Department of Computer Science & Engineering
University of Puerto Rico
Mayagüez Campus

CIIC 4020 / ICOM 4035 - Data Structures
Spring 2022-2023
Project #1 – Cracking the Crime Code

Introduction

Welcome! You are now part of the Police Department's IT team. We know we were not your first employment choice, but we will make it a memorable one! As part of our incredible team, you will assist the best police force in the country to stop crime.
Currently our city has many criminal organizations causing trouble. These organizations plan various operations, including bank robberies, grand larceny, and money laundering. To stop these operations, we want to be able to prevent the crimes from taking place and arrest as many members of the organizations as possible. Recently, we learned that these organizations communicate their plans through hidden messages. They post flyers that seem innocent enough, with content such as poems of the week or vocabulary words, but hidden within these flyers are messages that indicate details of the operation. This is where you come in.
You will lead a project to decipher part of these messages (congrats on the promotion to project manager). Due to budget cuts your project will consist of a 1-person team. Next, we will brief you on the details of your project.

Project Design

For your project you will be tasked with deciphering part of the hidden messages, designing the strategy for making the most arrests possible, and lastly filing a report covering how many arrests we were able to make.

Part 1: Deciphering the Messages

The first objective will be for you to decipher part of the message given in the flyers. When you read them, the content seems ordinary and unremarkable, but hidden within are the orders for a criminal organization's next big offense. Some information hidden is the name of the assigned leader for the operation, time, location, and much more. But do not fret, you will only extract *part* of this information. Specifically, you will **extract the identity of the leader for the operation *and* which organization is behind the message**. The rest of the team will handle the other information since it is much harder to produce.

How? Simple. We have identified that these messages follow a format, so even though the content varies a lot, we can still process them using the same algorithm. Here is an example of a flyer in Figure 1.

#1050609
Poem of the week:
Furtive creature of the night,
On silent paws she takes her flight.
Xenial glow of moonlight guides her sight.
--
By: Animal Neo Activists Lodge

Figure 1: A Haiku about a fox.

Notice that we start with a serial number, followed by a description of the content, the content itself and the "group" that created the flyer.

#1050609
Poem of the week:
Furtive creature of the night,
On silent paws she takes her flight.
Xenial glow of moonlight guides her sight.
--
By: Animal Neo Activists Lodge

Figure 2: A Haiku about a fox. Description of format.

The information you care about is the **serial number** and **the content**. Everything else is of little use to you. The serial number will be used to identify which organization is carrying out the operation and the content will have the leader's identity.

How will you extract this information? You will need to read the file. For this job we suggest you research the `BufferedReader` and `FileReader` classes in the Java

API. You will process the file by reading it line per line, and since you know the format, you should know what to expect per line. For example, the first line of the file has the serial number.

Now, *how exactly will you decipher the message?*

Identifying the organization:

In our research we learned that the organizations identify themselves with numbers 1 through 9. Given this, the police department opted to have a **List of the organizations**, where each of these organizations' position is determined by the number – 1. **For example, if the organization identifies itself with the number 6, its position in the List is 5**.

To get this number we must calculate the **digital root (digiroot)** of the serial number.

A **digital root** is obtained by adding each individual digit of a given number until a single-digit number remains.

Example: The serial number in Figure 1 and 2 is 1050609 (notice we drop the #).
Digiroot = 1 + 0 + 5 + 0 + 6 + 0 + 9 = **21** ← This is still not a single digit, so we repeat the process
Digiroot = 2 + 1 = **3**
The **digiroot** for the number 1050609 is **3**, which means this message belongs to the organization that is at position 2.

Note: Digiroot results in a single digit between 1 and 9. This means that *at most* there will be 9 criminal organizations.

Identifying the leader:

Something else we learned is that the leader's identity is found by reading the **first character of the `ith` word in each line** of the content. Notice that we said **line, *not* sentence**. Each organization uses a different value `i`. We will call this value `key` from now on. This can be better understood with some examples.

Example 1: Lets identify the leader in the message in Figure 3, shown below. Assume that for this message `key=1`, therefore, the first character of the first word in *each line* gives us the leader's identity.

Given this we find that the first word in each line is Furtive, On, and Xenial. If we take the first character of each we get the word FOX. **Therefore, the leader for this operation is known as Fox**. The characters are highlighted for better visualization.

```
#1050609
Poem of the week:
Furtive creature of the night,
On silent paws she takes her flight.
Xenial glow of moonlight guides her sight.
--
By: Animal Neo Activists Lodge
```

Figure 3: Shows the leader’s hidden name, Fox.

Example 2: Look at the message in Figure 4. **Can you find the leader knowing that the key = 3?**

```
#1534681
Crossword puzzle #45:
V W V K N
N A I L O
I N R A O
S G G E M
S E O A R
--
Mystical Order of Nightwatchers
```

Figure 4: A message with a crossword puzzle.

If we identify the *first* character of the third (key=3) word in each line, we can build the string VIRGO. Therefore, the leader in this example is **Virgo**. Figure 5 highlights the characters.

```
#1534681
Crossword puzzle #45:
V W V K N
N A I L O
I N R A O
S G G E M
S E O A R
--
Mystical Order of Nightwatchers
```

Figure 5: Shows the leader’s hidden name, Virgo.

Note: In the case a line has less words than key, in other words, that line doesn’t have an ith word, then instead of a character you will interpret it as a whitespace.

Part 2: Arrest Operation

After we have identified the leader of the operation we are ready to arrest the members.

How will the arrests work? We must first find the leader within the organization's hierarchy. Then we follow these steps:

1. Arrest the leader. (Change their status to arrested).
2. Arrest all their underlings.
3. Identify which of the underlings have the most underlings under their supervision.
4. Go to the underlings List of the members.
5. **Repeat steps 2 - 5**, until we reach the lowest ranking member (no underlings).

For example, look at Figure 6. This is the hierarchy for the Asterism organization.

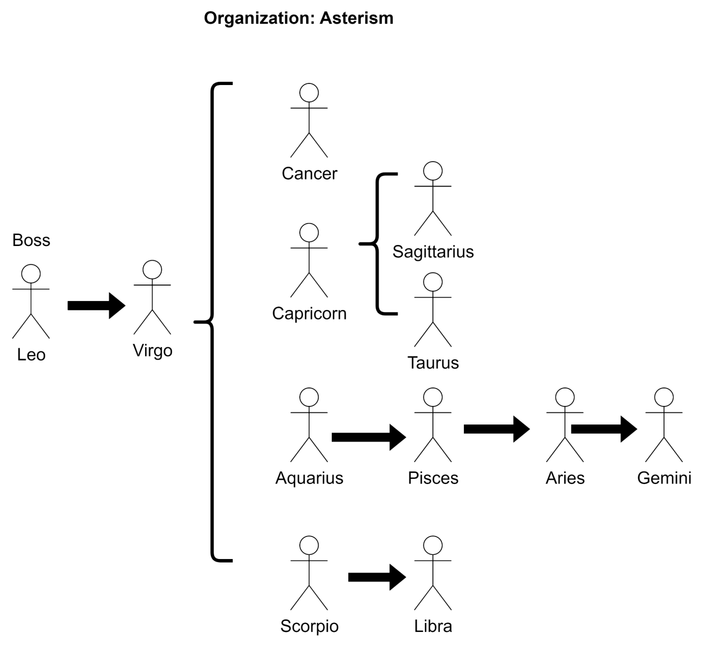

Figure 6: Organization hierarchy for Asterism.

We will now illustrate the process using Figure 7.

Let's assume that **Virgo** is the leader of the operation. We search and find them within the organization (1). We arrest them (2). We then move to their underlings and arrest all of them (3). We identify that the member with the most underlings is Capricorn, therefore we now proceed to arrest those underlings as well (4). After that we are done.

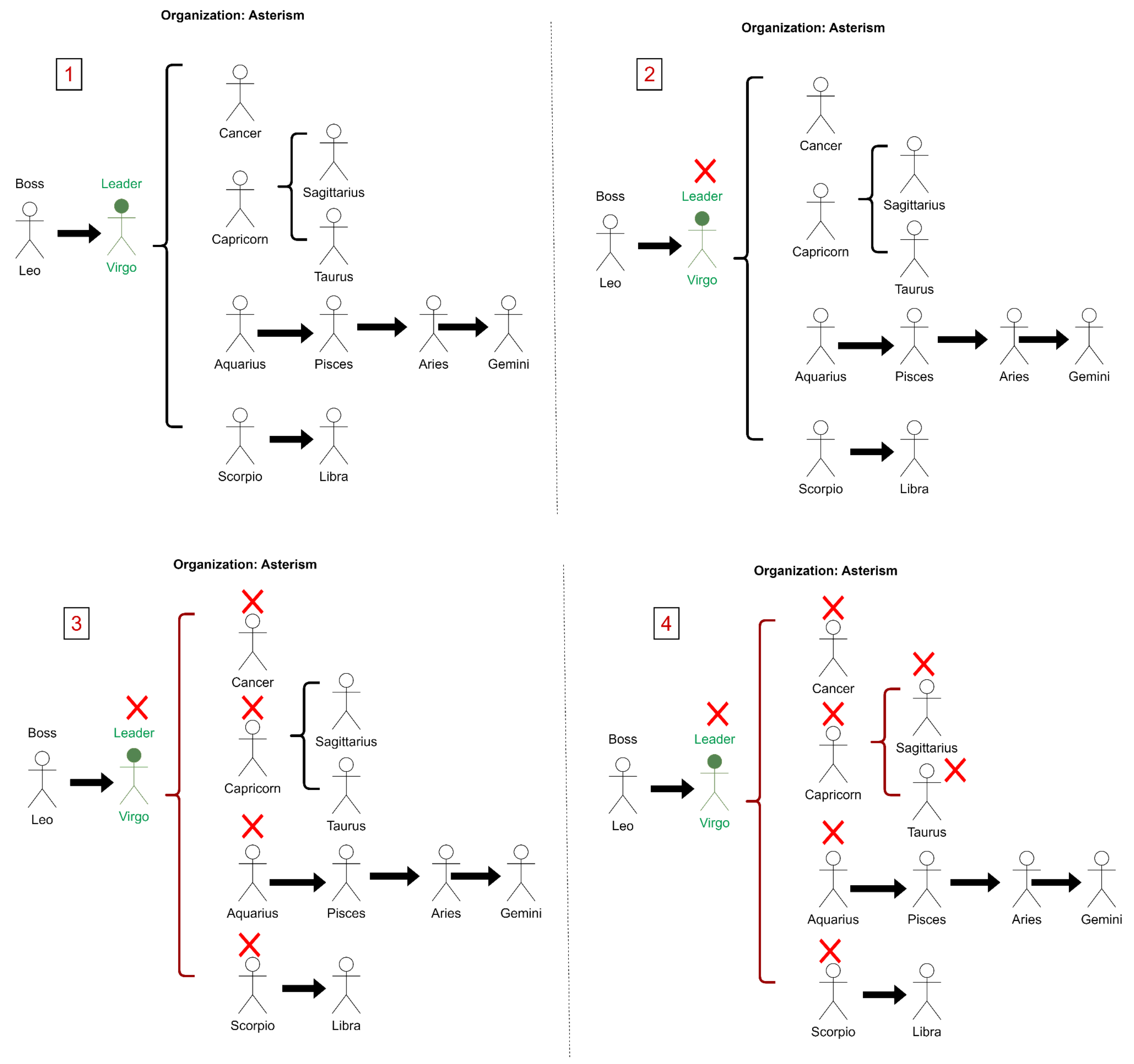

Figure 7: Illustration of arresting process.

Notice that it doesn't necessarily always get the most arrests, but it's the best we can do for now.

Part 3: Report

After processing all the messages included in the case, you will generate a report which will include the following:

- Name of the captain in charge.
- Total arrests made.
- The status of all the available criminal organizations.

Here's a sample:

```
CASE REPORT

In charge of Operation: Captain Morgan

Total arrests made: 6

Current Status of Criminal Organizations:

Organization: Atlas

Rank: 1
Name: Marcos Cesar aka Santiago THE BOSS

Rank: 2
Name: Luis Perez aka Buenos Aires | Worked under: Santiago
Name: Gabriela Mendez aka Lima | Worked under: Santiago

Rank: 3
Name: Bruce Wilde aka Washington | Worked under: Buenos Aires
(Arrested) Name: Alexander Romanov aka Paris | Worked under: Lima

Rank: 4
(Arrested) Name: Benjamin North aka Amsterdam | Worked under: Paris
Name: Nita Ford aka Rome | Worked under: Paris
Name: Tom Ibaka aka Nairobi | Worked under: Washington
---
Organization: Canin

Rank: Alpha
Name: Gabriel Holmes aka Wolf THE BOSS

Rank: Beta
(Arrested) Name: Ben Smith aka Fox | Worked under: Wolf
Name: Joe King aka Great Dane | Worked under: Wolf

Rank: Omega
(Arrested) Name: Gary Larry aka Chihuahua | Worked under: Fox
(Arrested) Name: Bary Larry aka Terrier | Worked under: Fox
(Arrested) Name: Billy Child aka Dingo | Worked under: Fox
Name: Millie Jones aka Jackal | Worked under: Great Dane
---
END OF REPORT
```

Notes:

- The file starts with the word “CASE REPORT” and ends with “END OF REPORT”.
- After each organization, we include “---” to separate it from the next organization.
- If the boss of an organization is arrested, it should say “DISSOLVED”, right before the organization's name. Look at the sample report below as an example. It assumes the leader of Canin, from the previous report, was arrested.
- The report should be stored in a `results` directory available in the project and the name of the report will be [caseFolder name]Report. Where [caseFolder name] is replaced by the name of the directory of the case being processed. (Look at the Input Data section). For example, if the caseFolder is called `case1`, then the report should be called `case1Report`.

```
…
Rank: 4
(Arrested) Name: Benjamin North aka Amsterdam | Worked under: Paris
Name: Nita Ford aka Rome | Worked under: Paris
Name: Tom Ibaka aka Nairobi | Worked under: Washington
---
DISSOLVED
Organization: Canin

Rank: Alpha
(Arrested) Name: Gabriel Holmes aka Wolf THE BOSS
…
```

Input Data

The input for this project can be found in the directory called `inputFiles`. There you will find directories associated with different cases. Each one of those case folders will have 2 directories inside: `CriminalOrganizations` and `Flyers`. The `CriminalOrganizations` directory has text files with information about each organization. The format of the files found here aren’t that relevant since the `Organization` class will handle them for you. The `Flyers` directory is where all the messages we have obtained are stored. The format of these files are as was described in Part 1.

```
inputFiles
|
|_ _ _CaseFolder
```

```
          |
          |_ _ _CriminalOrganizations
          |
          |_ _ _Flyers
```

Sample Data and Tester

You will be provided with a sample input file and a JUnit test so that you can test your implementation. Remember that one successful test case does ***NOT*** guarantee that your program is correct, but it should at least help you to find some of the most common errors. You should create your own input files and testers to verify other edge cases you feel the provided input file and tester are not covering.

Classes

Do not worry, you will not have to implement this project from scratch. We will give you a Java project with all the classes that you will need. Some of the classes will already be implemented (the difficult ones), and the empty ones are for you to fill. The structure is as follows:

```
src
|
|_ _ _criminals (Don't touch anything here)
|     |_ _ _Member.java
|     |_ _ _Organization.java
|
|_ _ _interfaces (Don't touch anything here)
|     |_ _ _List.java
|     |_ _ _MemberLambda.java
|
|_ _ _lists (Don't touch anything here)
|     |_ _ _ArrayList.java
|     |_ _ _DoublyLinkedList.java
|     |_ _ _SinglyLinkedList.java
|
|_ _ _police (Your job)
|     |_ _ _ArrestOperation.java
|     |_ _ _PoliceDepartment.java
|
|_ _ _tests (DON'T TOUCH ANYTHING HERE!!!)
      |
      |_ _ _PDTester.java
```

Your job is to finish the classes in the police package.

PoliceDepartment class

This class will hold the following information:

- Name of the captain of the Police Department.
- List with all the criminal organizations.

You are free to add more variables if you deem it necessary.

You will also implement the following methods:

```java
/**
* Creates an instance of PoliceDepartment. It receives the name of the
* Captain as a parameter.
*
* @param captain - (String) Name of the captain of the Police
* Department
*/
public PoliceDepartment(String captain);
/**
*  Returns the List of Criminal Organizations that the Police Department
*  has on record.
*/
public List<Organization> getCriminalOrganizations() {
/**
* Does the setup of the criminal organizations List.
*
* This method will read each organization file in the Criminal Organization
* folder and and add each one to the criminal organization List.
*
* NOTE: The order the files are read is important.
* The files should be read in alphabetical order.
*
* @param caseFolder - (String) Path to the folder where we can find the case
* files.
* @throws IOException
*/
public setUpOrganizations (String caseFolder);
/**
* Receives the path to the message and deciphers the name of the leader of the
* operation. This also identifies the index of the current organization we are
* processing.
* @param filePath - (String) Path to the flyer that has the hidden message.
* @return
* @throws IOException
*/
public String decipherMessage(String caseFolder);
/**
* Calculates the digital root (digiroot) of the number received. The number is
* received as a String since it makes processing each individual number a bit *
easier.
*
* @param numbers
* @return
*/
public int getDigiroot(String numbers);
/**
* Does the arrest operation by first finding the leader within its given
* organization. Then using that as a starting point arrest the most members
* possible.
*
* The idea is to arrest the leader and then arrest their underlings.
* Afterwards you identify which of the underlings has the most underlings
*under them and then move to arrest those. You will repeat this process until
*there are no more underlings to arrest.
*
* Notice that this process is pretty recursive...
*
* @param leader - (String) Identity of the leader of the criminal
* operation.
*/
public void arrest(String leader);
/**
* Generates the police report detailing how many arrests were achieved and how
* the organizations ended up afterwards.
*
* @param filePath
* @throws IOException
*/
public void policeReport(String filePath);
```

ArrestOperation Class

This class uses a `PoliceDepartment` object to follow all the steps explained in parts 1 - 3. This class will **consist of a `main()` method**, and you can add as many private methods as you deem necessary.

The `main()` will do the following:

- It will establish in which case you are currently working. In other words, the **path to where the input files for the case are (the caseFolder)**.
- It will create a PoliceDepartment object. The captain for the department will be Captain Morgan.
- Using that PoliceDepartment object, you will:
    1. Setup the Organization List for the PoliceDepartment.
    2. Iterate through the files in the `Flyer` folder. For each message you will:
        - Decipher the organization and leader.
        - Arrest the people involved.
    3. Generate the report with the arrest summary.

Member Class

This class has all the information related to members of criminal organizations. It will hold the following information:

- Name of the member
- Nickname of the person within the organization. This is the main identifier.
- List of underlings. This List contains all the members that are under the supervision of this person.
- The member's rank within the organization.
- The nickname of the member that supervises them.
- Whether this member is arrested or not. Initially this is false.

Organization Class

This class has all the information related to a criminal organization. It will hold the following information:

- Name of the organization
- Boss of the organization (Member object)
- Key for deciphering the leader.

Some methods of relevant information:

- `setupOrganization()`: This method receives the location of a file with the organization information and proceeds to setup the organization.
- `organizationTraversal()`: This method moves through the organization hierarchy and returns a sub-list with all the members that comply with the lambda function received as a parameter. The lambda works like a filter, so anything that complies with the function gets added to the sub List.

Note: You need to take a good look at the Organization class. You'll notice they *don't* have a List of members instead we only have access to the boss. To get to the other members we have to **move through the underlings Lists** of each member.

Hint: Look at the method `organizationTraversal()`, it should give you an idea of what to do, as well as be helpful for finding the leader.

Note: *You must use only locally defined data structures; you cannot use Java's built-in List structures*. However, you can be creative as to how you use them for convenience purposes when designing your algorithms.

Documentation & Comments

You **must properly document your code using the Javadoc format**. Include at the top of the class and methods some comments explaining the structure of your class, particularly **what** type of list(s) you are using and **why**. For example, *"An ArrayList was used to store the Organizations since…"*. Different people will have different implementations, so it's important that your code is well documented and commented so that we may understand your intention. **You will lose points** if you don't provide comments, if we deem they are inadequate, or that they do not follow the Javadoc format.

Submission

The final date to submit your program will be **Wednesday, March 22, 2022** at 11:59 PM. You will upload your code to your corresponding GitHub Classroom repository and it must follow the format explained in this document. We will evaluate your project with the latest commit before the deadline. Any code pushed to the repository after the cutoff date, as well any code that is not in the GitHub Classroom repository (e.g. via e-mail), will not be considered for grading.

Academic Integrity

Do NOT share your code! You may discuss design/implementation strategies, but *if we find projects that are too similar for it to be a coincidence, all parties involved will receive a grade of 0*. Don't cheat yourself out of a learning experience; seek our help if you need it.

Final Comments

The specifications of a project are the first, and arguably the most important, part of a software development project. Therefore, it's crucial that you read these specifications thoroughly so that you *understand* what is being asked of you. These are skills that you will need to succeed in your professional career, so it's imperative that you start applying and improving them now. If your program runs successfully, but does not adhere to the specifications, it is of no use. Before you submit your project, review these specifications one last time and make sure you meet all of the requirements that have been imposed.

If your code does not compile properly, your grade will be 0. NO EXCEPTIONS!