# Глава 2

## 2.1

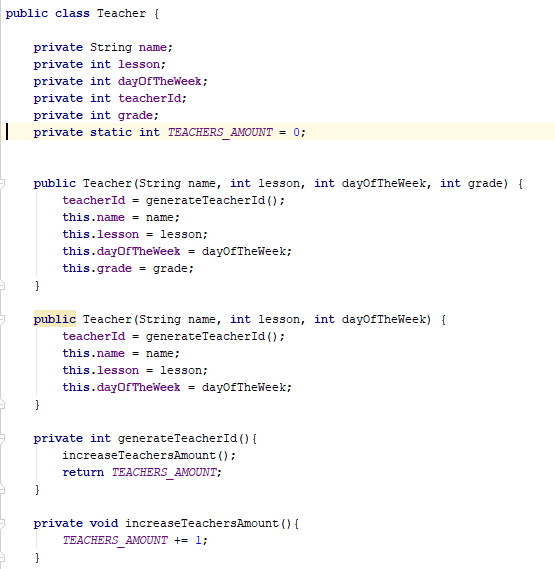
Резюмируя изложенное в главе 1, генетические алгоритмы – мощнейший инструмент для решения задач оптимизации и моделирования. Свое название получил также не случайно, ведь он моделирует генетические процессы, происходящие в живой природе. Исходя из этого мы можем смоделировать основные механизмы (мутация, селекция, скрещивание), наложить необходимые ограничения и эволюционным путем получить итоговое расписание.

Для решения задачи составления расписания используем язык программирования Java 1.8 и среду разработки Intellij Idea Community Edition.

На начальном этапе необходимо определить, из чего алгоритм должен создавать новые особи. Если представить расписание на день в виде одной из тех самых особей, то, для каждого класса, такой день будет состоять из предметов школьной программы. Каждый из предметов ведет определенный учитель. Так как расписание создается исходя из школьной программы и наличия рабочих часов учителей, то за минимальную единицу для составления расписание возьмем преподавателя.

Создадим класс *Teacher* для генерации преподавателей (рисунок 2.1) со следующими полями:

* *name* – имя преподавателя;
* *lesson –* номер урока по порядку;
* *dayOfTheWeek –* день недели, в который проводится урок.
* *teacherId –* уникальный идентификатор объекта класса *Teacher*
* *grade* – номер класса (Например: 1а, 8б)
* *TEACHERS\_AMOUNT –* количество созданных объектов класса *Teacher*.

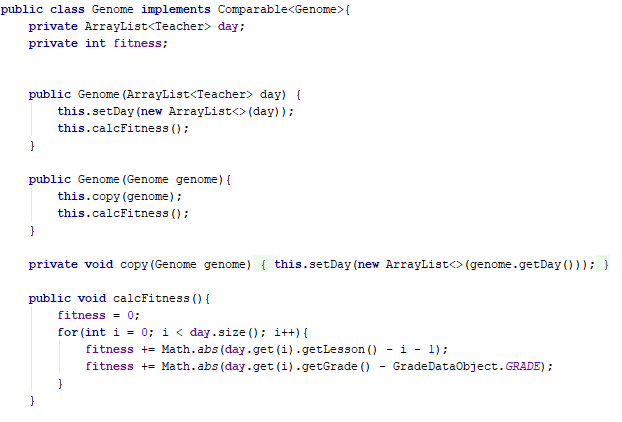


1. - Класс *Teacher*

Для того, чтобы объекты данного класса были действительно атомарными и неизменяемыми, разрешим задавать значение полей только при инициализации. Если учитель проводит несколько уроков в день, то для него создаются объекты в том же количестве (Например: 4 урока – 4 объекта).

Следующим шагом необходимо создать модель особи (генома), которая должна генерироваться с использованием объектов класса *Teacher.* Для этого напишем новый класс *Genome* (рисунок 2.2) со следующими полями:

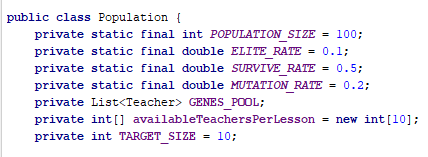
* *day –* список, содержащий в себе учителей в том порядке, в котором они ведут уроки в конкретный день недели;
* *fitness –* значение функции приспособленности для каждого объекта класса *Genome.*



1. - Класс *Genome*

За вычисление значения функции приспособленности отвечает функция *calcFitness().* Она вычисляет модуль разности между позицией преподавателя в списке *day* и номером урока по порядку, указанный в объекте *Teacher,* и добавляет к значению переменной *fitness*. Тоже самое происходит и с переменной, которая хранит в себе тот класс, который присутствует на уроке.   
Идеальной особью, которая попадет в финальную популяцию (расписание на день) является та, чье значение *fitness* равно 0.

Переходим к функциям, которые позволяют нам назвать разрабатываемый алгоритм генетическим. Они находятся в специально созданном классе *Population* и воспроизводят следующие механизмы живой природы: мутацию (*mutateGenome),* скрещивание (*mergeRandomGenomes)* иотбор (*selectElite).* Так же класс *Population* хранит набор переменных, констант и вспомогательных функций, которые позволяют реализовать алгоритм (рисунок 2.3).



1. - Переменные класса *Population*

Немного подробнее о переменных:

* *POPULATION\_SIZE –* определяет максимальный размер популяции;
* *ELITE\_RATE –* задаетпроцент особей, которые должны быть выбраны на этапе отбора;
* *SURVIVE\_RATE –* доля «выживших» особей. Позволяет ограничить количество особей, которые пригодны для скрещивания;
* *MUTATION\_RATE –* определяет шанс возникновения мутации;
* *GENES\_POOL* – хранит в себе все созданные объекты класса *Teacher,* которые могут быть использованы для создания особей;
* *availableTeachersPerLesson –* хранит информацию о доступных учителях на тот или иной урок;
* *TARGET\_SIZE* – максимальный размер учебного дня.

Рассмотрим реализацию отдельных функций.

*MutateGenome(Genome genome) –* производит мутацию одного гена (учителя) в геноме (учебный день). Для этого, с помощью переменной  *TARGET\_SIZE* и генератора случайных чисел от 0 до 1, вычисляется позиция гена, который заменяется на случайный. В качестве результата функция возвращает новый объект класса *Genome* (рисунок 2.4)*.*

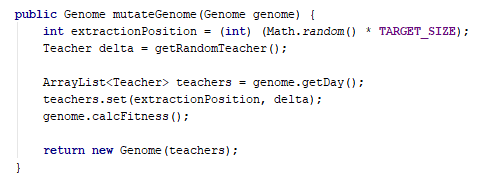


Рисунок 2.4 – Функция *mutateGenome*

*MergeRandomGenomes(List<Genome> population) -* производит слияние двух случайных геномов из всей популяции. Принцип действия основан на одноточечном кроссинговере. Изначально, из всей популяции, отбирается некоторое число особей, которые не будут подвержены никаким изменениям, то есть имеют наименьшее значение приспособленности (*fitness).* Затем с помощью генератора случайных чисел, переменных *POPULATION\_SIZE* и *SURVIVE\_RATE* рассчитываются позиции двух родителей в существующей популяции. Далее происходит слияние половинок обоих геномов в точке, выбранной случайным образом. На следующем этапе, с помощью генератора случайных чисел, решается, будет ли новая особь подвержена мутации или нет. После чего она добавляется в пул. Результатом работы функции является список геномов, состоящий из отобранной «элиты» с минимальным значением *fitness* и геномов, подвергшимся слиянию (рисунок 2.5).

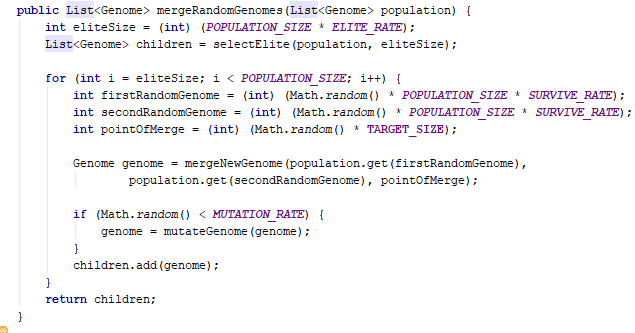


Рисунок 2.5 – Функция mergeRandomGenomes

Функция *selectElite(List<Genome> population, int eliteSize) –* является аналогом механизму отбора в живой природе. Принимает на вход отсортированную популяцию и размер выборки, которая должна получится. Результатом работы функции является список отобранных геномов с наименьшим значением функции приспособленности (рисунок 2.6).

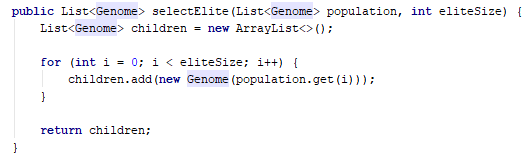


Рисунок 2.6 – Функция selectElite

Входные данные алгоритм получает из электронной таблицы формата *xlsx* (рисунок 2.7)*.* Она состоит из 4 столбцов: учитель, номер урока, день недели и класс. Номера уроков, в которые учитель может вести занятия, а также классы, с которыми готов работать, можно заносить в таблицу в разных форматах:

* единичным числом (Например: 1);
* набором чисел через запятую (Например: 1,2,4);
* диапазоном чисел (Например: 2-5);
* вариацией вышеописанных форматов (Например: 1,3-5,7).

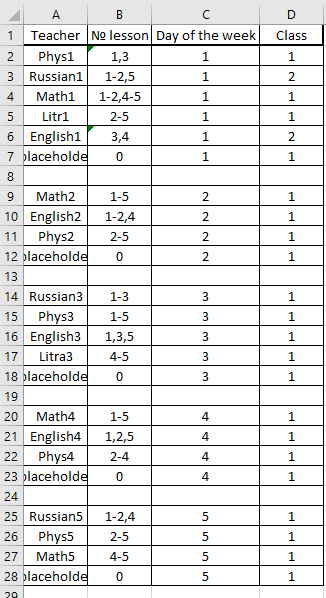


Рисунок 2.7 – Таблица входных данных

Для обработки данных из электронной таблицы создаем вспомогательные классы *TeachersCreator* и *XLSParser.* Как следует из названия, они необходимы для чтения файла и создания объектов класса *Teacher*.

Рассмотрим класс *XLSParser* (рисунок 2.8)*.* Он служит исключительно для считывания данных из файла и содержит лишь одну функцию *parse().* Для каждой строки создается переменная *String*, в которую входит содержимое всех её ячеек*.*

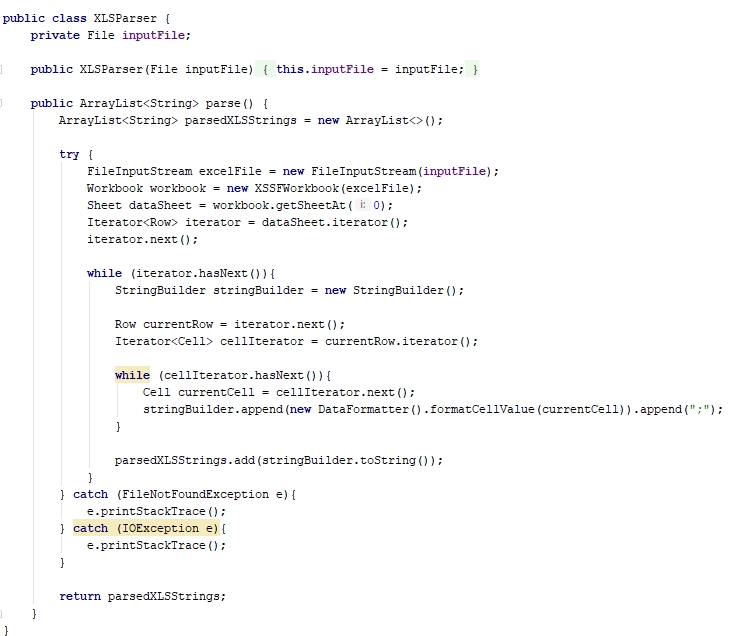


Рисунок 2.8 – Класс *XLSParser*

Полученный пул строк передается на обработку классу *TeachersCreator.* Он состоит из двух статических *(static)* и двух нестатических *(non-static)*  методов.

Статические методы *getTeachersForDay(int dayOfTheWeek)* и *getTeachers()* предоставляют доступ к пулу учителей на один конкретный день или на всю неделю соответственно (рисунок 2.9).

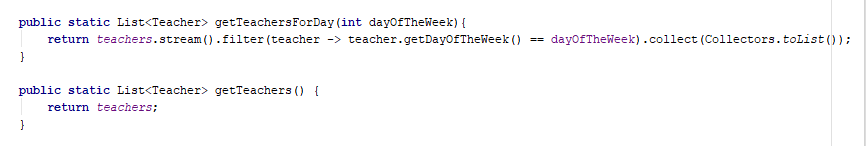


Рисунок 2.9 – Статические методы класса *TeachersCreator*

Что касается нестатических методов, то их устройство немного сложнее. *createTeachers()* проходит по всем строкам, полученным из метода *parse()* класса *XLSParser,* и создает из них пул объектов *Teacher*. Каждое поле объекта в строке отделяется знаком « ; ». Для преобразования последовательностей чисел, которыми представлены уроки и классы, используется вспомогательная функция *getNumbersFromString(String rangeOfLessons).* Для каждого урока и для каждого класса создается свой объект *Teacher* (рисунок 2.10).

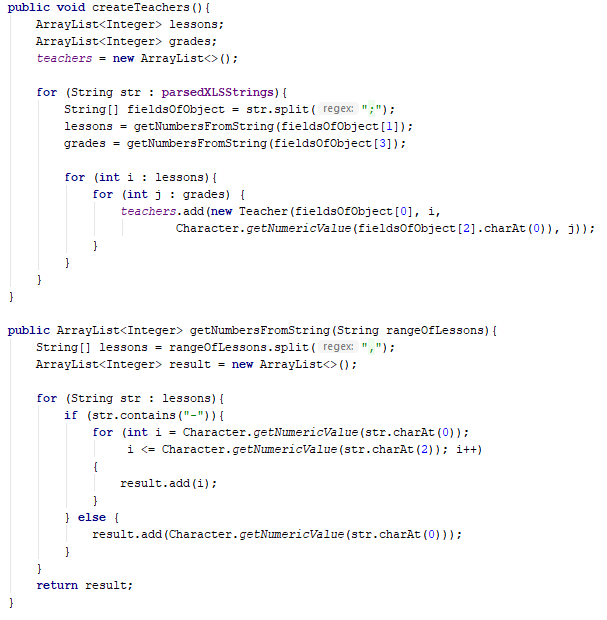


Рисунок 2.10 – Нестатические методы класса *TeachersCreator*