering, electrical engineering, mechatronics, and software engineering.

While a set is a collection of *unique members*, there can be a relation specifying the relationship among the members of a set. For example, consider a set containing three numbers 1, 2, and 3. Consider the "greater than" relation. Then we have a set of relation "2 is greater than 1", "3 is greater than 2", and "3 is greater than 1". As such, a relation on a set is essentially again a set. Hence, relations are sets (i.e., inheritance relationships) in object oriented programming.

The **objective** of this assignment is to develop a program called a `Set Calculator`. While the program is running, the user can open the input file, which contains the information about a set of strings and a set of relations. After extracting the information, the set calculator program checks whether the relation is `reflexive`, `symmetric`, `transitive`. Furthermore, in the case of `equivalence relation`, the program computes the `equivalence class` of a given element.

2 Background Information

In this section, we describe the background knowledge needed to complete this assignment.

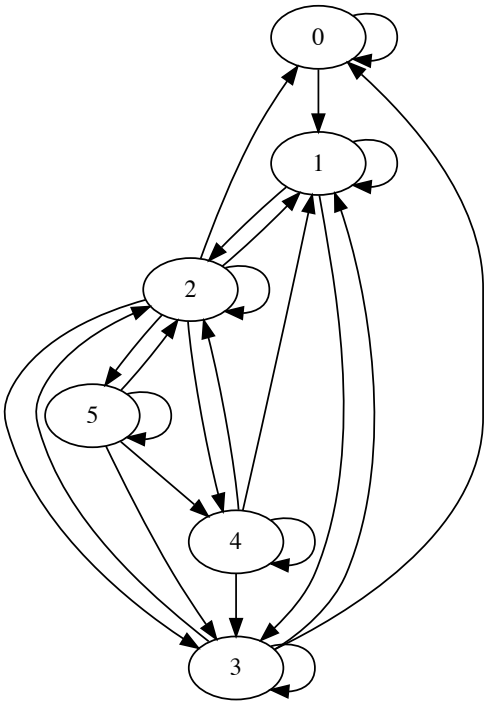
2.1 Binary Relations

A *relation*, by definition, is a subset of the product of sets i.e., $R \subseteq S_1 \times S_2 \times .. \times S_n$. Here $S_1, .., S_n$ are the participating sets and any subset of the product set is a relation, by definition. The above relation is known as an n-ary relation.

In this assignment, we will be only interested in one specific type of relation called a **binary relation**. Here, $R \subseteq S \times S$. Such relations have many practical applications, such as say in your GPS navigator. We will elaborate further on this, using the concept of **graphs**, in the following section.

2.2 Input File Format and Visualisation

In the input file, the information about the set and relation is stored as a graph. Graphs are mathematical objects of the form $< V, E >$ where V denotes a set of vertices and E denotes a set of edges. Graphs are used in multitude of applications. A very prominent one is GPS navigation, where the destinations in a city, for example, could be represented as the vertices and the edges between them represent the different paths.



(a) Graph visualization.

```
// 0, 1, 2, 3, 4, 5
digraph testgraph{
0 -> 0;
0 -> 1;
1 -> 1;
1 -> 2;
1 -> 3;
2 -> 0;
2 -> 1;
2 -> 2;
2 -> 3;
2 -> 4;
2 -> 5;
3 -> 3;
3 -> 0;
3 -> 1;
3 -> 2;
4 -> 2;
4 -> 3;
4 -> 1;
4 -> 4;
5 -> 5;
5 -> 4;
5 -> 3;
5 -> 2;
}
```

(b) Text representation.

Figure 1: An example graph of a set and relation.

In our example, we will consider simple graphs. Consider the example graph shown in Figure 1. Note that the set of vertices V can be considered as the set using which a relation may be created. In this example $V = \{0, 1, 2, 3, 4, 5\}$. The relation captures the edges of the graph. In this example the adjacency relation between vertices may be captured as a relation $R = \{(0,0), (0,1), (1,1), (1,2), (1,3), (2,0), (2,1), (2,2), (2,3), (2,4), (2,5),$
$(3,3), (3,0), (3,1), (3,2), (4,2), (4,3), (4,1), (4,4), (5,5), (5,4), (5,3), (5,2)\}$. Such a relation can be also represented using a specific text format in Figure 1b. Using a popular tool for graphs called Graphviz, a graph is visualised as in Figure 1. There is a web application for the Graphviz tool at `www.webgraphviz.com`.

The input file can be visualised for manual validation of your SetCalculator program. Try to visualize the sample input files in the TestCases folder. This can be done simply by copying the entire text in the file to the Graphviz tool.

It is important that the first line in Figure 1b must capture all the nodes in the graph. This is clear because the relations are described only using the existing nodes. However, it is possible that the input file is invalid. For this assignment, *you can assume that only the correct input files are tested during the marking process*.

There are three operations on sets that you need to implement: union, intersection, and difference.

Union: The union of two sets A_1 and A_2 is a set containing all elements that are in A_1 and in A_2. That is, $A_1 \cup A_2 = \{a : a \in A_1$ **or** $a \in A_2\}$.

Intersection: The intersection of two sets A_1 and A_2 is a set containing all elements that are both in A_1 and in A_2. That is, $A_1 \cap A_2 = \{a : a \in A_1$ **and** $a \in A_2\}$.

Difference: The difference of two sets A_1 and A_2 is a set containing all elements that are A_1 but not in A_2. That is,

$A_1 - A_2 = \{a : a \in A_1$ **and** $a \notin A_2\}$.

For this assignment we are interested in four set relations: reflexive, symmetric, transitive and equivalence.

Reflexive: A relation R on a set V is called *reflexive* if $(v, v) \in R$ for every element $(\forall)$ $v \in V$. In other words, every vertex in the graph has a self-loop. In our example in Figure 1 the relation is reflexive.

Symmetric: A relation R on a set V is called *symmetric* if $(v_1, v_2) \in R \rightarrow (v_2, v_1) \in R$ $\forall v1, v2 \in V$. In other words, If there is an edge from one vertex to another, there is an edge in the opposite direction. In our example in Figure 1 the relation is **not** symmetric. Indeed, the edge $(0, 1) \in R$ but $(1, 0) \notin R$

Transitive: A relation R on a set V is called *transitive* if whenever $(v_1, v_2) \in R$ and $(v_2, v_3) \in R$, then $(v_1, v_3) \in R$, $\forall v_1, v_2, v_3 \in R$. In our example in Figure 1 the relation is **not** transitive. Indeed, $(0, 1) \in R$, $(1, 2) \in R$ but $(0, 2) \notin R$.

Equivalence: A relation R on a set V is called an *equivalence* relation if it is reflexive, symmetric and transitive. In our example in Figure 1 the relation is **not** an equivalence relation. Indeed, it is not symmetric nor transitive.

We are also interested in t equivalence classes.

Equivalence classes: Given an equivalence relation R on a set V, the *equivalence class* of $v \in V$ is the set $\{ v_1 \in V : (v, v_1) \in R\} \subseteq V$.

3 Files provided

This program uses the architectural design pattern Model View Controller (MVC) and has the following classes: The `SetUI` class reads a file in dot format. It also has a method to read user commands over this file The entity classes are `SetOfStrings` and `StringRelation`. The `SetContrtol` is the main controller which has a `execute()` method that joins all the classes by passing appropriate messages

`src/main/java/nz/ac/auckland/softeng281/a3/SetUI.java` This is the view class in this program using MVC. The SetUI class reads a file in dot format. It also has a method to read user commands over this file. *You should not modify this class*.

`src/main/java/nz/ac/auckland/softeng281/a3/SetControl.java` . This is the controller class that has a execute() method that joins all the classes by passing appropriate messages. *You should not modify this class*.

`src/main/java/nz/ac/auckland/softeng281/a3/SetOfStrings.java` A class to create a Set of Strings and to support set operations. *You must modify this class to complete Task 1*.

`src/main/java/nz/ac/auckland/softeng281/a3/StringRelation.java` A class for binary relations over a set of strings. This class has one field `setMembers` of type `SetOfStrings` that corresponds to V. Because the class extends `SetOfStrings` it also inherits the field `elements`, which corresponds to R. *You must modify this class to complete Tasks 2, 3, 4, 5, 6*. An edge in R is represented as a String "x,y", where x and x are two set members.

The `test` folder contains three test classes:

`src/test/java/nz/ac/auckland/softeng281/a3/SetOfStringsTest.java` This Java file will contain the test cases that you implemented for completing Task 1. The file contains an example test case for the method `union()`.

`src/test/java/nz/ac/auckland/softeng281/a3/StringRelationTest.java` This Java file will contain the test cases that you implemented for completing the remaining tasks. The file contains an example test case for the method `isReflexive()`.

`src/test/java/nz/ac/auckland/softeng281/a3/TestsForMarking.java` This Java file contains half of the test cases that will be used for marking. You should not modify this Java file and you should run these tests when you think you completed the assignment to have confidence that you completed the tasks correctly.

The `testcases` folder contains some example files used by the `TestsForMarking.java` do not modify them, but you can add files if you want.

Makefile This Makefile will allow you to build and run this assignment. You can either build and run the code in your machine via Eclipse or via command line. You can also run your project in replit.com. Ultimately, once you finish your project and before you submit, you will want to **test your project using the command line in replit.com** as this is how your project will be marked. The main commands relevant to this file are:

- **make dependencies** Finds for the commands `java` and `javac`, if it does not find them it triggers an error `make: *** [dependencies] Error 1` . In such a case please follow the course help videos to properly setup your machine.
- **make clean** Removes the Java binary files (*.class) in the bin folder.
- **make build** Compiles the Java classes.
- **make run** Runs the *Set Calculator* application. In Eclipse it is equivalent to run `SetControl.java`.
- **make test-set** Compiles the Java classes and runs the SetOfStringsTest class (for TDD). In Eclipse it is equivalent to run `SetOfStringsTest.java`.
- **make test-relation** Compiles the Java classes and runs the StringRelationTest class (for TDD). In Eclipse it is equivalent to run `StringRelationTest.java`.
- **make test-marking** Compiles the Java classes and runs half of the test cases that will be used for marking. In Eclipse it is equivalent to run `TestsForMarking.java`.

4 Tasks

Before proceeding with the tasks, read carefully the code and try to run the application (`make run` or in Eclipse run the class `SetControl`). The console will ask you to insert a command

```
------------------------------------------------------------------------
Set Calculator. To know available commands, please type help
------------------------------------------------------------------------
>>
```

You need to first open a file contained in the `testcases` folder. For example, you can open the `example.txt` file which is the example in Figure 1. Then you can give the command "`list`", which will show the content of the graph.

```
------------------------------------------------------------------------
Set Calculator. To know available commands, please type help
------------------------------------------------------------------------
>>open example.txt
The command is open example.txt
The current directory is ....
The full path name is: ...testcases/example.txt
>>list
The command is list
The set elements are: {0,1,2,3,4,5}
The relational elements are: {(0,0)(0,1)(1,1)(1,2)(1,3)(2,0)(2,1)(2,2)
(2,3)(2,4)(2,5)(3,3)(3,0)(3,1)(3,2)(4,2)(4,3)(4,1)(4,4)(5,5)(5,4)(5,3)(5,2)}
>>
```

You can type "`check -r`" to check if the relation is reflexive, "`check -s`" to check if the relation is symmetric, "`check -t`" to check if the relation is transitive, and "`check -e`" to check if the relation is an equivalence relation. In case of equivalence relations you can compute the equivalence class of a node with the command "`eqclass 1`", where `1` is a node in the graph. We are assuming that only valid nodes are given in input, given "`eqclass x`" you don't need to check whether `x` is a node in the Graph. If you try the `eqclass` command you will see an exception If you try these commands you will see an exception `java.lang.UnsupportedOperationException("Not supported yet.");` this is because you need to implement the corresponding methods to complete the assignment.

5 Test Driven Development (TDD)

For this assignment we will follow the Test Driven Development (TDD) methodology. As such, you must write the unit tests before implementing the required methods. We will look into the GIT commits to check if you correctly followed the TDD methodology.

For each task, repeat the following until you complete the task.

1. write one test in the appropriate test class (`SetOfStringsTest.java` for Task 1, and `StringRelationTest.java` for Tasks 2 - 6) the test will fail because the method associated to the task is not implemented yet.
2. do a `git commit`.
3. write enough code in the appropriate class (`SetOfStrings.java` for Task 1, and `StringRelationt.java` for Tasks 2 - 6) that makes the test pass.
4. do a `git commit`.
5. repeat the following until you completed the task.
   (a) write one or more tests such that **exactly one** fails.
   (b) do a `git commit`.
   (c) write enough code in the appropriate class (`SetOfStrings.java` for Task 1, and `StringRelation.java` for Tasks 2 - 6) that make **all** the test pass.
   (d) do refactoring if needed.
   (e) do a `git commit`.
6. When all the tests pass, you finished refactoring, and you are confident the method is correct, do another `git commit`.
7. do a `git push`.
8. move to the next task (do not remove the test cases that you wrote for the current task!).

Note that

1. We encourage you to follow the TDD methodology for all the tasks. However, to get the full marks for following TDD you just need to show us that you properly followed the TDD methodology for at least one task. So, don't worry if you forgot to commit at the right moment for certain tasks.
2. After finishing the task if you want to change the code related to the finished task, no problem, this will not invalidate the TDD methodology.

You can add as many helper methods you want in `StringRelation.java` (e.g., get the first element x of the relation x, y) and `SetOfStrings.java` . We encourage you to also write unit tests for the helper methods. To do so you have to declare the helper methods `protected` otherwise the test class cannot invoke them (you cannot declare them `private`). However, you cannot change the signature of the existing methods (change name, return type or add/remove parameters).

5.1 Task 1 - Union, Intersection and Difference

Implement the methods `union`, `difference` and `intersection` in `StringOfStrings.java`. The methods return the union, difference, intersection of the current object and the given object `other`, respectively. It is important that these methods do not change the sets in input.

5.2 Task 2 - isReflexive

Write the tests and implementation for the method `isReflexive()` in `StringRelation.java` that returns `true` if the relation is reflexive, `false` otherwise.

HINT: You can implement it as follow: for each element $v \in V$ check if $(v, v) \in R$. If not, return `false`. If yes, check the next element. When you have checked all the elements (i.e., when you arrive at the end of the method) the method should return `true` because we are sure that if $(v, v) \notin R$ for any $v \in V$, the method would have terminated (returning `false`).

5.3 Task 3 - isSymmetric

Write the tests and implementation for the method `isSymmetric()` in `StringRelation.java` that returns `true` if the relation is symmetric, `false` otherwise.

HINT: You can implement it by scanning each $(v_1, v_2) \in R$ and return `false` if $(v_2, v_1) \notin R$, and return `true` at the end of the method.

5.4 Task 4 - isTransitive

Write the tests and implementation for the method `isTransitive()` in `StringRelation.java` that returns `true` if the relation is transitive, `false` otherwise.

HINT: You can implement it by scanning each pairs of edges $(v_1, v_2) \in R$ and $(v_3, v_4) \in R$ (using two nested while loops), then for each pair where $v_2 = v_3$ you check if (v_1, v_4) belongs to R or not. If $(v_1, v_4) \notin R$ you can return `false`. After checking all pairs the method returns `true`.

5.5 Task 5 - isEquivalence

Write the tests and implementation for the method `isEquivalence()` that returns `true` if the relation is an equivalence relation, `false` otherwise.

5.6 Task 6 - EquivalenceClass

Write the tests and implementation for the method computeEqClass(String node) that returns a `SetOfStrings` object containing the equivalent class of node. If the Relation is not an equivalence relation the method simply returns an empty `SetOfStrings` object.

Important: code style

In this assignment your code will **be marked** for good programming practices, you should follow standard Java naming conventions and code style. In particular:

- **method names:** Methods should be verbs, in camel case with the first letter lowercase, with the first letter of each internal word capitalised (e.g., `decideAction`).
- **variable names:** Variable names should be nouns in camel case with the first letter lowercase, and with the first letter of each internal word capitalised (e.g., `initialAmount`). Variable names should not start with underscore `_` or dollar sign `$` characters, even though both are allowed. Variable names should be short yet meaningful. One-character variable names should be avoided except for temporary "throwaway" variables (e.g., iterator `i` of a for loop).
- **correct indentation and brace placement:** You should enforce correct indentation of your Java classes. Eclipse can help you with this (Source -> Format).

- **code comments:** Comments in the code should give overviews of code and provide additional information that is not readily available in the code itself. Comments should contain only information that is relevant to reading and understanding the program. Do not comment every single line of code (!), only where necessary.
- **avoid duplicated code:** If you find yourself copy-and-pasting your code in different parts of your Java class, consider creating a `private` (`protected` if you want to write unit tests for it) method to avoid duplicated code and promote code reuse. This improves maintainability because when code is copied, bugs need to be fixed at multiple places, which is inefficient and error-prone.

Important: how your code will be marked

- Your code will be marked using a semi-automated process. If you fail to follow the setup given, your code **will not be marked**. All submitted files must compile without requiring any editing. Use the provided tests and `Makefile` to ensure your code compiles and runs without errors. Any tests that run for longer than 10 seconds will be terminated and will be recorded as failed.
- Although you may add more methods to the classes, you must leave unchanged the signature of the original methods.
- Do not move any existing code files to a new class, file, or directory.

Marking Scheme (total 10%)

- Task1 (max score 0.5%)
- Task2 (max score 2%)
- Task3 (max score 2%)
- Task4 (max score 2%)
- Task 5 (max score 0.5%)
- Task 6 (max score 1%)
- good TDD practice (max score 1%)
- good code style (max score 1%)

Frequent Submissions (GitHub)

You **must** use GitHub as version control for all assignments in this course, starting with this assignment. To get the starting code for this assignment, you must accept the GitHub Classroom invitation that will be shared with you. This will then clone the code into a **private** GitHub repository for you to use. This repository must remain private during the lifetime of the assignment. When you accept the GitHub assignment, you can clone the repository either to your own computer or work on it via replit.com. For the latter, a Repl badge/image is automatically added to the README.md file when you accept the GitHub invitation. By clicking on this, you will be able to run your code in the online replit.com IDE.

As you work on your assignment, **you must make frequent git commits**. If you only commit your final code, **it will look suspicious and you will be penalised**. In the case of this assignment, you should aim to commit your changes to GitHub every time you write new test cases or all your test cases pass. You can check your commits in your GitHub account to make sure the frequent commits are being recorded. **Using GitHub to frequently commit your changes is mandatory in this course.** In addition to teaching you to use a useful software development skill (version control with git), it will also protect you two very important ways:

1. Should something terrible happen to your laptop, or your files get corrupted, or you submitted the wrong files to Canvas, or you submitted late, etc, then you are protected as the commits in GitHub verify the progress in your assignment (i.e. what you did and when), and
2. If your final code submission looks suspiciously similar to someone else (see academic honesty section below), then GitHub provides a track record demonstrating how you progressed the assignment during that period.

Together, GitHub is there to help you.

Submission

You will submit via Canvas. **Make sure you can get your code compiled and running via command line** when running `make` in replit.com. Submit the following, in a single ZIP archive file:

- A **signed and dated Cover Sheet** stating that you worked on the assignment independently, and that it is your own work. Include your name, ID number, the date, the course and assignment number. You can generate and download this in Canvas, see the Cover Sheet entry.
- The entire contents of the **src** folder you were given at the start of the assignment, including the new code you have written for this assignment.
- Do NOT nest your zip files (i.e. do not put a zip file inside a zip file).

You must double check that you have uploaded the correct code for marking! There will be no exceptions if you accidentally submitted the wrong files, regardless of whether you can prove you did not modify them since the deadline. No exceptions. Get into the habit of downloading them again, and double-checking all is there.

Academic honesty

- The work done on this assignment must be your own work. Think carefully about any problems you come across, and try to solve them yourself before you ask anyone for help (struggling a little with it will help you learn, especially if you end up solving it). If you still need help, check on Canvas (if it is about interpreting the assignment specifications) or ask in the Lab help clinics (if you would like more personal help with Java). Under no circumstances should you take or pay for an electronic copy of someone else's work.
- **All submitted code will be checked using software similarity tools.** Submissions with suspicious similarity will result in an Investigative Meeting and will be forwarded to the Disciplinary Committee.
- Penalties for copying will be severe – to avoid being caught copying, don't do it.
- To ensure you are not identified as cheating you should follow these points:
  - Always do individual assignments by yourself.
  - Never show or give another person your code.
  - Keep your Repl workspace private, and do not share your Repl with anyone.
  - Never put your code in a public place (e.g. Reddit, public GitHub repository, forums, your website).
  - Never leave your computer unattended. You are responsible for the security of your account.
  - Ensure you always remove your USB flash drive from the computer before you log off.
  - Frequently commit your code to GitHub. This provides a track record of your work and will allow the teaching team to follow your footsteps as you completed your assignment. If you do not frequently commit your code, it will look suspicious.

Late submissions

Late submissions will incur the following penalties:

- 15% penalty for zero to 24 hours late
- 30% penalty for 25 to 48 hours late
- 100% penalty for over 48 hours late (dropbox automatically closes)