# PROJECT 1 DATA STRUCTURES

CIIC 4020 - 030

Luis M. Cintron Zayas
Luis.cintron16@upr.edu
841-14-1275
CIIC 4020-030

Luis M.Cintron Zayas
841 14 1275

## Contents

## Execution of Program

The program can be executed from the terminal(linux/unix) or the command prompt. To do so make sure you are in the correct directory path as show in the documentation provided by the professor. If the program will be executed from an IDE, then head to p1MainClasses and run Part1Main or Part2Main.

### Part1Main

To execute Part1Main from the terminal or command prompt, head to the correct directory and write the following command. Part1Main reads all the files that were generated by the file generator main and finds union and intersection of each of the sets.

```
java -classpath ./bin  p1MainClasses/Part1Main n
```

Where n Specifies the Solutions:

n = 1 ----> Executes P1's solution

n = 2 ----> Executes P2's solution

n = 3 ----> Executes P3's solution

n = 4 ----> Executes P4's solution

If no n or an invalid n is provided, then the program will execute all solutions.

Luis M.Cintron Zayas
841 14 1275

## Part2Main

To execute Part2Main from the terminal, head to the correct directory and write the following command

```
java -classpath ./bin  p1MainClasses/Part2Main n  m isize  fsize  istep  rep
```

The following parameters signify how the data will be generated:

n-----> Number of Companies

m-----> Number of Crime Events

isize -----> Initial size for the data that will be generated

fsize ------> The final size of the generated data

isetp -------> Incremental size Step, how the much the size of data will increase per trial

rep --------> The number of repetitions per size to find the average time of each implementation per size

## Part1Explination

This part contains a working version of the four set intersection implementations (P1,P2,P3,P4)

To further test each of these classes a P1tester.Class was added to the testerClasses package. Here each solution was tested with

smaller parameters. To properly test if all solutions where properly implemented.

## Part2 Explanation

This part generates all results for the execution times of each one of the four strategies we provided

and adds them to a text file.  "experimentalResults/allResults.txt".

Luis M.Cintron Zayas
841 14 1275

# Code Overview

Here the name of each class that was created for this project and it what package they are will be provided. To completely understand the hierarchy of the classes please look at the uml design "Luis M.CintronZayas CIIC4020-030 -UMLDesign.ucls"

## dataGenerator

----> DataGenerator.java (This Class generates that data that will be added to the “inputFiles” folder

----> DataReader.java (This class is used to read all the files that were generated.

## ExperimentalClasses

----> ExperimentalController.java (This class is important for part two because it is the one that runs all the provided solutions and saves the results)

----> StrategiesTimeCollection.java (This class is used to run the trails of each solution and collects the time.

## Interfaces

---->IntersectionFinder.java (Provides all the method that will be used in the solutions of each programmer)

---->MySet.java (Provides the methods that each of the implemented sets will used)

## mySetImplementation

---->AbstractMySet.java (Provides the methods that are general to each of the sets that where implemented and add abstract methods

that will have to be implemented different depending on the set.)

---->Set1.java (Implementation of mySet using ArrayList)

---->Set2.java (Implementation of mySet using HashMaps)

Luis M.Cintron Zayas
841 14 1275

## p1MainClasses

---->FilesGeneratorMain.java (This main class creates all the files using the DataGenerator class of the dataGenerator package)

---->Part1Main.java (Provides the implementation of all solutions as stated above)

---->Part2Main.java (Collects the times of each trial of all solutions as stated above)

## setIntersectionFinder

---->AbstractIntersectionFinder.java (This class provides the constructor and getName method that will be used by each of the 4 solutions

## Solutions

----->P1P2.java (Provides an implementation of intersect sets of P1 and P2 where P1 uses Set1 for his implementation and P2 uses Set2 for his, they both have generally the same idea but there set implementations they used are different from each other.

----->P3.java (Provides an implement of P3's solution where all elements are put in a sorted ArrayList and the elements that are repeated m times are added to a set of type Set2

----->P4.java (Provides an implement of P4's solution where all elements are put in a ArrayList and then those elements are added to a hash map with the value of how many times they appear, after that the elements that have a value equal to m are added to a set of type Set2.)

## TesterClases

-----> DataGeneratorTester.java (Tests if the data is generated properly and prints the sets and their sizes)

----->DataReaderTester.java(Tests if the data if being properly read from each file in the inputFiles folder of F_i_j)

----->P1tester.java (Testes all solutions (P1,P2,P3,P4) with smaller values)

Luis M.Cintron Zayas
841 14 1275

## Tester Outputs

### P1Tester

This tester class was added due to the fact that Part1Main uses very large files and this way it was simpler to not only test each set but also to do the intersections manually and check if the output is correct. This is an example of the output of that tester.

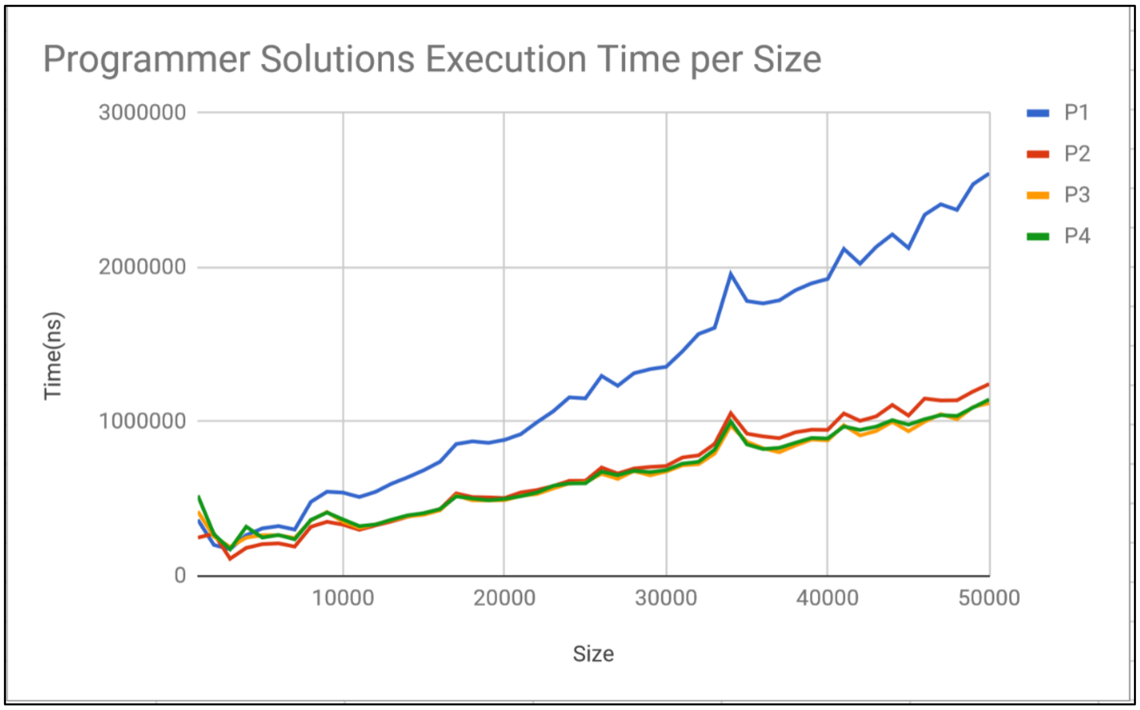
```
n = 3  m = 5

The Sets for testing are:

Sets are:
Set[0][0] = 4  14
Set[0][1] = 14
Set[0][2] = 0  1  2  8  14
Set[0][3] = 0  1  2  3  5  6  7  8  9  10  11  12  13  14
Set[0][4] = 1  2  3  4  6  7  8  9  10  11  12  14
Set[1][0] = 2  4
Set[1][1] = 0  1  2  3  4  5  6  7  8  9  10  11  12  13
Set[1][2] = 0  1  2  4  5  7  8  10  12  13
Set[1][3] = 1  4  5  7  11
Set[1][4] = 1  3  4  5  7  8  9  10  12  13
Set[2][0] =
Set[2][1] = 0  3  7  11
Set[2][2] = 0  2  3  5  6  7  8  10  11  12  13
Set[2][3] = 1  3  4  6  7  8  9  11  12
Set[2][4] = 1

The Solutions for the intersections are:

P1 solution: {4, 14, 2}
P2 solution: {2, 4, 14}
P3 solution: {2, 4, 14}
P4 solution: {2, 4, 14}
```

Luis M.Cintron Zayas
841 14 1275

## Part2Main

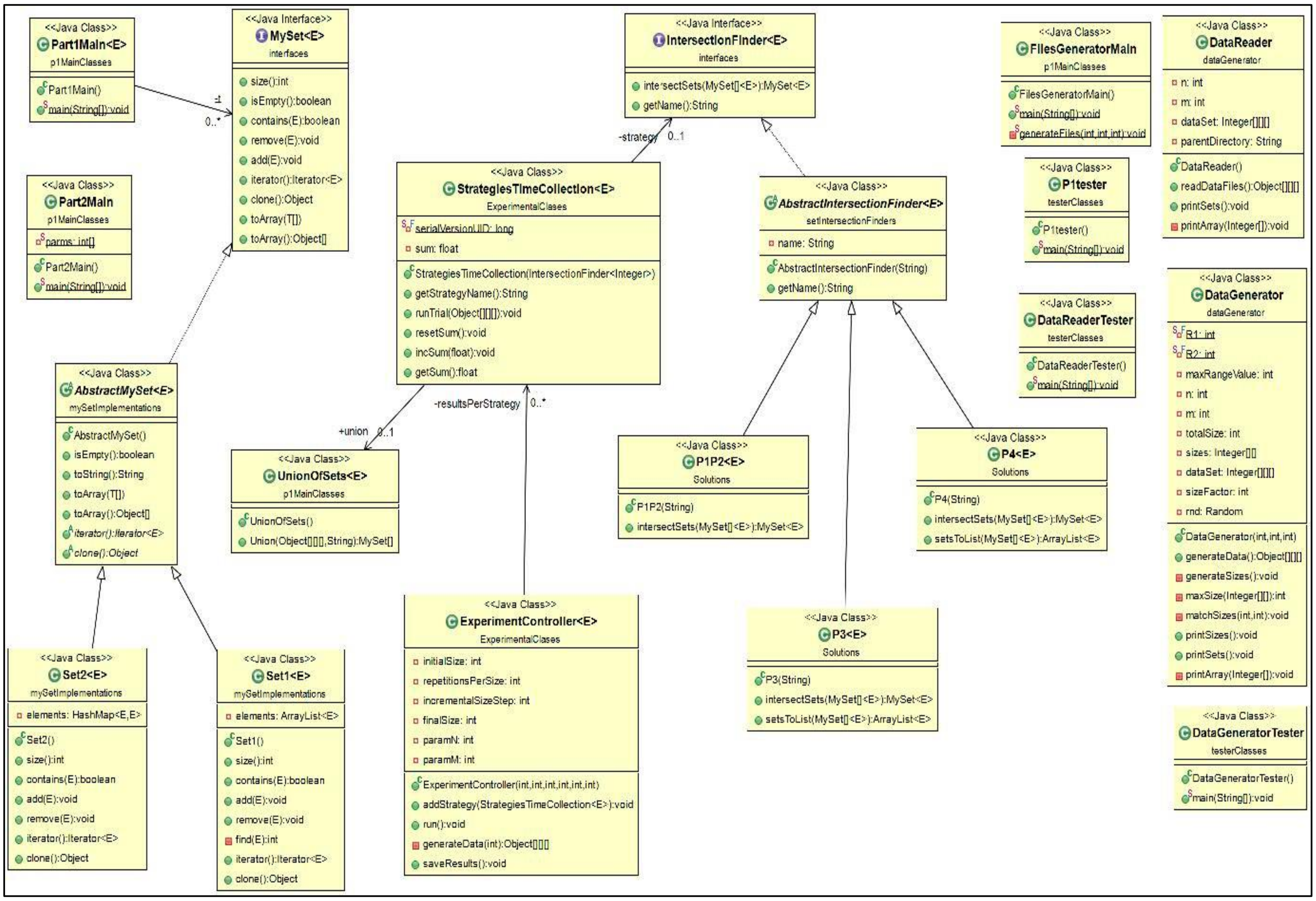
This is the output of Part 2 Main. What will be shown here is a table with possible execution times, depending on the hardware the program is being ran.

| Size | P1 | P2 | P3 | P4 |
|---|---|---|---|---|
| **1000** | 365183.75 | 248761.22 | 419348.2 | 521980.12 |
| **2000** | 201575.5 | 274670.88 | 255555.06 | 270851.47 |
| **3000** | 174184.69 | 112297.51 | 188291.52 | 173827.8 |
| **4000** | 265624.88 | 182592.94 | 249066.92 | 319807.47 |
| **5000** | 308913 | 206937.69 | 264915.12 | 249464.88 |
| **6000** | 324254.6 | 212000.19 | 264547.94 | 266277.22 |
| **7000** | 301050.53 | 191175.55 | 246884.23 | 238246.4 |
| **8000** | 480630.03 | 319263.8 | 359891.47 | 364184.9 |
| **9000** | 546761.5 | 351983.88 | 415023.9 | 412006.53 |
| **10000** | 540699.9 | 333280.2 | 351181.8 | 366691.47 |
| **11000** | 512318.44 | 299951.1 | 312346.94 | 323815.72 |
| **12000** | 545661.9 | 328223.9 | 334447.47 | 333877.1 |
| **13000** | 598905.06 | 354226 | 362299.84 | 365680.25 |
| **14000** | 640236.4 | 384769.56 | 386484.38 | 393739.7 |
| **15000** | 686468.2 | 406742.88 | 397711.2 | 407788.97 |
| **….** | …….. | ……. | ……. | …….. |



Luis M.Cintron Zayas
841 14 1275

## UML Design



<<Java Class>>
Part1Main<E>
p1MainClasses
Part1Main()
main(String[]):void
<<Java Interface>>
MySet<E>
interfaces
size():int
isEmpty():boolean
contains(E):boolean
remove(E):void
add(E):void
iterator():Iterator<E>
clone():Object
toArray(T[])
toArray():Object[]
0..*
<<Java Class>>
Part2Main
p1MainClasses
parms: int[]
Part2Main()
main(String[]):void
<<Java Interface>>
IntersectionFinder<E>
interfaces
intersectSets(MySet[]<E>):MySet<E>
getName():String
-strategy
0..1
<<Java Class>>
StrategiesTimeCollection<E>
ExperimentalClases
serialVersionUID: long
sum: float
StrategiesTimeCollection(IntersectionFinder<Integer>)
getStrategyName():String
runTrial(Object[][][]):void
resetSum():void
incSum(float):void
getSum():float
<<Java Class>>
AbstractIntersectionFinder<E>
setIntersectionFinders
name: String
AbstractIntersectionFinder(String)
getName():String
<<Java Class>>
FilesGeneratorMain
p1MainClasses
FilesGeneratorMain()
main(String[]):void
generateFiles(int,int,int):void
<<Java Class>>
DataReader
dataGenerator
n: int
m: int
dataSet: Integer[][][]
parentDirectory: String
DataReader()
readDataFiles():Object[][][]
printSets():void
printArray(Integer[]):void
<<Java Class>>
P1tester
testerClasses
P1tester()
main(String[]):void
<<Java Class>>
DataReaderTester
testerClasses
DataReaderTester()
main(String[]):void
<<Java Class>>
DataGenerator
dataGenerator
R1: int
R2: int
maxRangeValue: int
n: int
m: int
totalSize: int
sizes: Integer[][]
dataSet: Integer[][][]
sizeFactor: int
rnd: Random
DataGenerator(int,int,int)
generateData():Object[][][]
generateSizes():void
maxSize(Integer[][]):int
matchSizes(int,int):void
printSizes():void
printSets():void
printArray(Integer[]):void
<<Java Class>>
DataGeneratorTester
testerClasses
DataGeneratorTester()
main(String[]):void
<<Java Class>>
AbstractMySet<E>
mySetImplementations
AbstractMySet()
isEmpty():boolean
toString():String
toArray(T[])
toArray():Object[]
iterator():Iterator<E>
clone():Object
-resultsPerStrategy
0..*
+union
0..1
<<Java Class>>
UnionOfSets<E>
p1MainClasses
UnionOfSets()
Union(Object[][][],String):MySet[]
<<Java Class>>
P1P2<E>
Solutions
P1P2(String)
intersectSets(MySet[]<E>):MySet<E>
<<Java Class>>
P4<E>
Solutions
P4(String)
intersectSets(MySet[]<E>):MySet<E>
setsToList(MySet[]<E>):ArrayList<E>
<<Java Class>>
P3<E>
Solutions
P3(String)
intersectSets(MySet[]<E>):MySet<E>
setsToList(MySet[]<E>):ArrayList<E>
<<Java Class>>
ExperimentController<E>
ExperimentalClases
initialSize: int
repetitionsPerSize: int
incrementalSizeStep: int
finalSize: int
paramN: int
paramM: int
ExperimentController(int,int,int,int,int,int)
addStrategy(StrategiesTimeCollection<E>):void
run():void
generateData(int):Object[][][]
saveResults():void
<<Java Class>>
Set2<E>
mySetImplementations
elements: HashMap<E,E>
Set2()
size():int
contains(E):boolean
add(E):void
remove(E):void
iterator():Iterator<E>
clone():Object
<<Java Class>>
Set1<E>
mySetImplementations
elements: ArrayList<E>
Set1()
size():int
contains(E):boolean
add(E):void
remove(E):void
find(E):int
iterator():Iterator<E>
clone():Object