Andreas Landgrebe
Lucas Hawk
Victor Zheng
Progress Report 1
Computer Science 250
March 30, 2015

So far in the final project of Computer Science 250, we are in the development process of creating a system for class scheduling. In this system, we have 7 java source code algorithms. Class.java has different objects to be used to organize this system the most effective way. In this file, we also get and setters and return these objects to get information about the professor, number of students already enrolled, the information about a course, and classroom that these classes will be held at.

The next file, Classroom.java layouts another part of the class scheduling system. This java file has getter and setter method to be able to return the ID of a student, get the name of a student, and also return the number of seats that a class has to offer.

Course.java is yet another file to be used for the design of this project. This will return the ID and the name of the specific course offered.

Professor.java will include yet more objects to be used. This java file will return the information about objects for the ID of the professor, the classes that the professor is teaching, the maximum number of classes that the professor could possibly teach, and the current amount of classes that a professor is teaching.

The next java file that is in the designed system is Schedule.java. This java file is meant to have students be able to organize their schedule by knowing the specific times that courses are being offered. This will have students be able to add and remove classes from their schedule given the time that it is being offered. We have also added the fitness to determine how well this schedule will work with a student. We still have to discuss how we will write this algorithm to determine how well a proposed schedule will work with a student.

Student.java will return a list of courses that a student is taking.

The last java file in the current proposed project is TimeTabler.java. This file

is our main file to include a number of ArrayList to present all of the information from the objects that have been declared in the past files.

So far, we have organized a system to use genetic algorithms to determine a class schedule between a professor and a student. There is great amount of programming to be done in order to complete this proposed project.

Source Code

Class.java

```
public class Class{
   private final int enrolled;
   private final Professor prof;
   private final Course course;
   private Classroom room;    // Maybe store classroom ref?

   public Class(int enr, Professor theProf, Course theCourse){
      enrolled = enr;  // Maybe initialize as zero, increment as we
          parse students?
      prof = theProf;
      course = theCourse;
   }

   public Professor getProf(){
      return prof;
   }
public class Class{
   private final int enrolled;
   private final Professor prof;
   private final Course course;
   private Classroom room;    // Maybe store classroom ref?

   public Class(int enr, Professor theProf, Course theCourse){
      enrolled = enr;  // Maybe initialize as zero, increment as we
          parse students?
      prof = theProf;
      course = theCourse;
```

```
    }

    public Professor getProf(){
        return prof;
    }

    public int getEnrolled(){
        return enrolled;
    }

    public Course getCourse(){
        return course;
    }

    public Classroom getRoom(){
        return room;
    }

    public void setRoom(Classroom theRoom){
        room = theRoom;
    }

    /* For if we initialize enrolled = 0, increment as we parse students
    public void incEnrolled(){
        enrolled++;
    }
    */

}
    public int getEnrolled(){
```

```
    return enrolled;
  }

  public Course getCourse(){
    return course;
  }

  public Classroom getRoom(){
    return room;
  }

  public void setRoom(Classroom theRoom){
    room = theRoom;
  }

  /* For if we initialize enrolled = 0, increment as we parse students
  public void incEnrolled(){
    enrolled++;
  }
  */

}
```

Classroom.java

```
import java.lang.String;

public class Classroom{
  private final int ID;
  private final String name;    // Probably not needed
  private final int seats;
```

```
   public Classroom(int id, String nam, int seat){
      ID = id;
      name = nam;
      seats = seat;
   }

   public int getID(){
      return ID;
   }

   public String getName(){
      return name;
   }

   public int getSeats(){
      return seats;
   }

}
```

Course.java

```
import java.lang.String;

public class Course{
   private final int ID;
   private final String name;    // Not needed, really

   public Course(int id, String nam){
      ID = id;
```

```
      name = nam;
   }

   public int getID(){
      return ID;
   }

   public String getName(){
      return name;
   }

}
```

Professor.java

```
import java.util.List;

public class Professor{
   private final int ID;
   private final List<Course> taught;
   private final int maxClasses;  // Max classes Prof can teach
   private int currentClasses;     // Current number of classes Prof has

   // Professor with max classes specifies
   public Professor(int id, List<Course> courses, int max){
      ID = id;
      taught = courses;
      maxClasses = max;
   }

   // Professor with max classes not specified - defaults to 3
```

```
   public Professor(int id, List<Course> courses){
      ID = id;
      taught = courses;
      maxClasses = 3;
   }

   public int getId(){
      return ID;
   }

   public List<Course> getCourses(){
      return taught;
   }

   public int getMaxClasses(){
      return maxClasses;
   }

   public int getCurrentClasses(){
      return currentClasses;
   }
}
```

Schedule.java

```
import java.util.Map;
import java.util.HashMap;
import java.util.List;

public class Schedule{
   private Map<Integer, List<Class>> sched = new HashMap<Integer,
```

```
    List<Class>>();
private int fitness = 0; // Fitness of the schedule [initialized to 0]

public Schedule(){
   // Ints 8 thru 17 represent timeslots (8AM - 5PM)
   // Initializes as empty schedule
   for(int i = 8; i <= 17; i++){
      sched.put(i, null);
   }

}

public List<Class> getClassesAt(int timeslot){
   return sched.get(timeslot);
}

public void addClass(int timeslot, Class theClass){
   if(timeslot >= 8 && timeslot <= 17){
      List<Class> classList = sched.get(timeslot);
      classList.add(theClass);

      sched.put(timeslot, classList);
   }
   //else (?)
}

public void addClass(int timeslot, List<Class> classes){
   if(timeslot >= 8 && timeslot <= 17){
      sched.put(timeslot, classes);
   }
```

```
    }

    public void removeClass(int timeslot, Class theClass){
        if(timeslot >= 8 && timeslot <= 17){
            List<Class> classList = sched.get(timeslot);
            classList.remove(theClass);

            sched.put(timeslot, classList);
        }
    }

    public void removeClass(int timeslot, List<Class> classes){
        if(timeslot >= 8 && timeslot <= 17){
            classes.remove(classes);

            sched.put(timeslot, classes);
        }
    }

    public void calcFitness(ArrayList <Professor> profs, ArrayList
         <Class> classes, ArrayList <Student> students){
        // Equation to calc fitness to be determined
    }

    public int getFitness(){
        return fitness;
    }

}
```

Student.java

```
public class Student{
   private Course [] courses = new Course[4];

   public Student(){
      // Empty constructor ?
   }

   public Course[] getCourses(){
      return courses;
   }
}
```

TimeTabler.java

```
//-------------------------------------
// Class contains the genetic algorithm to
// create solutions for the given input
//-------------------------------------
import java.util.*;
import java.io.*;

public class TimeTabler{

   Schedule [] schedules = new Schedule [100]; // random solutions
   ArrayList<Professor> professors = new ArrayList<Professor>();
   ArrayList<Class> classes = new ArrayList<Class>();
   ArrayList<Student> students = new ArrayList<Student>();
   ArrayList<Course> courses = new ArrayList<Course>();
   ArrayList<Classroom> rooms = new ArrayList<Classroom>();
```

```
public static void main (String [] args){
   double mutationProb = 0.1;    // Chance for any solution to
       mutate? [0,1]
   double fitThreshold = 0.95;   // Necessary fitness to end
       evolution [0,1]
   int numKilled = 20;        // Number of solutions killed/reborn
       each generation
   int numParents = 10;     // Number of top solutions to use for
       crossover
   int maxGens = 100;         // Maximum number of evolutions (will
       quit after this many regardless of fit)
   Random r = new Random();   // Random doubles for random mutation

   createObjects(input.xml);  // Creates objects by parsing xml

   // Generates schedules
   for(int i = 0; i < schedules.length; i++){
      schedules[i] = generateRandomSchedule();  // Generates random
          schedule based on all objects
   }

   // Evolve until fitness threshold is met
   int genCounter = 0;   // Tracks num of generations
   while(schedule[0].getFitness() < fitThreshold && genCounter <
       maxGens){
      genCounter ++;     // Increment generation counter

      // Calculate fitness of every solution
      for(int i = 0; i < schedules.length; i++){
```

```
            schedules[i].calcFitness(professors, classes, students);
        }

        sortByFitness();        // Sorts schedules by fitness

        // perform crossover for all survivors to next generation
        for(int i = 0; i < schedules.length - numKilled; i ++){
            schedules[i] = performCrossover(schedules[i]);
        }

        // Kill worst solutions and create new ones
        for(int i = schedules.length - numEvolved - 1; i < 100; i++){
            schedules[i] = generateChildren(); // Random crossover from
                top solutions
        }

        // Perform random mutation on schedules
        for(int i = 0; i < schedules.length; i++){
            if(r.nextDouble() <= mutationProb){
                mutate(schedules[i]);
            }
        }
    }

    // Hooray, we're done!

}

//-------------------------
// Creates all objects from xml(?) file using SAX(?)
```

```
    //--------------------------
    public static void createObjects(File input){
        // Create profs, classes, etc.
    }

    public static void mutate(Schedule schedule){
        // perform random mutation
    }

    public static Schedule generateChildren(){
        // Replaces worst solutions with children of top solutions
    }

    public static Schedule performCrossover(Schedule schedule){
        // Create new schedule thru crossover with old schedule and random
            top solution
    }

    public static Schedule generateRandomSchedule(){
        // Generate the random schedule
    }

    public static void sortByFitness(){
        // Sort schedules by fitness
    }


}

/*
```

```
- Parse input file to generate all objects
- Generate random schedules
while(fitness < ?){
- Calculate fitness of all
- Sort by fitness  (What algo?)
- Replace bottom 20 (?) with new solutions generated from crossover of
    top 20 (?)
- Random mutation (?)
}
- Most fit is best solution
```
