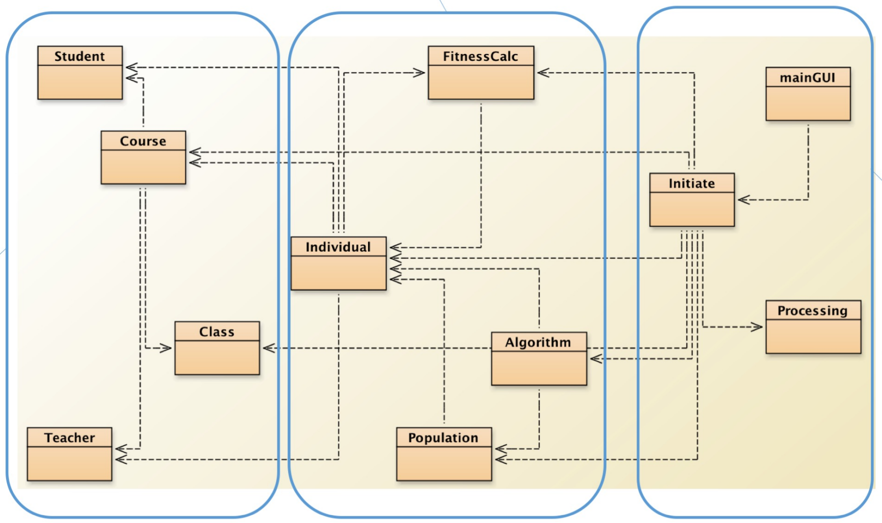
Criterion C: Development

List of techniques

- Data Structure—Hashmap and ArrayList
- Genetic Algorithm
- Inner Class and Two-Dimensional Array
- Graphical User Interface
- File I/O
- Exception Handling
- Multithreading

Class UML Diagram

Classes related to genetic algorithm



Classes related to the scenario

Classes related to GUI, I/O and pre-process of data

Figure C.1 Class Unified Modeling Language (UML) Diagram

Data Structure—Hashmap and ArrayList

The goal of the project is to generate six groups of classes so that all the success criteria are fulfilled. The six groups are represented by a Hashmap data structure with six keys (1-6). The value corresponding to each key is an ArrayList. Each ArrayList consists of multiple units called ***Gene*** objects. Essentially, a ***Gene*** object represents one specific course selection of a student. It has several key fields: **student name**, **course subject**, **course level**, **maximum class number** of the course, and **teacher number** of the course. If some ***Gene*** objects in the same group have the same course name and course level, students represented by these objects will be considered as in the same class. This representation is used to facilitate the calculation of the program.

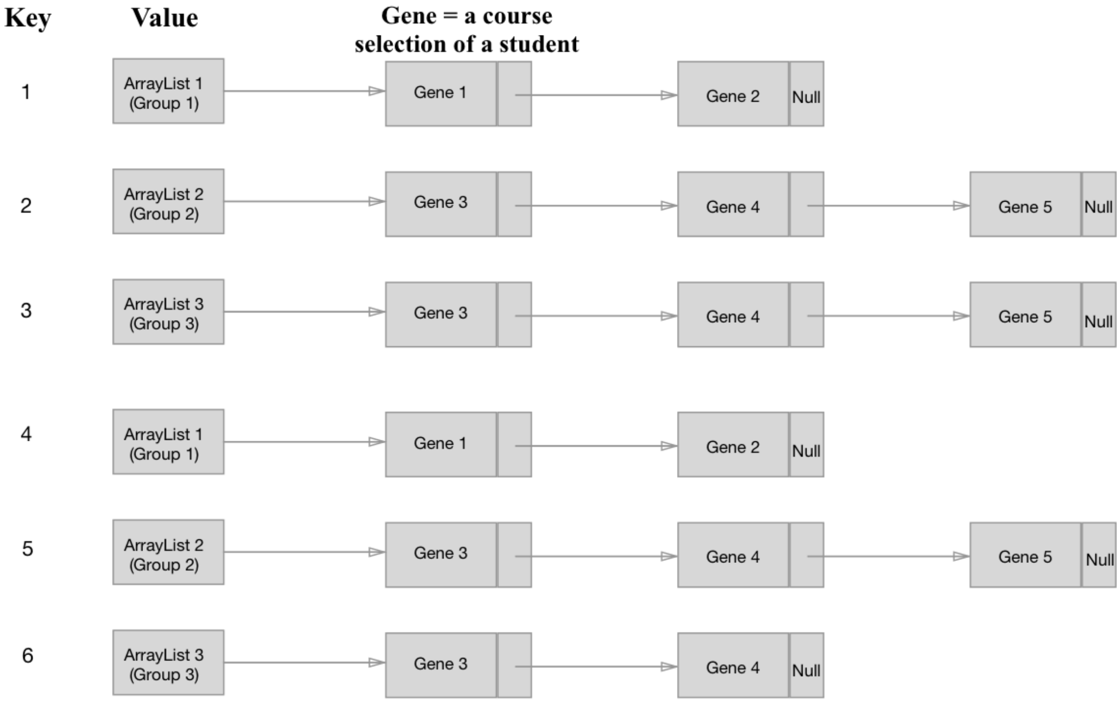


Figure C.2 Hashmap and ArrayList Data

Now recall the key success criteria:

1. Each student has only one class in each group;

2. The size of a class should be more than 3 and less than 20;

3. If a subject has only one teacher, there cannot be two classes of the subject in the same group;

Under the representation described above, the success criteria can be translated into following three criteria, which will be referred to as **fitness criteria**:

1. In each ArrayList, no two ***Gene*** objects have the same **student name**;

2. The number of ***Gene*** objects with the same **course subject** and **course level** in an ArrayList should be more than 3 and less than 20;

3. If teacher number is 1, ***Gene*** objects with the same **course subject** but different **course level** should not appear in the same ArrayList.

Genetic Algorithm

Genetic algorithm is a meta-heuristic approach to optimize a solution. It is suitable to this project because there are numerous arrangements of ***Gene*** objects and it is impossible to try them all.

Genetic algorithm starts from a collection of randomly generated solutions. The collection is called **Population** and each solution is an ***Individual*** object. Each ***Individual*** objects is consisted of ***Gene*** objects. The algorithm gradually evolves **Population** to find an optimum ***Individual***. Two key operations of evolvement are **crossover** and **mutation.**

A **fitness value** of an ***Individual*** object, calculated based on **fitness criteria**, measures the perfectness of a solution. The higher the fitness value, the better the solution. If the fitness value equals to the maximum fitness value, the perfect solution is found.[1]

[1] Jacobson, Lee. "Creating a genetic algorithm for beginners." *Theprojectspot.com.* Theprojectspot.com, 2012. Web. 25 Jul. 2017.

The following diagram illustrates genetic algorithm with the goal of **finding the 8-digit binary number, 00001111**. The upper two blocks explain the relationship between ***Gene***, ***Individual***, and ***Population***. The middle block explains **crossover** and **mutation**. The third block shows the new population after one evolution, and evolutions will continue on until the algorithm gets the result, 00001111.

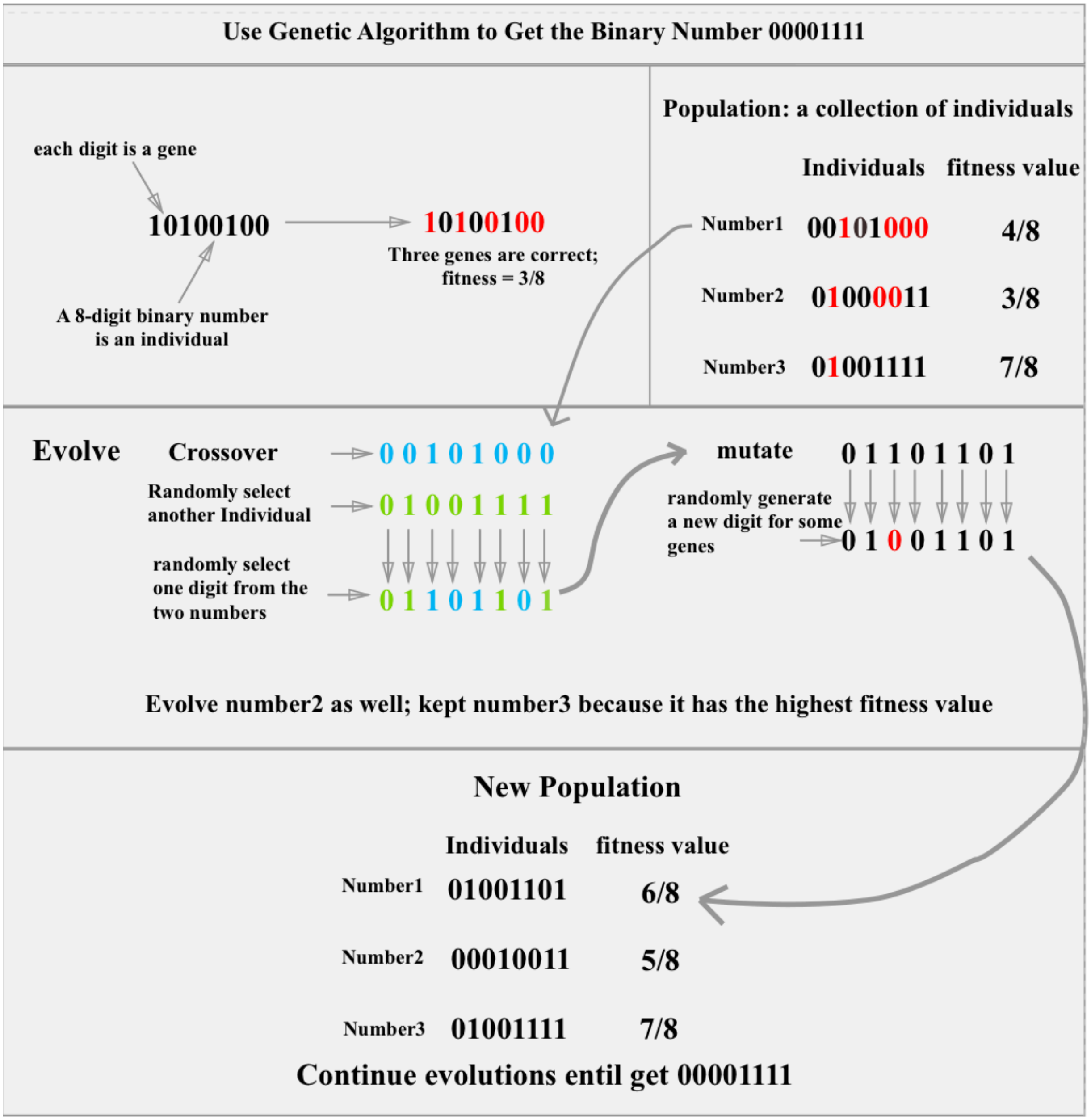
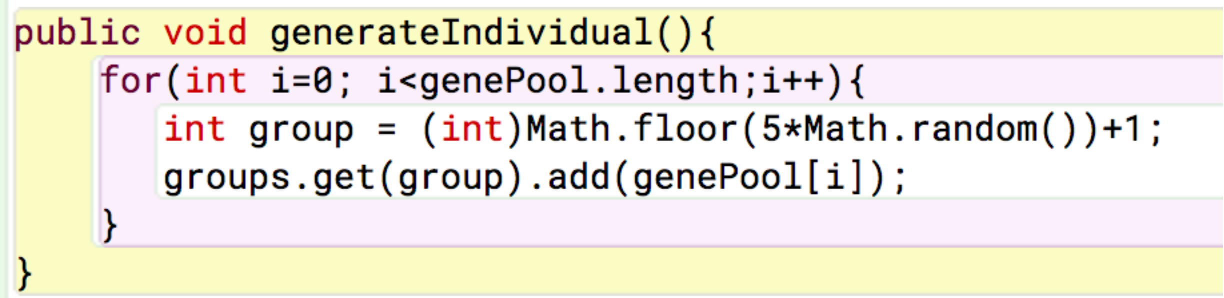


Figure C.3 Demonstration of Genetic Algorithm

In the system, I represent the features of genetic algorithm as following:

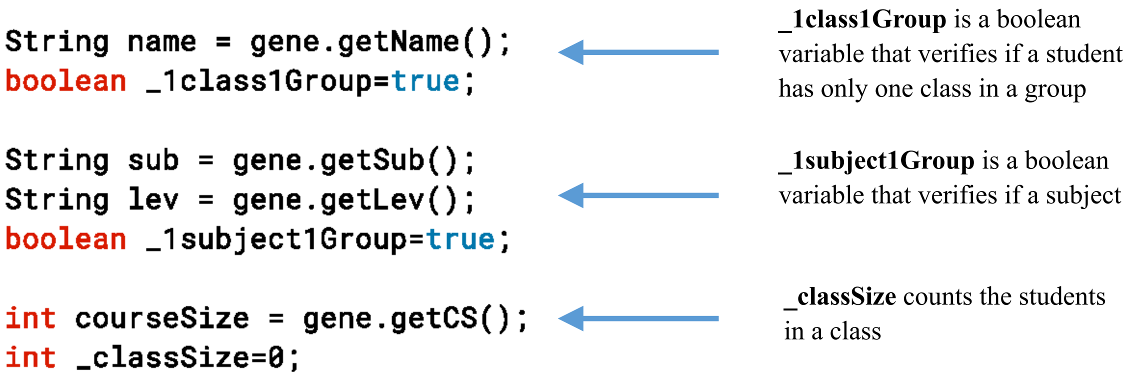
1. A ***Gene*** object represents a specific course selection of a student.
2. An ***Individual*** object is a Hashmap such that ***Gene*** objects are assigned to an ArrayList corresponding to a key (1-6) randomly generated by **5 * *Math.random()*+1**.



```java
public void generateIndividual(){
    for(int i=0; i<genePool.length;i++){
        int group = (int)Math.floor(5*Math.random())+1;
        groups.get(group).add(genePool[i]);
    }
}
```

Figure C.4 The method that initializes an ***Individual*** object by assigning ***Gene*** objects to random groups

3. A population is an ArrayList consisting of a certain number of ***Individual*** objects.
4. Fitness value is calculated based on three **fitness criteria** described above. For each ***Gene*** object, it will be awarded 1 for each fulfilled criterion. Therefore, the total fitness value is (3 * number of ***Gene*** objects).



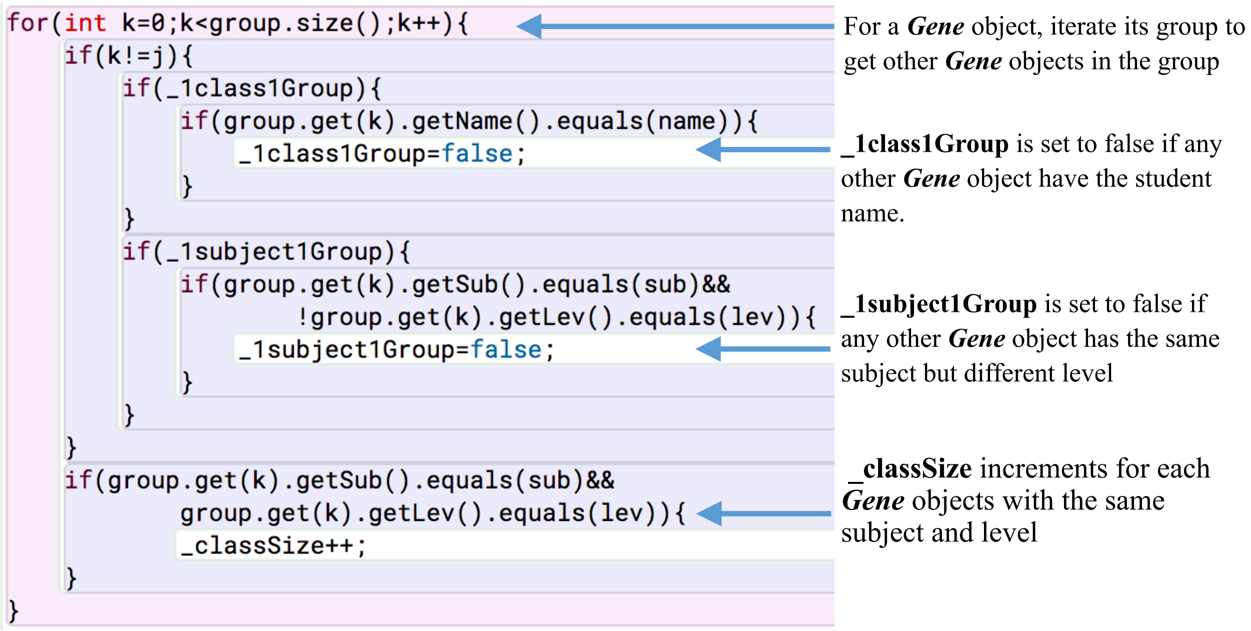
```java
String name = gene.getName();
boolean _1class1Group=true;

String sub = gene.getSub();
String lev = gene.getLev();
boolean _1subject1Group=true;

int courseSize = gene.getCS();
int _classSize=0;
```

_1class1Group is a boolean variable that verifies if a student has only one class in a group

_1subject1Group is a boolean variable that verifies if a subject

_classSize counts the students in a class

Figure C.5-1 The method that calculated fitness values
Necessary variables are declared.



```
for(int k=0;k<group.size();k++){
    if(k!=j){
        if(_1class1Group){
            if(group.get(k).getName().equals(name)){
                _1class1Group=false;
            }
        }
        if(_1subject1Group){
            if(group.get(k).getSub().equals(sub)&&
                    !group.get(k).getLev().equals(lev)){
                _1subject1Group=false;
            }
        }
    }
    if(group.get(k).getSub().equals(sub)&&
            group.get(k).getLev().equals(lev)){
            _classSize++;
    }
}
```

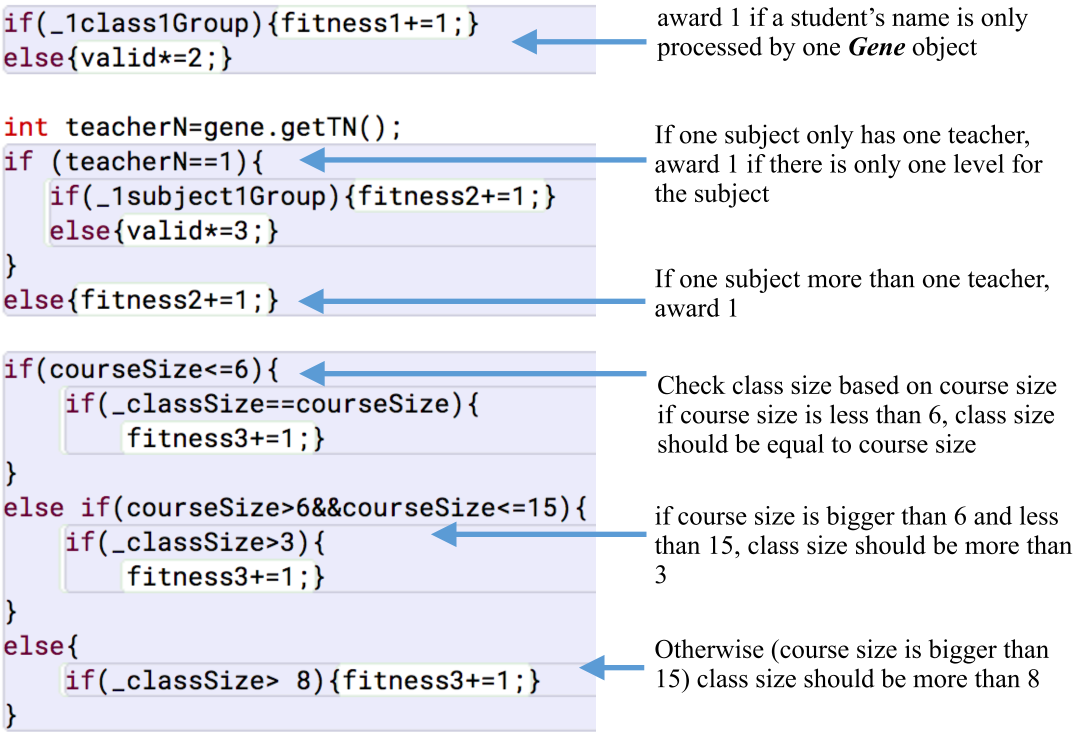
For a ***Gene*** object, iterate its group to get other ***Gene*** objects in the group

_1class1Group is set to false if any other ***Gene*** object have the student name.

_1subject1Group is set to false if any other ***Gene*** object has the same subject but different level

_classSize increments for each ***Gene*** objects with the same subject and level

Figure C.5-2 The method that calculates fitness values



```
if(_1class1Group){fitness1+=1;}
else{valid*=2;}

int teacherN=gene.getTN();
if (teacherN==1){
   if(_1subject1Group){fitness2+=1;}
   else{valid*=3;}
}
else{fitness2+=1;}

if(courseSize<=6){
    if(_classSize==courseSize){
        fitness3+=1;}
}
else if(courseSize>6&&courseSize<=15){
    if(_classSize>3){
        fitness3+=1;}
}
else{
    if(_classSize> 8){fitness3+=1;}
}
```

award 1 if a student's name is only processed by one ***Gene*** object

If one subject only has one teacher, award 1 if there is only one level for the subject

If one subject more than one teacher, award 1

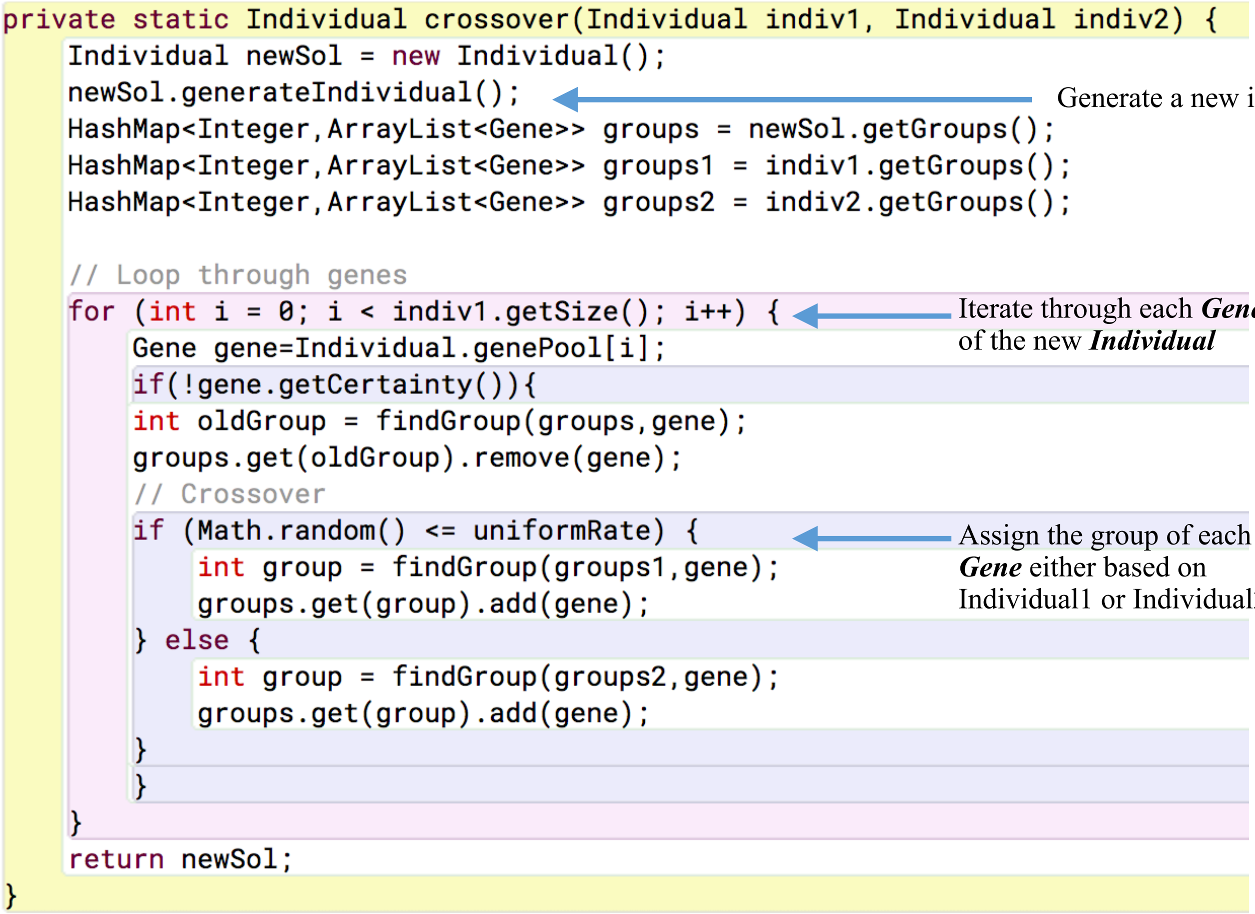
Check class size based on course size if course size is less than 6, class size should be equal to course size

if course size is bigger than 6 and less than 15, class size should be more than 3

Otherwise (course size is bigger than 15) class size should be more than 8

Figure C.5-3 The method that calculates fitness values
Fitness value is awarded according to fitness (success) criteria.

5. Crossover is done by combining the ***Gene*** objects of two ***Individual*** objects.



```
private static Individual crossover(Individual indiv1, Individual indiv2) {
    Individual newSol = new Individual();
    newSol.generateIndividual();
    HashMap<Integer,ArrayList<Gene>> groups = newSol.getGroups();
    HashMap<Integer,ArrayList<Gene>> groups1 = indiv1.getGroups();
    HashMap<Integer,ArrayList<Gene>> groups2 = indiv2.getGroups();

    // Loop through genes
    for (int i = 0; i < indiv1.getSize(); i++) {
        Gene gene=Individual.genePool[i];
        if(!gene.getCertainty()){
        int oldGroup = findGroup(groups,gene);
        groups.get(oldGroup).remove(gene);
        // Crossover
        if (Math.random() <= uniformRate) {
            int group = findGroup(groups1,gene);
            groups.get(group).add(gene);
        } else {
            int group = findGroup(groups2,gene);
            groups.get(group).add(gene);
        }
        }
    }
    return newSol;
}
```

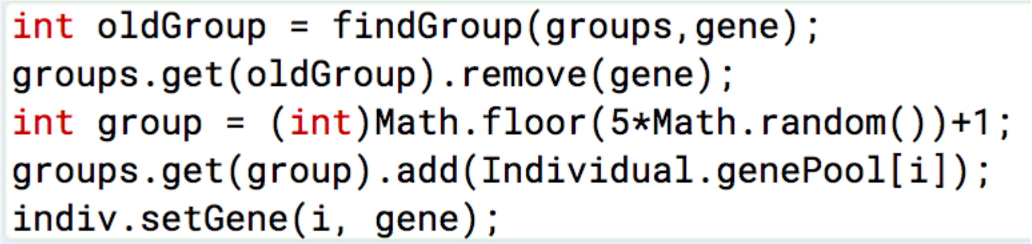
Generate a new individual

Iterate through each ***Gene*** of the new ***Individual***

Assign the group of each ***Gene*** either based on Individual1 or Individual2

Figure C.6 Crossover method

6. Mutation is done by randomly select a ***Gene*** object and change its group.



```
int oldGroup = findGroup(groups,gene);
groups.get(oldGroup).remove(gene);
int group = (int)Math.floor(5*Math.random())+1;
groups.get(group).add(Individual.genePool[i]);
indiv.setGene(i, gene);
```

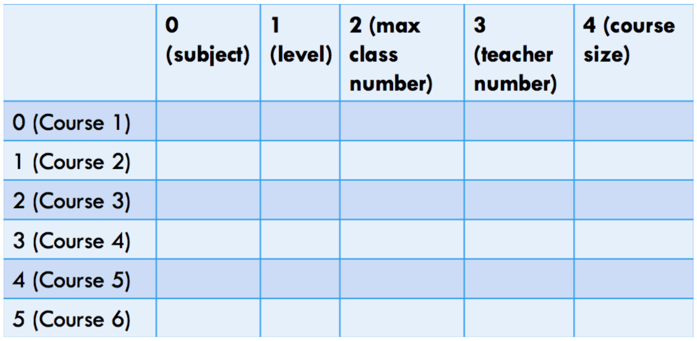
Assign the ***Gene*** object to another group randomly

Figure C.7 Mutation method

Inner Class and Two-Dimensional Array

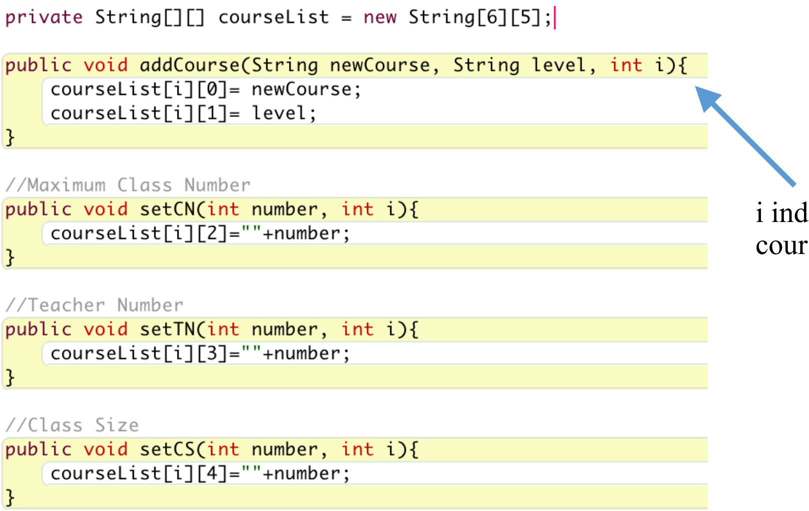
Gene class is an inner class rather than an independent class because ***Gene*** objects are solely instantiated and used in ***Individual*** class. By declaring ***Gene*** as an inner class, I can avoid unnecessary implementation of accessor and mutator methods and increase encapsulation of the program, making the program more readable and maintainable.[2]

Since each student chooses six courses, there are (6 * number of students) ***Gene*** objects. The values of fields of ***Gene*** objects are taken from a field of ***Student*** class, a two-dimensional array, that stores information of courses. Each row consists of a course's information which will be used to instantiate one ***Gene*** object. Each column stands for a specific type of information of courses.



	0 (subject)	1 (level)	2 (max class number)	3 (teacher number)	4 (course size)
0 (Course 1)					
1 (Course 2)					
2 (Course 3)					
3 (Course 4)					
4 (Course 5)					
5 (Course 6)					

Figure C.8 Graphical representation of the 2D-array



```
private String[][] courseList = new String[6][5];

public void addCourse(String newCourse, String level, int i){
    courseList[i][0]= newCourse;
    courseList[i][1]= level;
}

//Maximum Class Number
public void setCN(int number, int i){
    courseList[i][2]=""+number;
}

//Teacher Number
public void setTN(int number, int i){
    courseList[i][3]=""+number;
}

//Class Size
public void setCS(int number, int i){
    courseList[i][4]=""+number;
}
```

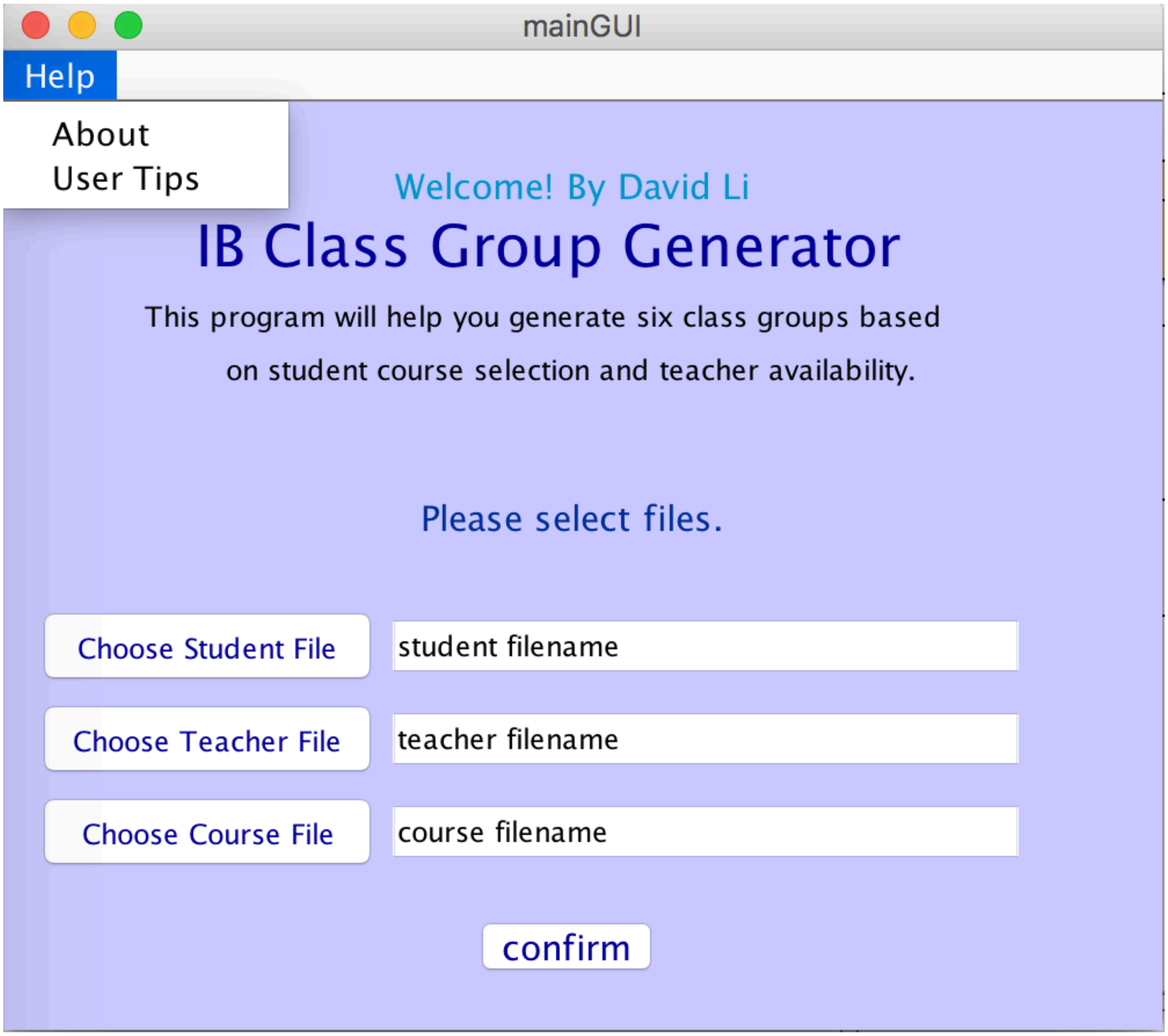
i indicates a specific course (0-5)

Figure C.9 Initiation of the 2D-array (methods in ***Student*** class)

[2] SwankSwashbucklers. "What are the purposes of inner classes." *Questions.* StackOverflow, 30 Jan. 2015. Web. 15 Aug. 2017.

Graphical User Interface

Two windows are created for the project. The first window, initiated by ***mainGUI*** class, uses ***JFileChooser*** to ask a user to input three files. The second window, initiated by ***Processing*** class, displays a progression bar and a percentage that indicates the perfectness of the current best solution. This is achieved by passing the fitness value as a parameter to ***Processing*** class and using ***Graphics*** class in Java to **paint** and **repaint** rectangles with different lengths. Both windows use ***JMenu*** to provide further instructions to the user.[3]



Welcome! By David Li

IB Class Group Generator

This program will help you generate six class groups based on student course selection and teacher availability.

Please select files.

student filename

teacher filename

course filename

Figure C.10 The first window, initiated by mainGUI

[3] "JMenu (Java Platform SE 7)." *Class*. Oracle, 2017. Web. 15 Aug. 2017.

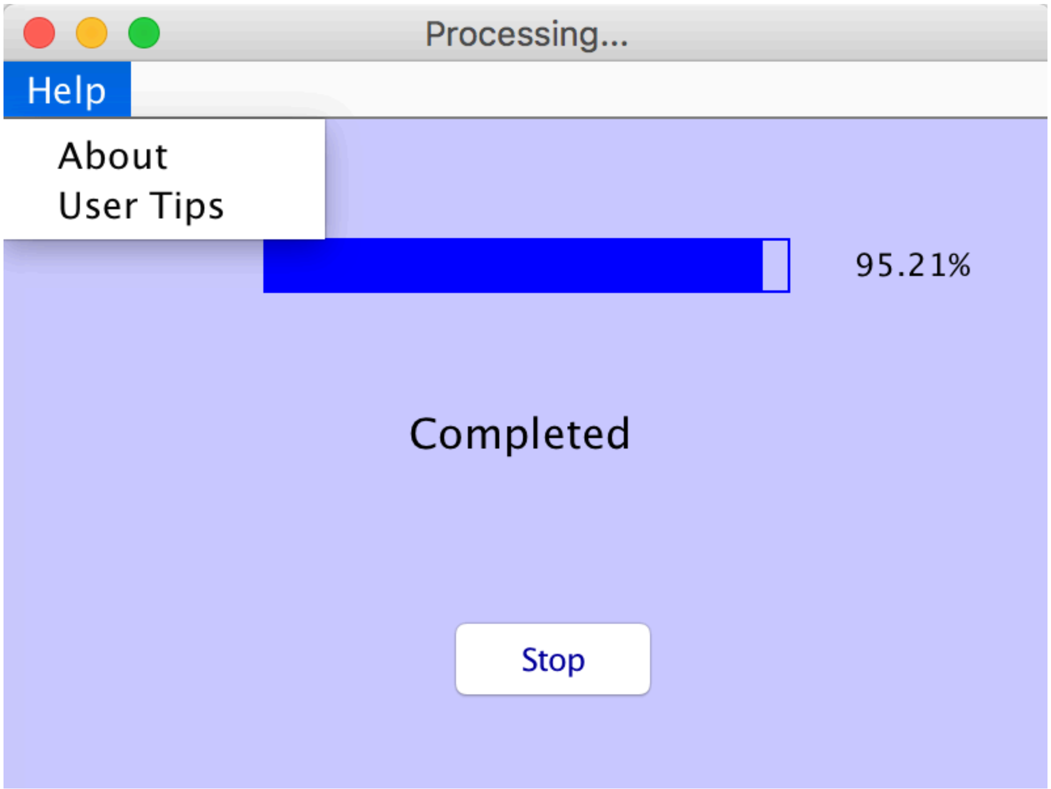
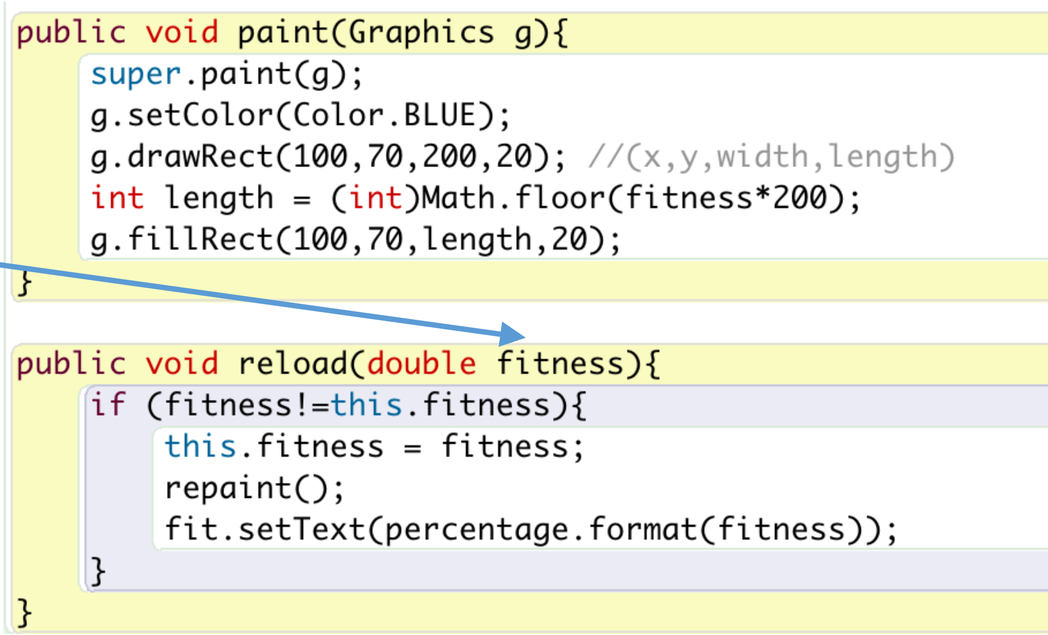


Figure C.11 The second window, initiated by Processing



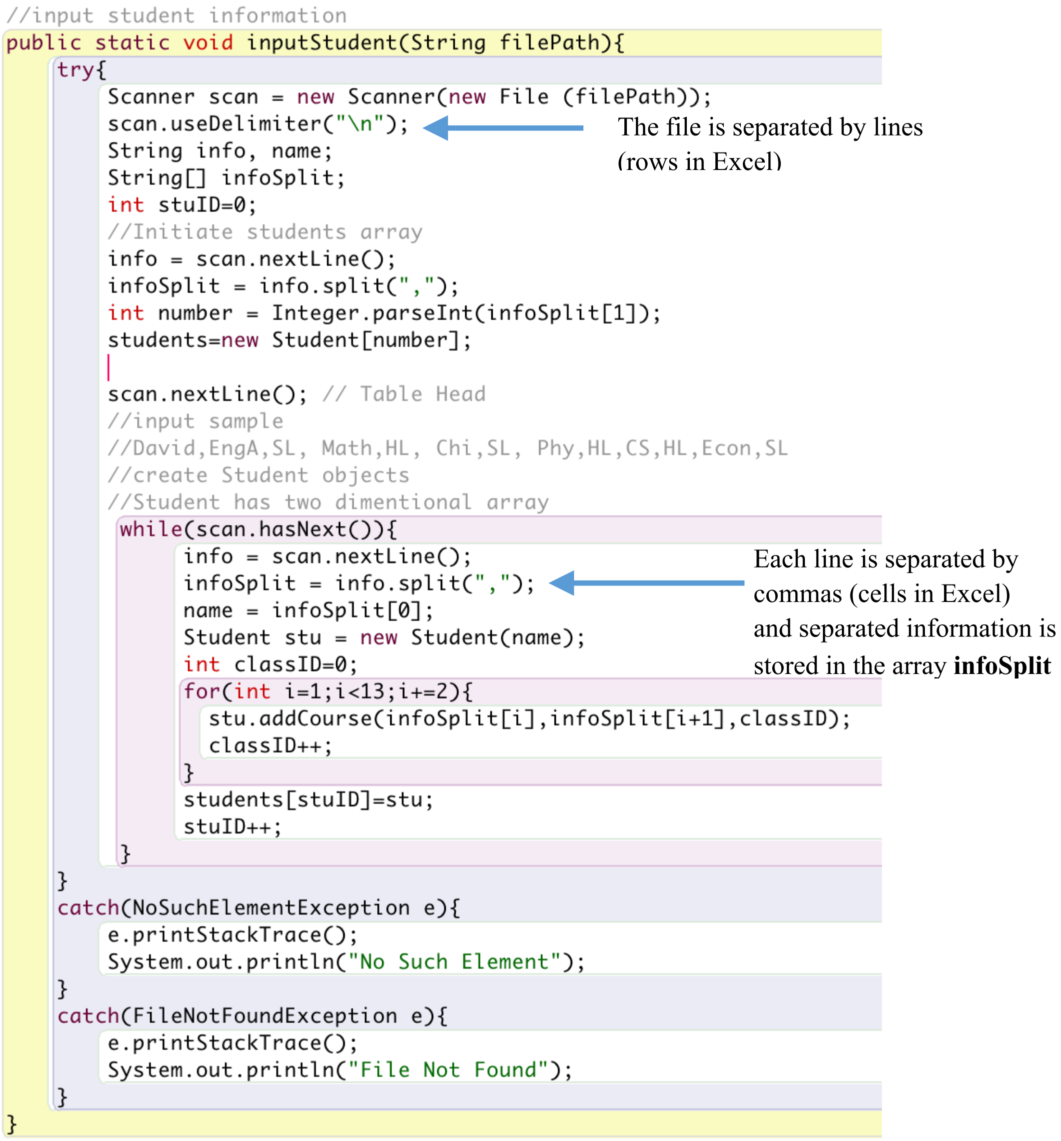
```
public void paint(Graphics g){
    super.paint(g);
    g.setColor(Color.BLUE);
    g.drawRect(100,70,200,20); //(x,y,width,length)
    int length = (int)Math.floor(fitness*200);
    g.fillRect(100,70,length,20);
}

public void reload(double fitness){
    if (fitness!=this.fitness){
        this.fitness = fitness;
        repaint();
        fit.setText(percentage.format(fitness));
    }
}
```

Object can call the method ***reload()*** and the parameter will be used to update the field, which decides the length of the progression bar.

Figure C.12 Methods that displays of the progression bar

File I/O

The program uses ***Scanner*** class to input information CSV files. CSV files are used because a user can edit the files with Microsoft Excel, and the files can be parsed with "\n" into separate lines, and each line can be parsed with "," into separate words by ***Scanner.useDelimiter()*** and ***String.split()***. The program uses ***FileWriter*** to output information. ***toString()*** methods of Course and Class are used to output formatted information.[4]



```
//input student information
public static void inputStudent(String filePath){
    try{
        Scanner scan = new Scanner(new File (filePath));
        scan.useDelimiter("\n");
        String info, name;
        String[] infoSplit;
        int stuID=0;
        //Initiate students array
        info = scan.nextLine();
        infoSplit = info.split(",");
        int number = Integer.parseInt(infoSplit[1]);
        students=new Student[number];

        scan.nextLine(); // Table Head
        //input sample
        //David,EngA,SL, Math,HL, Chi,SL, Phy,HL,CS,HL,Econ,SL
        //create Student objects
        //Student has two dimentional array
        while(scan.hasNext()){
            info = scan.nextLine();
            infoSplit = info.split(",");
            name = infoSplit[0];
            Student stu = new Student(name);
            int classID=0;
            for(int i=1;i<13;i+=2){
                stu.addCourse(infoSplit[i],infoSplit[i+1],classID);
                classID++;
            }
            students[stuID]=stu;
            stuID++;
        }
    }
    catch(NoSuchElementException e){
        e.printStackTrace();
        System.out.println("No Such Element");
    }
    catch(FileNotFoundException e){
        e.printStackTrace();
        System.out.println("File Not Found");
    }
}
```

The file is separated by lines
(rows in Excel)

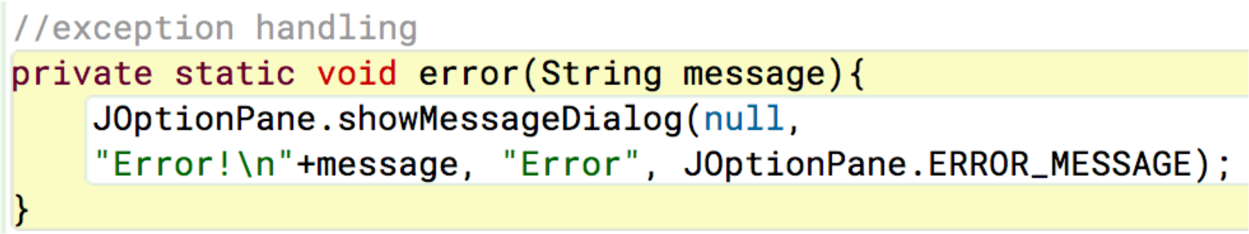
Each line is separated by
commas (cells in Excel)
and separated information is
stored in the array **infoSplit**

Figure C.13 The Method that inputs student information

[4] mkyong. "How to read and parse CSV file in Java." *Mcyong.* Mcyong.com, 2016. Web. 15 Aug. 2017.

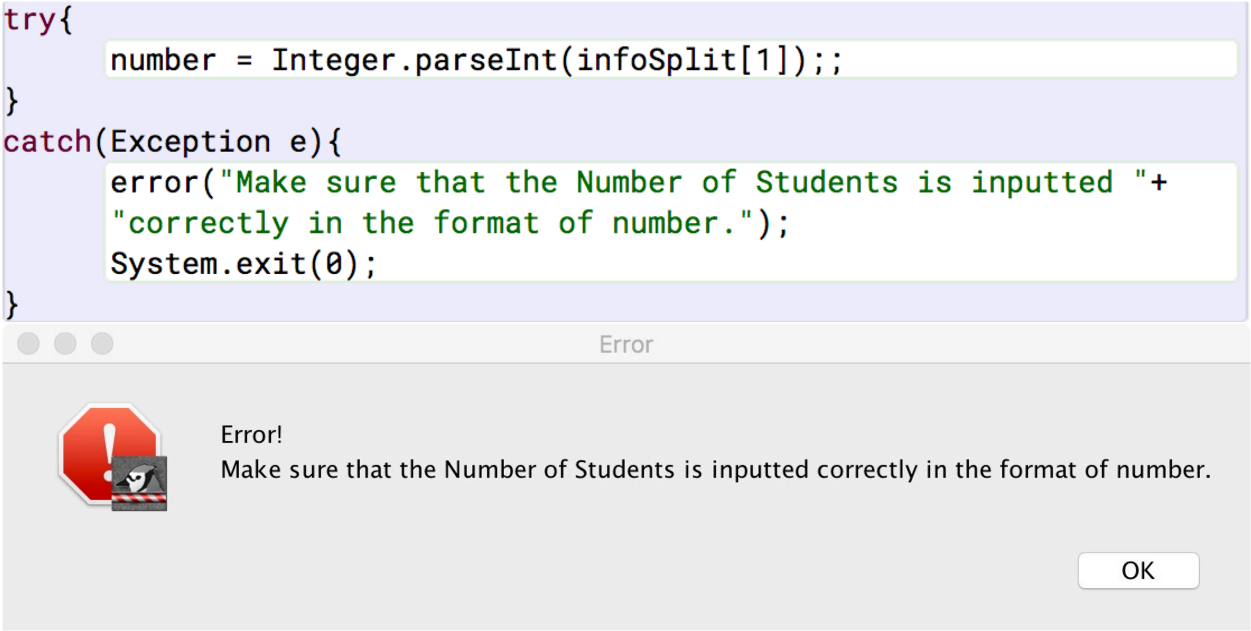
Exception Handling

Because the system requires the user to input data, a wide range of possible errors and exceptions exist. For example, a user may input a String instead of a number or input a wrong number.I use try-catch blocks to prevent exceptions from interrupting the program. Also, by utilizing ***JOptionPane***, I writes a method to display error messages, which makes the program more reusable.



```
//exception handling
private static void error(String message){
    JOptionPane.showMessageDialog(null,
    "Error!\n"+message, "Error", JOptionPane.ERROR_MESSAGE);
}
```

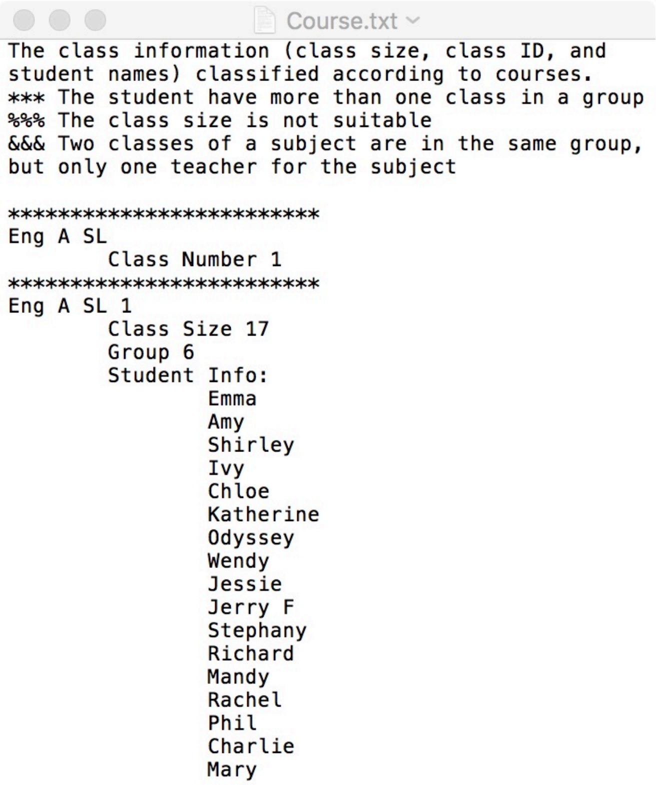
Figure C.10 The method that displays error messages



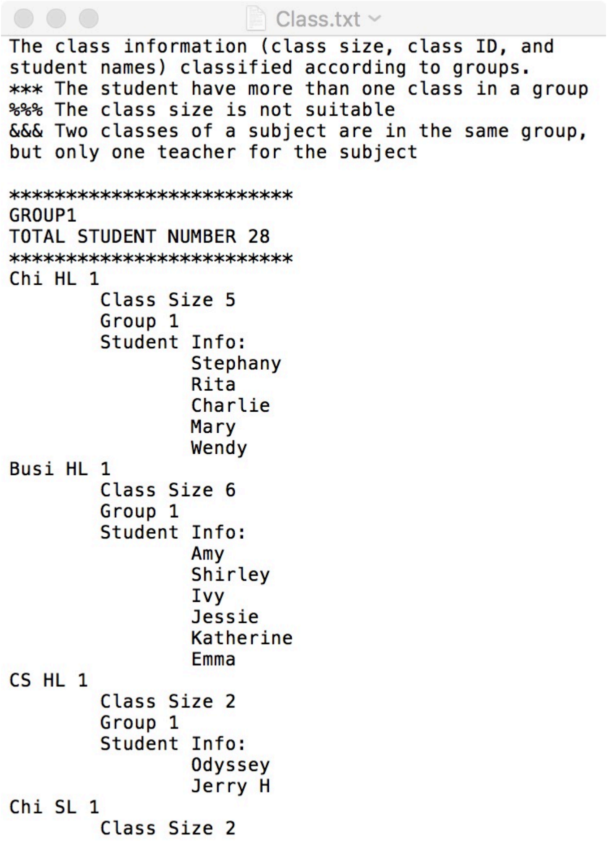
```
try{
       number = Integer.parseInt(infoSplit[1]);;
}
catch(Exception e){
       error("Make sure that the Number of Students is inputted "+
       "correctly in the format of number.");
       System.exit(0);
}
```

Error!
Make sure that the Number of Students is inputted correctly in the format of number.

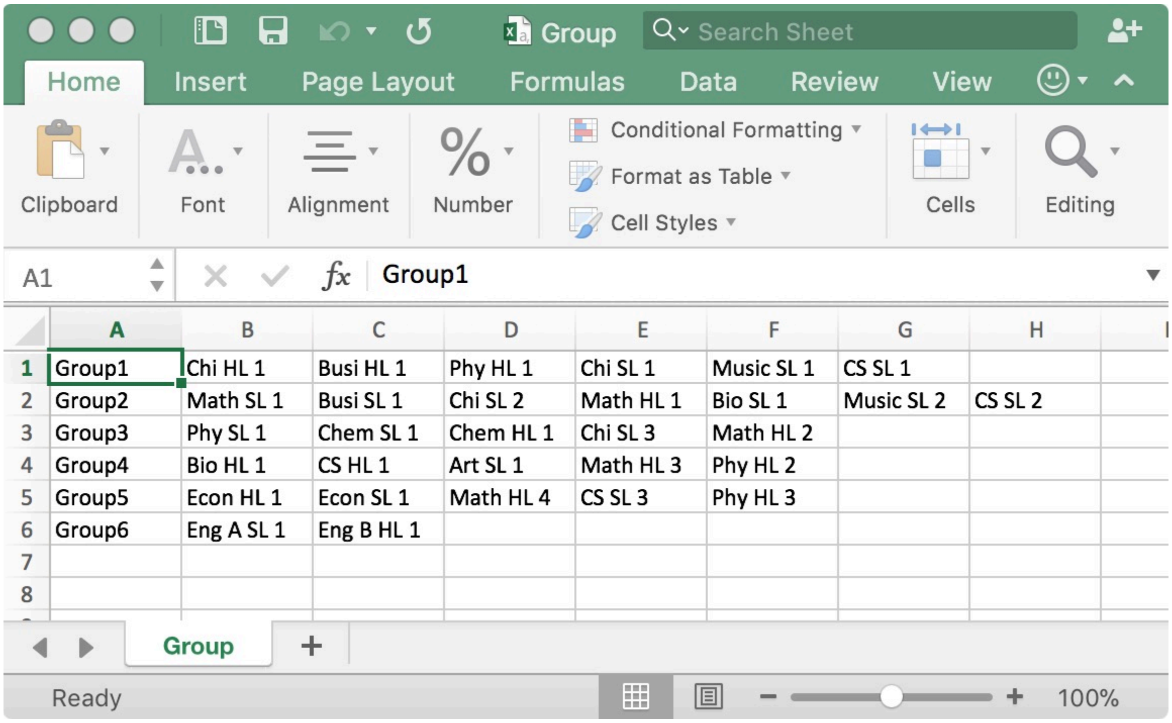
Figure C.15 Exception handling and error message example



```
The class information (class size, class ID, and
student names) classified according to courses.
*** The student have more than one class in a group
%%% The class size is not suitable
&&& Two classes of a subject are in the same group,
but only one teacher for the subject

*************************
Eng A SL
        Class Number 1
*************************
Eng A SL 1
        Class Size 17
        Group 6
        Student Info:
                Emma
                Amy
                Shirley
                Ivy
                Chloe
                Katherine
                Odyssey
                Wendy
                Jessie
                Jerry F
                Stephany
                Richard
                Mandy
                Rachel
                Phil
                Charlie
                Mary
```



```
The class information (class size, class ID, and
student names) classified according to groups.
*** The student have more than one class in a group
%%% The class size is not suitable
&&& Two classes of a subject are in the same group,
but only one teacher for the subject

*************************
GROUP1
TOTAL STUDENT NUMBER 28
*************************
Chi HL 1
        Class Size 5
        Group 1
        Student Info:
                Stephany
                Rita
                Charlie
                Mary
                Wendy
Busi HL 1
        Class Size 6
        Group 1
        Student Info:
                Amy
                Shirley
                Ivy
                Jessie
                Katherine
                Emma
CS HL 1
        Class Size 2
        Group 1
        Student Info:
                Odyssey
                Jerry H
Chi SL 1
        Class Size 2
```



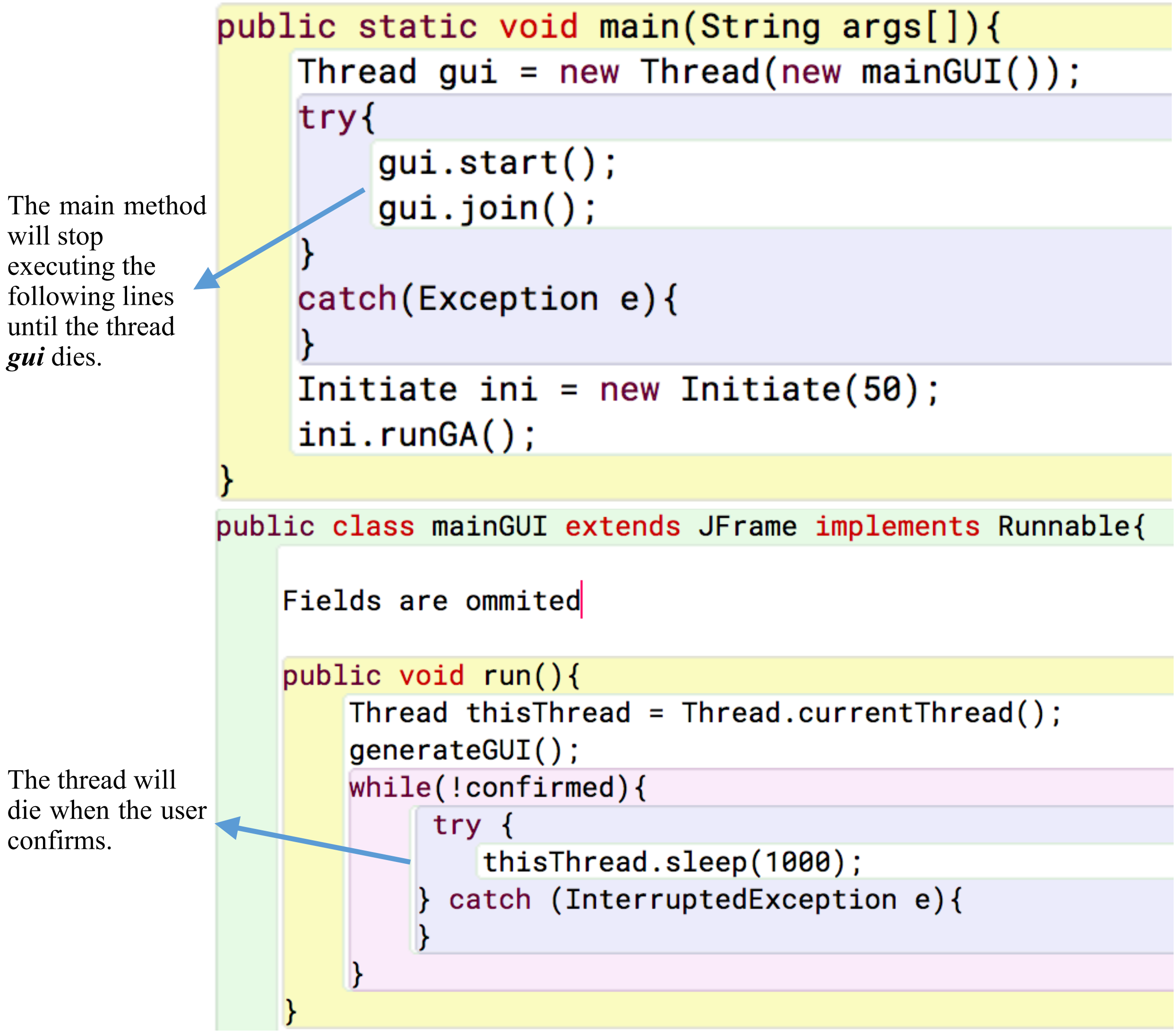
Formula bar: Group1

	A	B	C	D	E	F	G	H
1	Group1	Chi HL 1	Busi HL 1	Phy HL 1	Chi SL 1	Music SL 1	CS SL 1	
2	Group2	Math SL 1	Busi SL 1	Chi SL 2	Math HL 1	Bio SL 1	Music SL 2	CS SL 2
3	Group3	Phy SL 1	Chem SL 1	Chem HL 1	Chi SL 3	Math HL 2		
4	Group4	Bio HL 1	CS HL 1	Art SL 1	Math HL 3	Phy HL 2		
5	Group5	Econ HL 1	Econ SL 1	Math HL 4	CS SL 3	Phy HL 3		
6	Group6	Eng A SL 1	Eng B HL 1					
7								
8								

Figure C.14 Output results (Excerpts)

Multithreading

The system requires several threads, the algorithm that generates the result and GUI windows, to run at the same time. Therefore, ***mainGUI*** class implements ***Runnable***, ***manGUI*** object is used as a thread in the main method. The system will wait for ***mainGUI*** object to run until it dies when the user presses "confirm" and three files are received.[5]



```
public static void main(String args[]){
    Thread gui = new Thread(new mainGUI());
    try{
        gui.start();
        gui.join();
    }
    catch(Exception e){
    }
    Initiate ini = new Initiate(50);
    ini.runGA();
}
```

The main method will stop executing the following lines until the thread ***gui*** dies.

```
public class mainGUI extends JFrame implements Runnable{

    Fields are ommited

    public void run(){
        Thread thisThread = Thread.currentThread();
        generateGUI();
        while(!confirmed){
            try {
                thisThread.sleep(1000);
            } catch (InterruptedException e){
            }
        }
    }
```

The thread will die when the user confirms.

Figure C.16 Implementation of multithreading

[5] thenewboston. "Intermediate Java Tutorial - 27 - What do I look like, a Thread?" *YouTube.* YouTube, 3 May 2010. Web. 15 Aug. 2017.