This program, at its core, is used to find one or more elements that repeat a total maximum amount of times in a two-dimensional collection. If arranged in a two-dimensional plane, x being events and y being providers, each point is a list of elements that y provider has on x event. Some providers may not have list for some events, but all events are covered by at least one provider. The programs looks for the elements that appear in all events.

Part1Main finds this intersection using up to 4 different types of procedures. The procedures read the files and puts their data in a data collection Set1 or Set2 depending which procedure we take. Those collections are processed through IntersectionFinder1_2, IntersectionFinder3, or IntersectionFinder4, also depending on the procedure taken, and return the elements we want.

The Part2Main class tries to find the time it takes the four different procedures to execute and deliver for different sizes of initial element data and for certain amount of repetitions each. With that time data we should be able to graphically see the estimated time each procedure takes and how that is affected with increasing initial data, therefore finding which route is better, faster, and more efficient.

To run Part1Main you would need to run FilesGeneratorMain first to create the data files. You can do this by opening a Command window (tested with Git Bash) in the project directory **p1_40354020_172**.

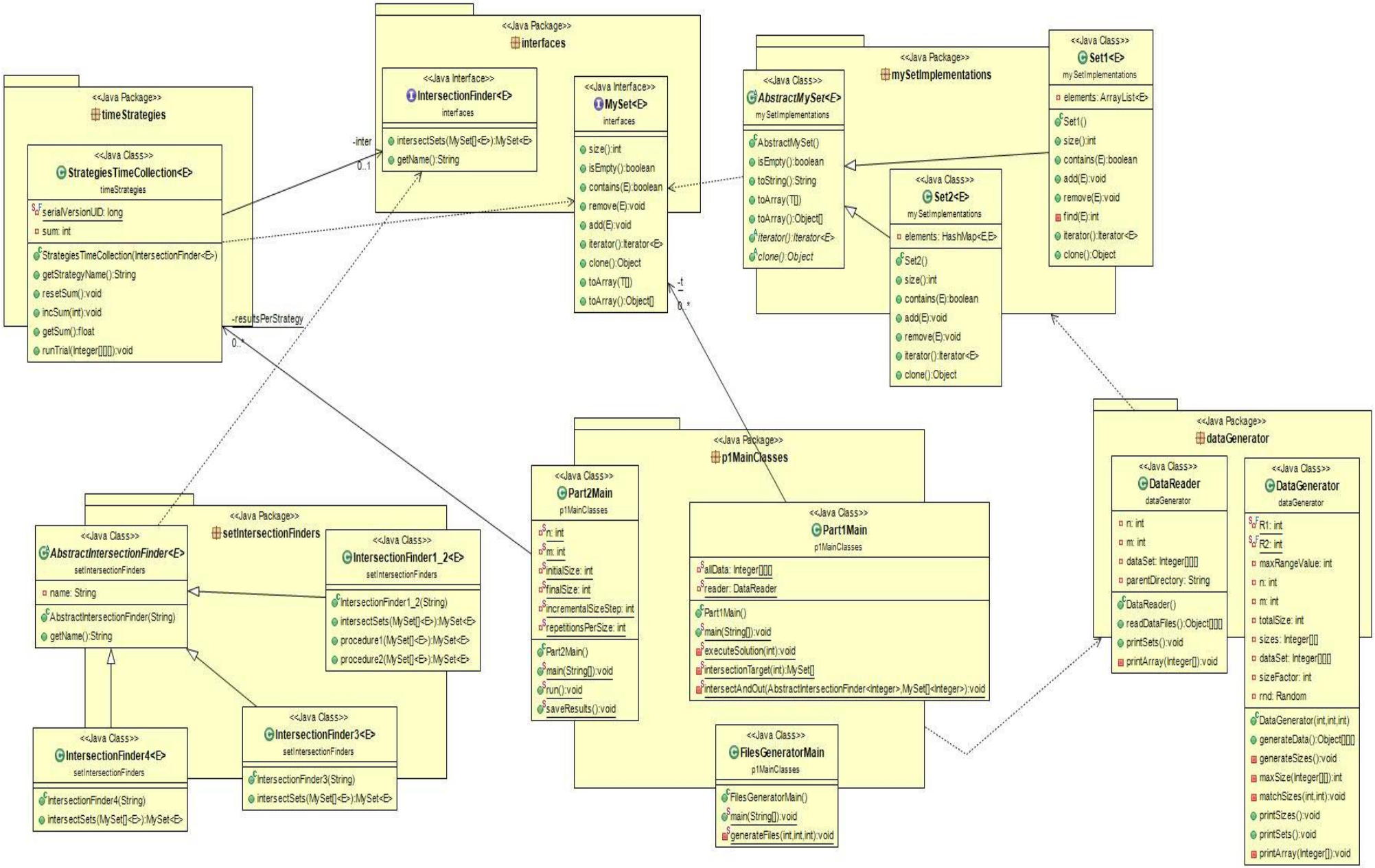
Run *java -classpath ./bin p1MainClasses/FilesGeneratorMain*	This creates the data files in folder **inputfiles**
Now run *java -classpath ./bin p1MainClasses/Part1Main*	You should see the final intersected sets in the console
You can also run *java -classpath ./bin p1MainClasses/Part1Main #*	Change # for a number from 1-4 to run only that procedure
To run Part2Main execute *java -classpath ./bin p1MainClasses/Part2Main*	The console will display your progress (it may take a while)

You can also run *java -classpath ./bin p1MainClasses/Part2Main #1 #2 #3 #4 #5 #6* where:

#1 is the amount of companies	#2 is the amount of events	#3 is the initial size
#4 is the final size	#5 is the size step or difference between sizes	#6 is the amount of repetitions

To specify one # you must also specify the ones before it (Ex. Running *10 50 500 2500* changes the first 4 values but not #5 and #6)
After it is done running, you can find the results in the folder **experimentalResults** in allResults.txt
If you want to run Part1Main again, run FilesGeneratorMain before doing so, just to make sure you have the correct files

In the project directory, you will also find a graph that shows the runtimes from allResults.txt


<<Java Package>>
timeStrategies
<<Java Class>>
StrategiesTimeCollection<E>
timeStrategies
serialVersionUID: long
sum: int
StrategiesTimeCollection(IntersectionFinder<E>)
getStrategyName():String
resetSum():void
incSum(int):void
getSum():float
runTrial(Integer[][][]):void
-inter
0..1
-resultsPerStrategy
0..*
<<Java Package>>
interfaces
<<Java Interface>>
IntersectionFinder<E>
interfaces
intersectSets(MySet[]<E>):MySet<E>
getName():String
<<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[]
-t
0..*
<<Java Package>>
mySetImplementations
<<Java Class>>
AbstractMySet<E>
mySetImplementations
AbstractMySet()
isEmpty():boolean
toString():String
toArray(T[])
toArray():Object[]
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
<<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 Package>>
setIntersectionFinders
<<Java Class>>
AbstractIntersectionFinder<E>
setIntersectionFinders
name: String
AbstractIntersectionFinder(String)
getName():String
<<Java Class>>
IntersectionFinder1_2<E>
setIntersectionFinders
IntersectionFinder1_2(String)
intersectSets(MySet[]<E>):MySet<E>
procedure1(MySet[]<E>):MySet<E>
procedure2(MySet[]<E>):MySet<E>
<<Java Class>>
IntersectionFinder4<E>
setIntersectionFinders
IntersectionFinder4(String)
intersectSets(MySet[]<E>):MySet<E>
<<Java Class>>
IntersectionFinder3<E>
setIntersectionFinders
IntersectionFinder3(String)
intersectSets(MySet[]<E>):MySet<E>
<<Java Package>>
p1MainClasses
<<Java Class>>
Part2Main
p1MainClasses
n: int
m: int
initialSize: int
finalSize: int
incrementalSizeStep: int
repetitionsPerSize: int
Part2Main()
main(String[]):void
run():void
saveResults():void
<<Java Class>>
Part1Main
p1MainClasses
allData: Integer[][][]
reader: DataReader
Part1Main()
main(String[]):void
executeSolution(int):void
intersectionTarget(int):MySet[]
intersectAndOut(AbstractIntersectionFinder<Integer>,MySet[]<Integer>):void
<<Java Class>>
FilesGeneratorMain
p1MainClasses
FilesGeneratorMain()
main(String[]):void
generateFiles(int,int,int):void
<<Java Package>>
dataGenerator
<<Java Class>>
DataReader
dataGenerator
n: int
m: int
dataSet: Integer[][][]
parentDirectory: String
DataReader()
readDataFiles():Object[][][]
printSets():void
printArray(Integer[]):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