Class Scheduling

Genetic Algorithm Problem for college timetable

Shaosheng Yin & Tianrun Gao
Group 226
2018 Fall

INFO 6205 Final Project Report

Problem Description

When designing a timetable, we need to consider many factors at same time. In this project, we are going to solve schedule problem by using genetic algorithm. We will check different scenarios at first. Then we will build a class scheduler, which can be expanded to support more complex implementations.

As mentioned above, class scheduling is a complex case, which needs to be satisfied all constraints. Violation of any one factor, the schedule is flagged with failure. Constraints has two parts, hard constraints and soft constraints.

The hard constraints:

The classroom need to be big enough to hold the class

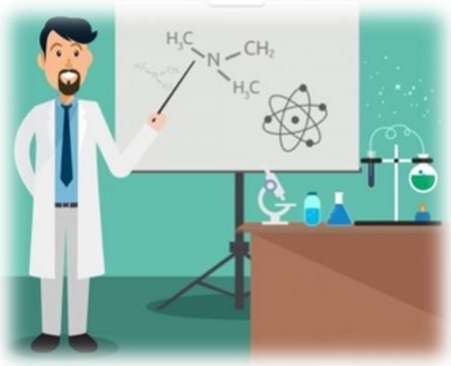
Classroom can only hold one class at any given time

Classroom must contain any required equipment

Professor can only be in one class at any given time

The soft constraints:

Room capacity should be suitable for the class size



Preferred classroom of the professor

Preferred class time of the professor

The hard constraints are the first priority, which need to be satisfied to produce a functional solution. Although the soft constraints are not necessary, the executor should also consider the cost. Like, if one classroom has 50 seats providing to ten-students class, it is not a good solution. Sometimes, multiple soft constraints may conflict, the algorithm should provide suitable solution for this case.

We will notice that class scheduling problem is little different from others algorithm problem. Our program will stop at the generation that are not violated with all hard constraints and are compatible with some soft constraints. It is because that our program is based the data we create. Once the schedule is suitable, the program will stop mutation and print result.

We could add more soft constraints in order to guarantee more accurate generation. More details will explain later.

Model Details Design

Scenario：

Each class will be assigned a timeslot, a professor, a room and a student group by the class scheduler. We can calculate the total number of classes that will need scheduling by summing the number of student groups multiplied by the number of modules each student group is enrolled in.

The schedule should consider following hard constraints and soft constraints.

hard constraints

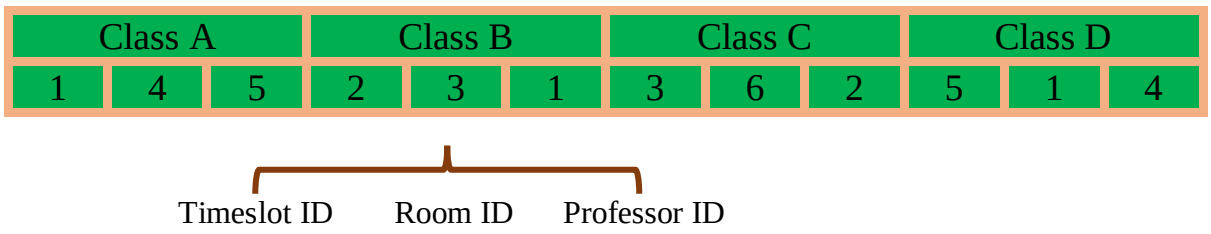
- Classroom should big enough to hold the class
- Classroom should be served one class and the class should be set in free room
- The professor can only teach one class at any given time.

soft constraints

- Preferred classroom of the professor
- Preferred class time of the professor

Chromosome Design

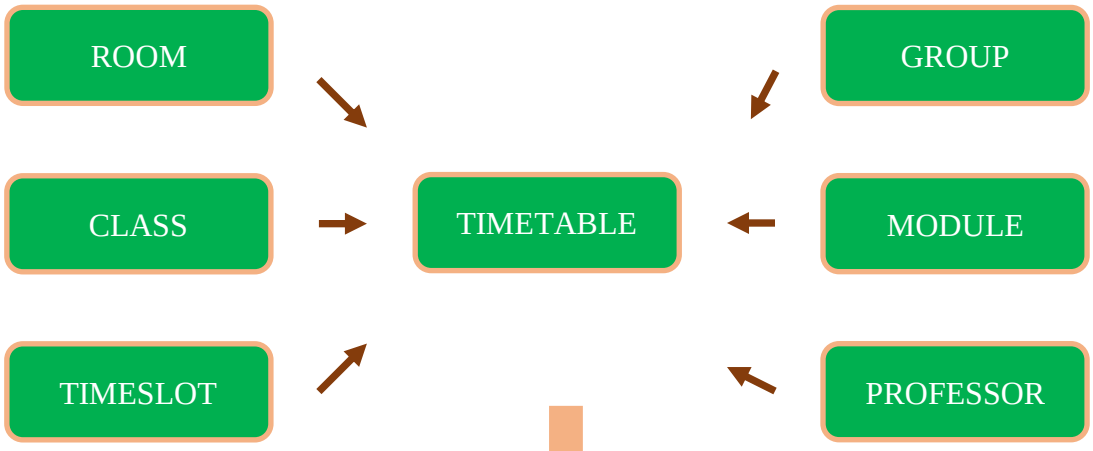
Chromosome is an important factor in the genetic algorithm where the design of chromosome will affect how the crossover implemented. We could assign several ID to each professor, classroom and timeslot. Form of gene are as below.



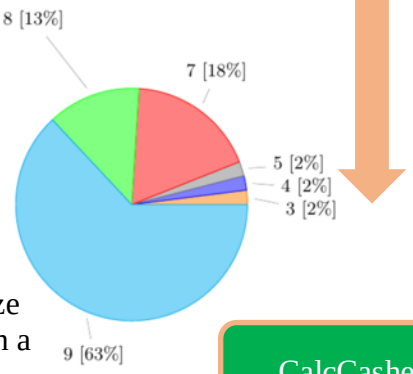
Each class has three components, 1 gene for time, 1 gene for professor, 1 gene for room.

Initialization

First of all, we have to initialize some information including class, classroom, professor, timeslot, group and module. Forms of each Java class are as below.



Evaluation

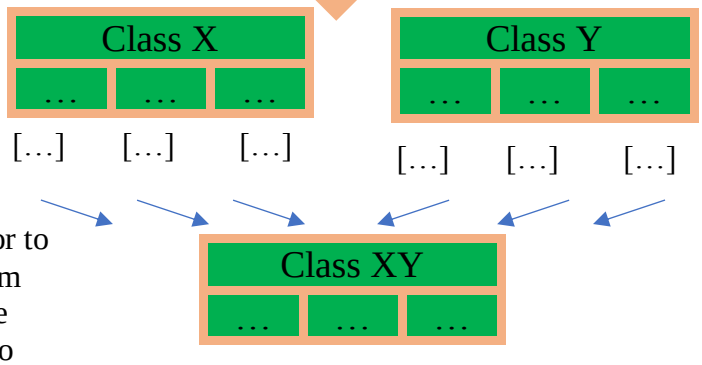


CalcCashes

If any hard constraints or soft constraints are violated, for example, if the selected room is too small, if there is a scheduling conflict for the room, the method returns the total number of clashes it found

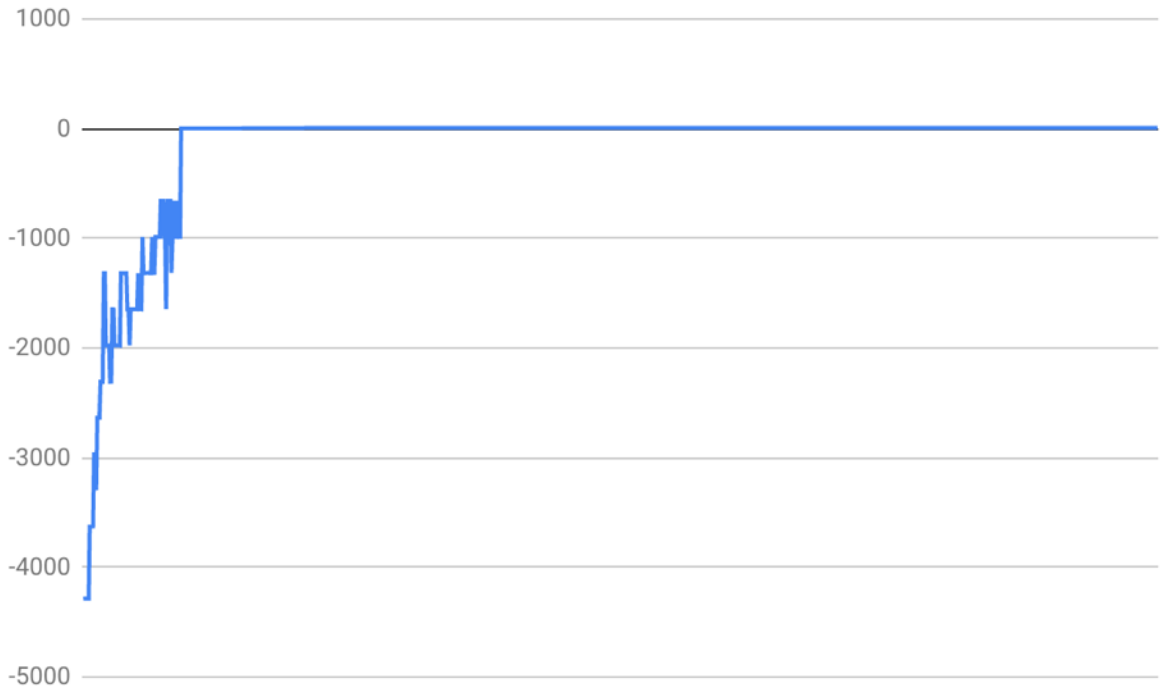
the goal is to optimize our class timetable in a way that will avoid breaking as many constraints as possible.

Mutation



Use Individual constructor to create a brand new random individual and select gene from random individual to copy into individual to be mutated

Result and Conclusion



(the first generation gets so bad fitness, which lead to the best generation that gets 8.83 nearly close to zero)

It can be predicted that the first generation could perform really bad because they have such a small amount of good genes generated by list of random number. Under such circumstance, the functional schedule could not make a right class timetable.

With generation increasing, the fitness gets better. The generation stops changing when the generation is 468. Unlike the others genetic algorithm problem, most solutions in this scenario are invalid and we stop only when we find the first valid solution or run out of time.

The difference between the two problems is as follows: in other genetic algorithm problem, it's easy to create a valid but in the class scheduler, creating a valid solution is the difficult part.

When considering only three hard constraints, it is easy to stop at nearly 200 generation. This is because the database we create is too small that means there is no very complex situations. It is enough for program to find valid results.

In order get more accurate generations, we introduce soft constraints, adding soft constraints to the class scheduler changes the problem significantly.

To guarantee a valid from the GA algorithm, we need to make sure that hard constraints have significant impact on the fitness. In other words, if the individual is not valid, the sum of points gain from match the soft constraints should less than the penalties of hard constraints which make sure that the final fitness less than 0. So, penalties of hard constraints fellow the change of population size. If population size scales up, the penalties goes up which still larger than the sum of rewards gained from matching the soft constraints.

Bonus: parallel processing

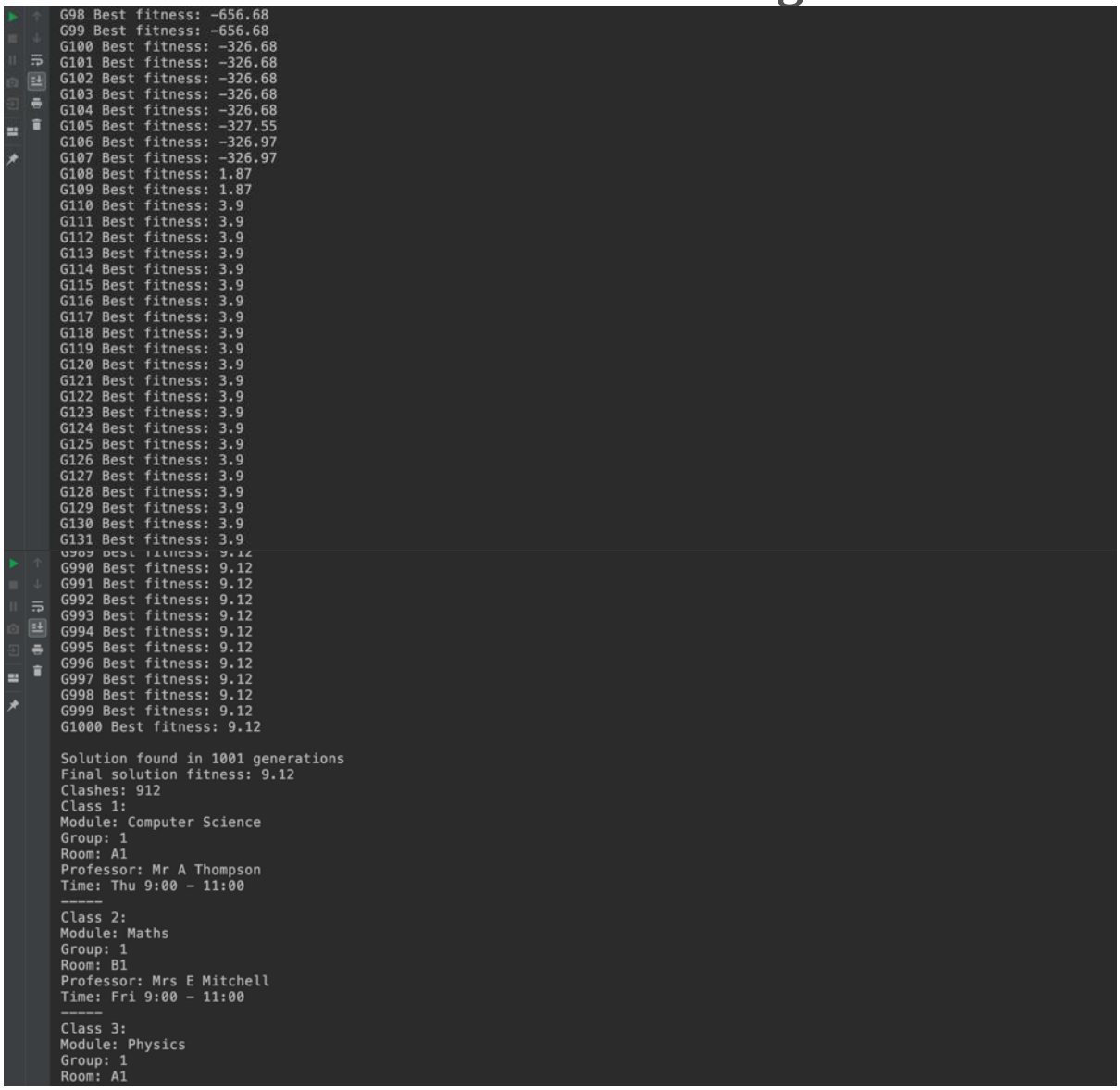
We implement parallel processing in calculating fitness. The fitness function is often going to be the bottleneck of genetic algorithm. Using parallel processing can calculate the fitness of numerous individuals simultaneously, which makes a huge difference when there are often hundreds of individuals to evaluate per population.

We use IntStream () to achieve parallel processing in fitness function. To ensure the objects in thread will not affect objects in another thread, we clone the timetable object before calculating the individuals' fitness.

```
IntStream.range(0, population.size()).parallel()
        .forEach(i ->
this.calcFitness(population.getIndividual(i),
                timetable));
```

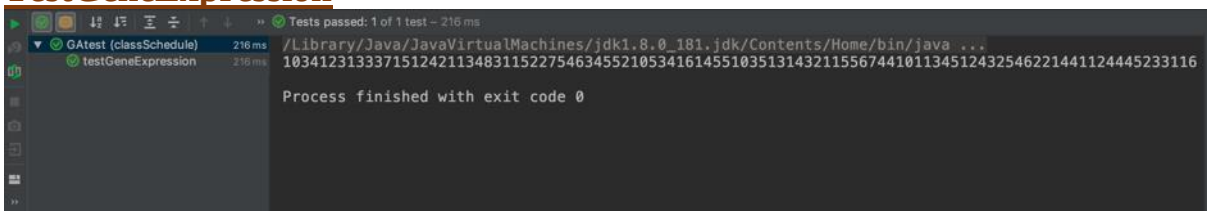
Range from 0 to the size of population, parallel() returns an equivalent stream that is parallel. In the forEach(), we give the action-calculate fitness to it.

Evidence of Running



Unit Tests

TestGeneExpression

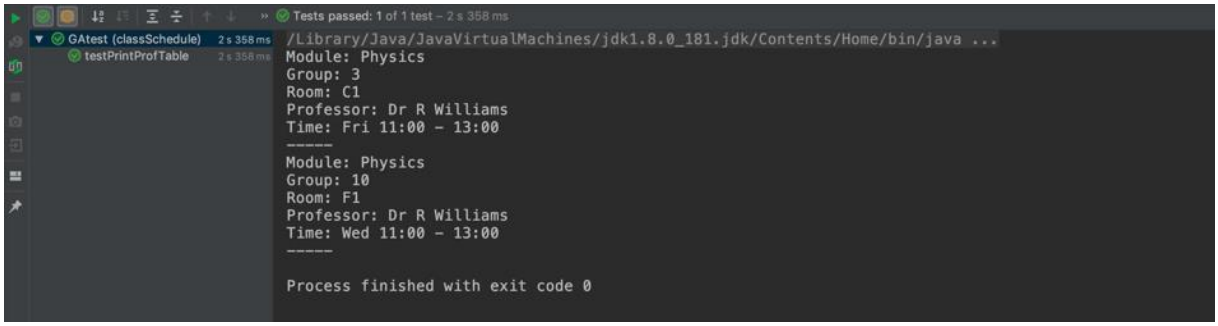


TestEval

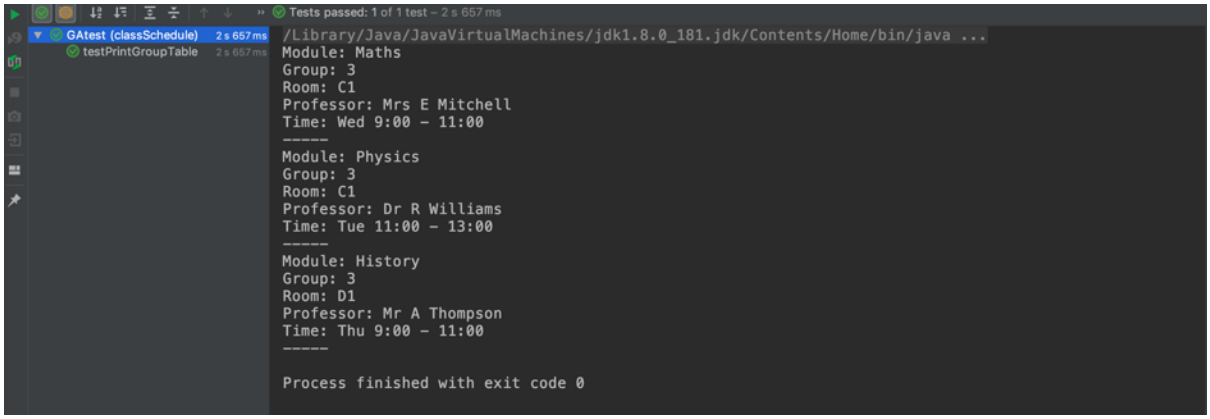
TestMutate

TestCrossover

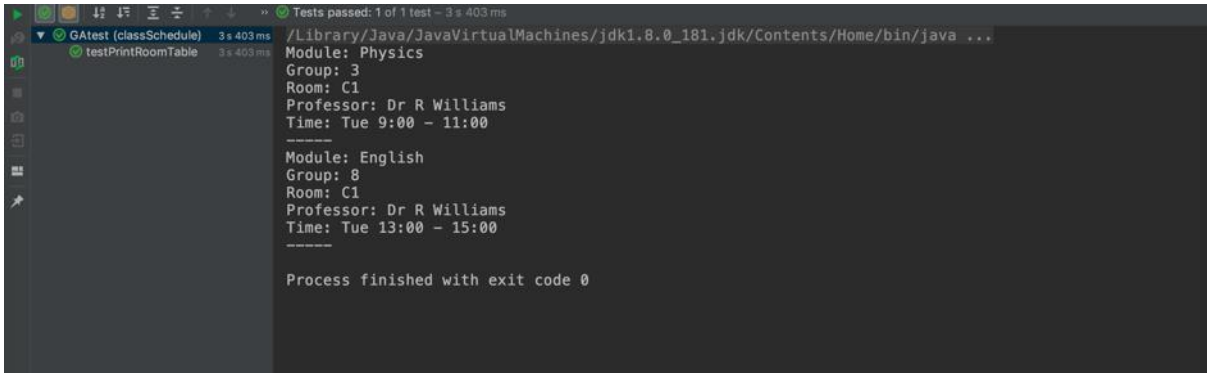
TestPrintProfessorTable



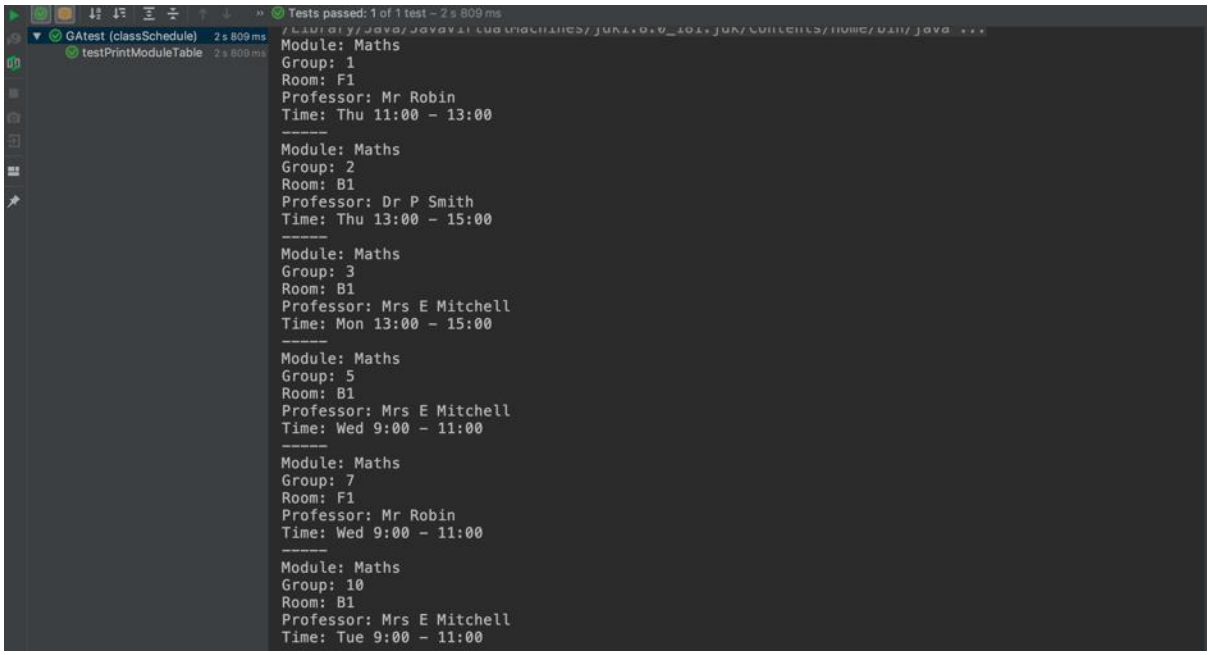
TestPrintGroupTable



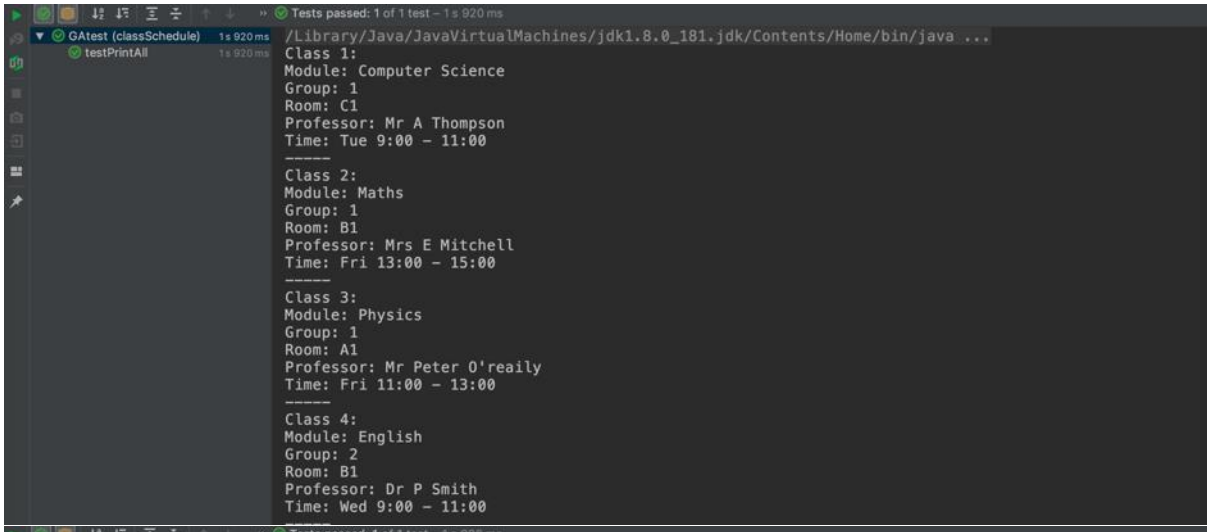
TestPrintRoomTable



TestPrintModuleTable



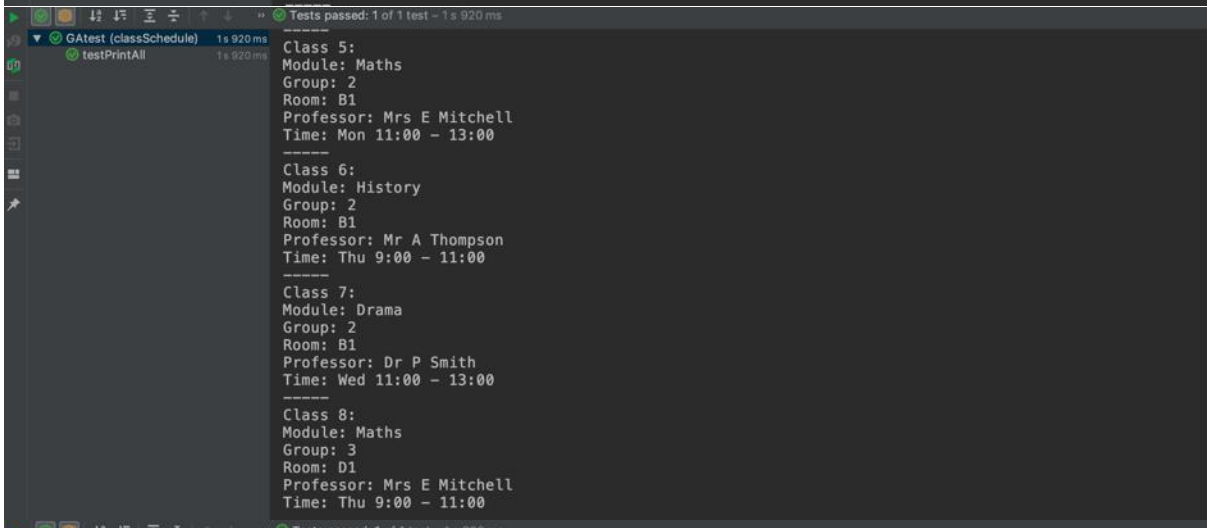
TestPrintAll


Tests passed: 1 of 1 test – 1 s 920 ms
GAtest (classSchedule) 1 s 920 ms
testPrintAll 1 s 920 ms
/Library/Java/JavaVirtualMachines/jdk1.8.0_181.jdk/Contents/Home/bin/java ...
Class 1:
Module: Computer Science
Group: 1
Room: C1
Professor: Mr A Thompson
Time: Tue 9:00 - 11:00

Class 2:
Module: Maths
Group: 1
Room: B1
Professor: Mrs E Mitchell
Time: Fri 13:00 - 15:00

Class 3:
Module: Physics
Group: 1
Room: A1
Professor: Mr Peter O'reaily
Time: Fri 11:00 - 13:00

Class 4:
Module: English
Group: 2
Room: B1
Professor: Dr P Smith
Time: Wed 9:00 - 11:00

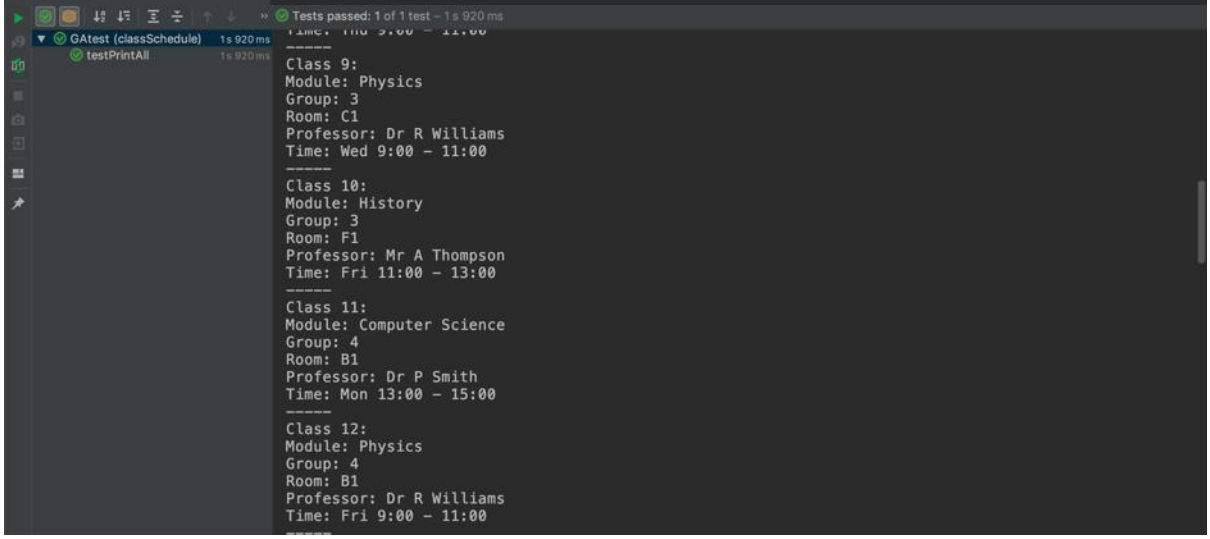

Tests passed: 1 of 1 test – 1 s 920 ms
GAtest (classSchedule) 1 s 920 ms
testPrintAll 1 s 920 ms

Class 5:
Module: Maths
Group: 2
Room: B1
Professor: Mrs E Mitchell
Time: Mon 11:00 - 13:00

Class 6:
Module: History
Group: 2
Room: B1
Professor: Mr A Thompson
Time: Thu 9:00 - 11:00

Class 7:
Module: Drama
Group: 2
Room: B1
Professor: Dr P Smith
Time: Wed 11:00 - 13:00

Class 8:
Module: Maths
Group: 3
Room: D1
Professor: Mrs E Mitchell
Time: Thu 9:00 - 11:00

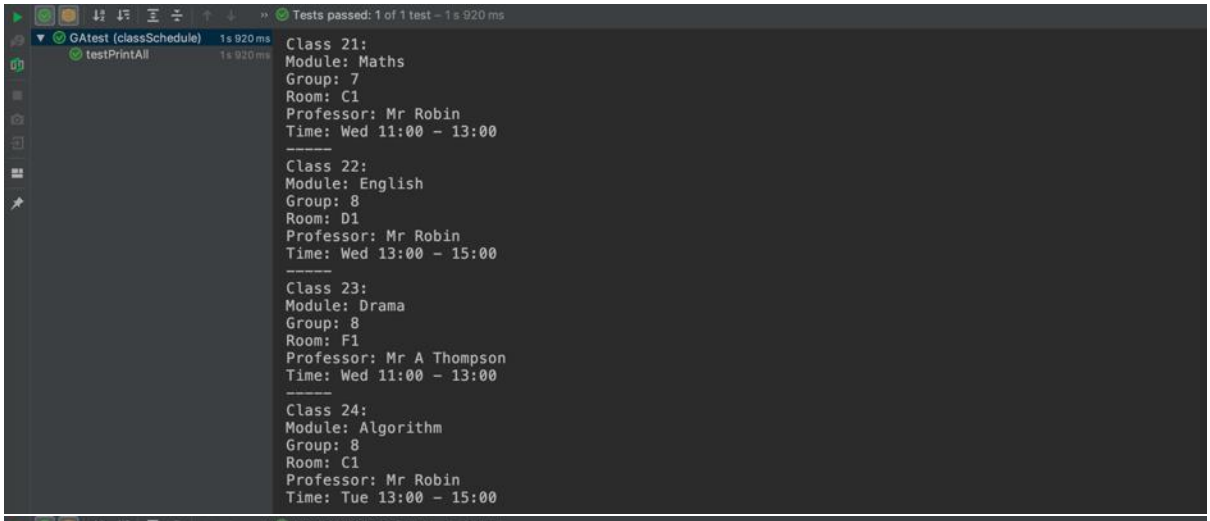

Tests passed: 1 of 1 test – 1 s 920 ms
GAtest (classSchedule) 1 s 920 ms
testPrintAll 1 s 920 ms

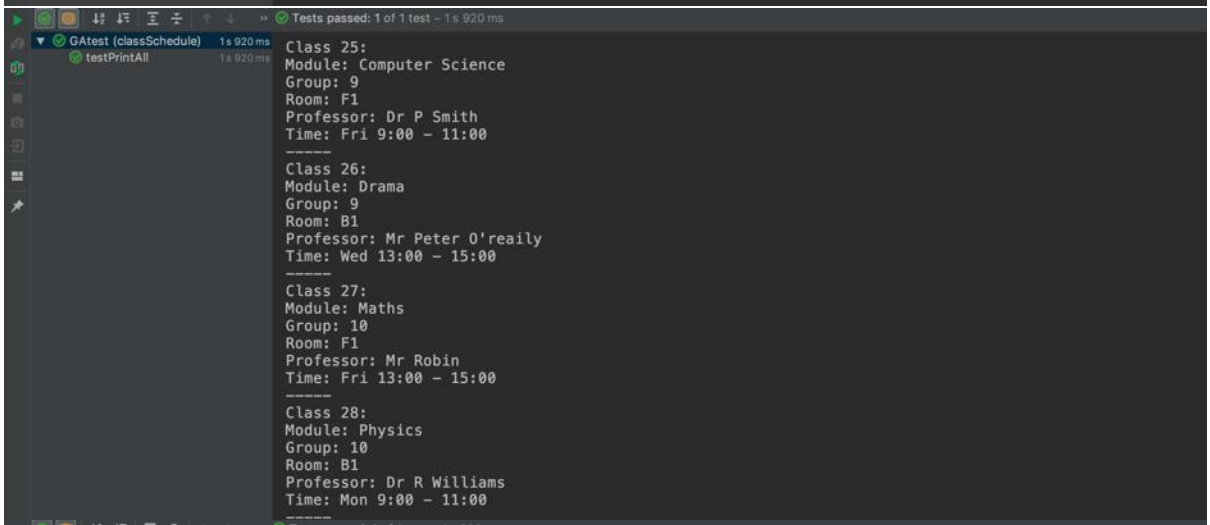
Class 9:
Module: Physics
Group: 3
Room: C1
Professor: Dr R Williams
Time: Wed 9:00 - 11:00

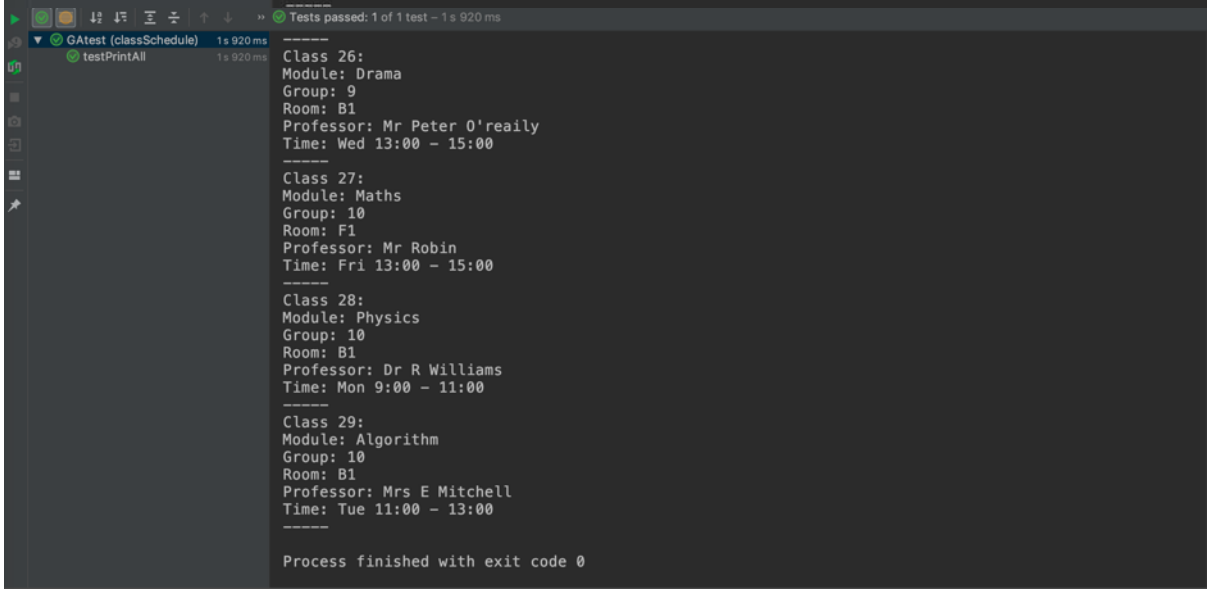
Class 10:
Module: History
Group: 3
Room: F1
Professor: Mr A Thompson
Time: Fri 11:00 - 13:00

Class 11:
Module: Computer Science
Group: 4
Room: B1
Professor: Dr P Smith
Time: Mon 13:00 - 15:00

Class 12:
Module: Physics
Group: 4
Room: B1
Professor: Dr R Williams
Time: Fri 9:00 - 11:00


Tests passed: 1 of 1 test – 1 s 920 ms
GAtest (classSchedule) 1 s 920 ms
testPrintAll 1 s 920 ms
Class 21:
Module: Maths
Group: 7
Room: C1
Professor: Mr Robin
Time: Wed 11:00 - 13:00
_____
Class 22:
Module: English
Group: 8
Room: D1
Professor: Mr Robin
Time: Wed 13:00 - 15:00
_____
Class 23:
Module: Drama
Group: 8
Room: F1
Professor: Mr A Thompson
Time: Wed 11:00 - 13:00
_____
Class 24:
Module: Algorithm
Group: 8
Room: C1
Professor: Mr Robin
Time: Tue 13:00 - 15:00


Tests passed: 1 of 1 test – 1 s 920 ms
GAtest (classSchedule) 1 s 920 ms
testPrintAll 1 s 920 ms
Class 25:
Module: Computer Science
Group: 9
Room: F1
Professor: Dr P Smith
Time: Fri 9:00 - 11:00
_____
Class 26:
Module: Drama
Group: 9
Room: B1
Professor: Mr Peter O'reaily
Time: Wed 13:00 - 15:00
_____
Class 27:
Module: Maths
Group: 10
Room: F1
Professor: Mr Robin
Time: Fri 13:00 - 15:00
_____
Class 28:
Module: Physics
Group: 10
Room: B1
Professor: Dr R Williams
Time: Mon 9:00 - 11:00
_____


Tests passed: 1 of 1 test – 1 s 920 ms
GAtest (classSchedule) 1 s 920 ms
testPrintAll 1 s 920 ms
_____
Class 26:
Module: Drama
Group: 9
Room: B1
Professor: Mr Peter O'reaily
Time: Wed 13:00 - 15:00
_____
Class 27:
Module: Maths
Group: 10
Room: F1
Professor: Mr Robin
Time: Fri 13:00 - 15:00
_____
Class 28:
Module: Physics
Group: 10
Room: B1
Professor: Dr R Williams
Time: Mon 9:00 - 11:00
_____
Class 29:
Module: Algorithm
Group: 10
Room: B1
Professor: Mrs E Mitchell
Time: Tue 11:00 - 13:00
_____
Process finished with exit code 0