COMSATS University, Islamabad

**Project**

**Title: Time Table Generator**

Course: Object Oriented Programming



Submitted By: Saima Kausar (FA19-BCS-070)

Yousaf Raza (FA19-BCS-088)

Submitted To: Mam Saneeha Amir

Date: 1 January, 2021

Department of Computer Science,

Islamabad Campus

**Project Description:**

Time table is generated by assigning courses and classes to slots present in room.

Classes and Rooms are present with free slots available. Slots are present from start to end time of university. Each class has maximum 10 number of courses. On same day and time, a teacher and class can’t have more than one class.

Room capacity that is nearest to largest class strength is assigned to that class so that room space is not wasted. If class strength is larger than capacity of rooms then that slot remain unassigned.

# Class Class

## Member Variables:

Class member variable includes

## name: A string variable that stores the name of class.

## strength: A integer variable that stores the strength of class.

## active: Boolean variable that tells whether class is occupied or not

## courses: An array of type courses that stores all the courses of class. Maximum no of courses in a class are 10.

## courseCount: A integer variable that stores no of courses in a class.

## classCount: Static variable of type int which tells about no of classes.

## slots: An array of type room slots

## classSlots: An array of type class slots

## classSlotCount: A variable of type int that stores no of class slots available

## Member Functions:

Member functions of class includes constructors, and getter and setters for the member variables. It also includes functions:

* addClass(): Return true if class is added and false otherwise
* removeClass(): Return true if class is removed successfully
* editClass(): Return true if class is edited successfully
* addCourse(): Return true if course is added successfully
* removeCourse(): Return true if course is removed successfully
* makeSlots(): Make Slots For Classes
* resetSlots(): Resetting Class Slots

# Class ClassSlot

## Member Variables:

ClassSlot class member variable includes

* sClass: A variable of type Class that stores class taken in class slot.
* course: A variable of type course that stores the course studied in class slot.
* classNo: A variable of type int that stores class number.
* used: A variable of type boolean that tells whether a class slot is taken or not.

## Member Functions:

Member functions of class ClassSlot includes constructors, and getter and setters for the member variables.

# Class Course

## Member Variables:

Course class member variable includes

* name: A string variable that stores the name of course.
* active: A boolean variable that tells whether a class is ongoing.
* teacher: A string variable that stores the course teacher name.
* weeklyClassCount: A integer variable that stores number of classes of course in a week.

## Member Functions:

Member functions of class Course includes constructors, and getter and setters for the member variables.

# Class Room

## Member Variables:

Room class member variable includes

* name: A string variable that stores the name of room.
* capacity: A integer variable that stores the capacity of room.
* roomCount: A Static variable of type int that stores number of rooms.
* startTime: A variable of type Time that stores start time of room slot.
* endTime: A variable of type Time that stores end time of room slot.
* roomSlots: An array of type RoomSlot that stores room slots present in room.
* slotCount: A variable of type int that stores number of slots in room.

## Member Functions:

Member functions of class Room includes constructors, and getter and setters for the member variables. It also includes functions:

* addRoom(): Return true if room is added and false otherwise
* removeRoom(): Return true if room is removed successfully
* editRoom(): Return true if room is edited successfully
* extendSlots(): Increasing Slots for Room
* makeSlots(): Making Slots for Room
* resetSlots(): Resetting Room Slots

# Class RoomSlot

## Member Variables:

Member variable of the class RoomSlot includes

* name: A string variable that stores the name of room.
* capacity: A integer variable that stores the capacity of room.
* day: A integer variable that stores day.
* startTime: A variable of type Time that stores start time of room slot.
* endTime: A variable of type Time that stores end time of room slot.
* course: A variable of type course that stores the course to be studied in room slot.
* sClass: A variable of type Class that stores class taken in room slot.

## Member Functions:

Member functions of class RoomSlot includes constructors, and getter and setters for the member variables and a addClass function which is use to add Class and Course to Room Slot.

# Class TimeTableSpace

## Member Variables:

Member variable of the class TimeTableSpace includes

* name: A string variable that stores the name of space.
* autoSaveEnabled: A boolean variable that tells if auto save is enabled or not.
* classes: An array of type Class that stores all data of classes.
* Rooms: An array of type rooms that stores all data of rooms.

## Member Functions:

Member functions of class TimetableSpace includes constructors, and getter and setters for the member variables. It also includes functions:

* saveSpace(): return true if room and classes are write to file
* loadSpace(): return true if rooms and classes are readied from file
* saveClasses(): return true if all classes are write in file
* saveRooms(): return true if all rooms are write in file
* loadClasses(): return true if classes data is readied from file
* loadRooms(): return true if rooms data is readied from file
* addClass(): return true if class is added
* removeClass(): return true if class is removed
* addRoom(): return true if room is added
* removeRoom(): return true if class is removed

# Class TimeTableCreator

## Member Variables:

Member variable of the class TimeTableCreator includes

* roomSlots: An array of type RoomSlot that stores room slots that are assigned.
* remainingRoomSlots: An array of type RoomSlot that stores room slots that need to be assigned.
* classSlots: : An array of type ClassSlot that stores class slots that are assigned.
* remainingClassSlots: An array of type ClassSlot that stores class slots that need to be assigned.
* roomSlotCount: A int variable that stores number of room slots.
* classSlotCount: A int variable that stores number of class slots.
* roomCount: A integer variable that stores number of rooms.
* dailyClassesPerRoom: A integer variable that store number of classes daily in room.

## Member Functions:

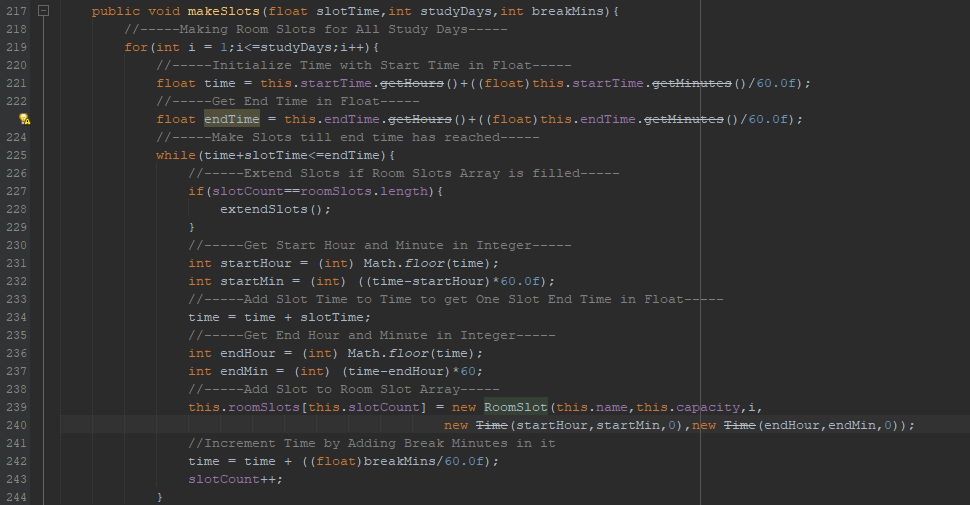
Member functions of class TimeTableCreator includes constructors, and getter and setters for the member variables and a makeTimeTable function which is use to make time table.

# Relations between classes

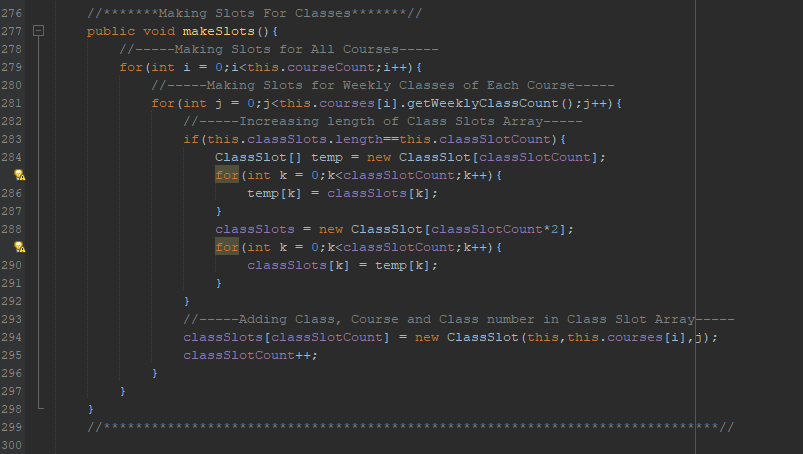
All Classes implement serializable interface.

* Composition between class Course and Class
* Composition between class ClassSlot and Class
* Composition between class RoomSlot and Class
* Composition between class ClassSlot and Course
* Composition between class Room and RoomSlot
* Composition between class RoomSlot and Course
* Composition between class TimeTableSpace and Class
* Composition between class TimeTableSpace and Room
* Composition between class TimeTableCreator and ClassSlot
* Composition between class TimeTableCreator and RoomSlot

**Making Room Slots:**

****

**Making Class Slots:**

****

**Time Table Creation:**

 //==========================

//| CREATING TIME TABLE |

//==========================

public class TimeTableCreator {

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*DATA MEMBERS\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

RoomSlot[] roomSlots = new RoomSlot[100];

RoomSlot[] remainingRoomSlots = new RoomSlot[100];

ClassSlot[] classSlots = new ClassSlot[100];

ClassSlot[] remainingClassSlots = new ClassSlot[100];

int roomSlotCount;

int classSlotCount;

int roomCount;

int dailyClassesPerRoom;

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*CONSTRUCTORS\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

//--------ARGUMENT CONSTRUCTOR--------//

public TimeTableCreator(TimeTableSpace space, GuiMain gui) {

//-----Getting number of Rooms-----

roomCount = Room.roomCount;

//-----Get start, end, duration entered in text fields in float-----

float start = Integer.parseInt(gui.startTimeHourField.getText())+(Integer.parseInt(gui.startTimeMinField.getText())/60.0f);

float end = Integer.parseInt(gui.endTimeHourField.getText())+(Integer.parseInt(gui.endTimeMinField.getText())/60.0f);

float duration = Integer.parseInt(gui.durationHourField.getText())+(Integer.parseInt(gui.durationMinField.getText())/60.0f);

//-----Get Number of Classes in Room-----

dailyClassesPerRoom = (int)((end - start)/duration);

//-----Room and Class Slots are Initialized 0-----

roomSlotCount = 0;

classSlotCount = 0;

//-----Read all Classes from file-----

Class[] classes = space.getClasses();

//-----Loop till Number of classes-----

for(int i = 0;i<Class.classCount;i++){

//-----Make Slots of Classes-----

classes[i].makeSlots();

//-----Assign all class slots to class slots and remaining class slots at start-----

for(int j = 0;j<classes[i].getClassSlotCount();j++){

//-----Increasing Class Slot Count array length if filled-----

if(classSlotCount==classSlots.length){

ClassSlot[] temp = new ClassSlot[classSlotCount];

for(int k = 0;k<classSlotCount;k++){

temp[k] = classSlots[k];

}

classSlots = new ClassSlot[classSlotCount\*2];

remainingClassSlots = new ClassSlot[classSlotCount\*2];

for(int k = 0;k<classSlotCount;k++){

classSlots[k] = temp[k];

remainingClassSlots[k] = temp[k];

}

}

classSlots[classSlotCount] = classes[i].getClassSlots()[j];

remainingClassSlots[classSlotCount] = classes[i].getClassSlots()[j];

classSlotCount++;

}

}

//-----Read all Rooms from file-----

Room[] rooms = space.getRooms();

//-----Geeting time fo reach slot by reading text field of duration-----

float slotTime = Integer.parseInt(gui.durationHourField.getText())+(Integer.parseInt(gui.durationMinField.getText())/60.0f);

//-----Loop till Number of Rooms-----

for(int i = 0;i<Room.roomCount;i++){

//-----Get start and end time entered in text fields and make slots using duration, day text fields-----

rooms[i].setStartTime(new Time(Integer.parseInt(gui.startTimeHourField.getText()),Integer.parseInt(gui.startTimeMinField.getText()),0));

rooms[i].setEndTime(new Time(Integer.parseInt(gui.endTimeHourField.getText()),Integer.parseInt(gui.endTimeMinField.getText()),0));

rooms[i].makeSlots(slotTime,Integer.parseInt(gui.daysField.getText()),0);

//-----Assign all room slots to room slots and remaining room slots at start-----

for(int j = 0;j<rooms[i].getSlotCount();j++){

//-----Increasing Room Slot Count array length if filled-----

if(roomSlots.length==roomSlotCount){

RoomSlot[] temp = new RoomSlot[roomSlotCount];

for(int k = 0;k<roomSlotCount;k++){

temp[k] = roomSlots[k];

}

roomSlots = new RoomSlot[roomSlotCount\*2];

remainingRoomSlots = new RoomSlot[roomSlotCount\*2];

for(int k = 0;k<roomSlotCount;k++){

roomSlots[k] = temp[k];

remainingRoomSlots[k] = temp[k];

}

}

roomSlots[roomSlotCount] = rooms[i].getSlots()[j];

remainingRoomSlots[roomSlotCount] = rooms[i].getSlots()[j];

roomSlotCount++;

}

}

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

//\*\*\*\*\*\*\*MAKING TIME TABLE\*\*\*\*\*\*\*//

public void makeTimeTable(){

//-----Variable to determine unchecked slots of classes-----

int uncheckedClassSlots = this.classSlotCount;

//-----Loop Till All Slots of Classes are Filled-----

while (uncheckedClassSlots>0){

//-----Finding lagrest strength and it's index from all classes-----

int largestClassStrength = -1;

int largestClassIndex = -1;

//-----Loop Till Number of Slots of Classes-----

for(int i = 0;i<classSlotCount;i++){

//-----If Class Slot is present and class strength is greater than previous strength and space for that slot is present in room (slot is used) then update largest strength-----

if(remainingClassSlots[i]!=null&&remainingClassSlots[i].getsClass().getStrength()>largestClassStrength&&!remainingClassSlots[i].isUsed()){

largestClassStrength = remainingClassSlots[i].getsClass().getStrength();

largestClassIndex = i;

}

}

//-----Finding room capacity which is nearest to largest strength so not to waste room space-----

int smallestCompatibleRoomCapacity = Integer.MAX\_VALUE;

int smallestCompatibleRoomIndex = Integer.MAX\_VALUE;

//-----If Class Strngth is not -1-----

if(largestClassStrength != -1){

//-----Loop Till Number of Slots of Rooms-----

for(int i = 0;i<roomSlotCount;i++){

//-----If Room Slot is present and Room Capacity is greater than Class strength and room capacity is more nearer to previous one-----

if(remainingRoomSlots[i]!=null&&remainingRoomSlots[i].getCapacity()>=largestClassStrength&&remainingRoomSlots[i].getCapacity()<smallestCompatibleRoomCapacity){

boolean repetition = false;

Time timeCompare = remainingRoomSlots[i].getStartTime();

int dayCompare = remainingRoomSlots[i].getDay();

//-----On same time and day a teacher and class can't have two classes-----

for(int j = 0;j<roomSlotCount;j++){

if(roomSlots[j].getDay() == dayCompare&&roomSlots[j].getStartTime().equals(timeCompare)){

if(roomSlots[j].getsClass()!=null&&(roomSlots[j].getsClass().getName().equals(classSlots[largestClassIndex].getsClass().getName())||roomSlots[j].getCourse().getTeacher().equals(classSlots[largestClassIndex].getCourse().getTeacher()))){

repetition = true;

break;

}

}

}

//-----If no repetition in time and day for teacher and class assign the capacity of room to be compatible for class-----

if(!repetition){

smallestCompatibleRoomCapacity = remainingRoomSlots[i].getCapacity();

smallestCompatibleRoomIndex = i;

}

}

}

}

//-----If class has no strength then break-----

else{

break;

}

//-----If a room with compatible space is found assign class and course to it and decrement un checked class slots-----

if(smallestCompatibleRoomCapacity!=Integer.MAX\_VALUE){

roomSlots[smallestCompatibleRoomIndex].addClass(remainingClassSlots[largestClassIndex].getsClass(),remainingClassSlots[largestClassIndex].getCourse());

remainingRoomSlots[smallestCompatibleRoomIndex] = null;

remainingClassSlots[largestClassIndex] = null;

uncheckedClassSlots--;

}

//-----If no room with compatible space is available then that slot is unused-----

else{

remainingClassSlots[largestClassIndex].setUsed(true);

}

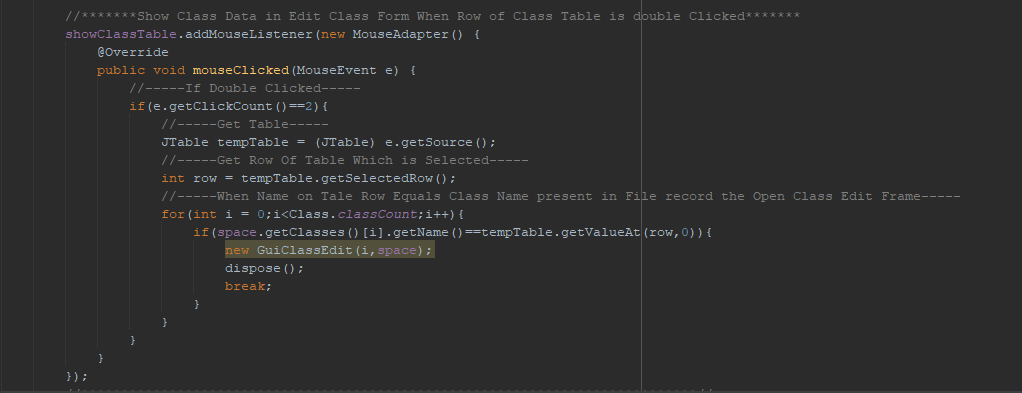
}

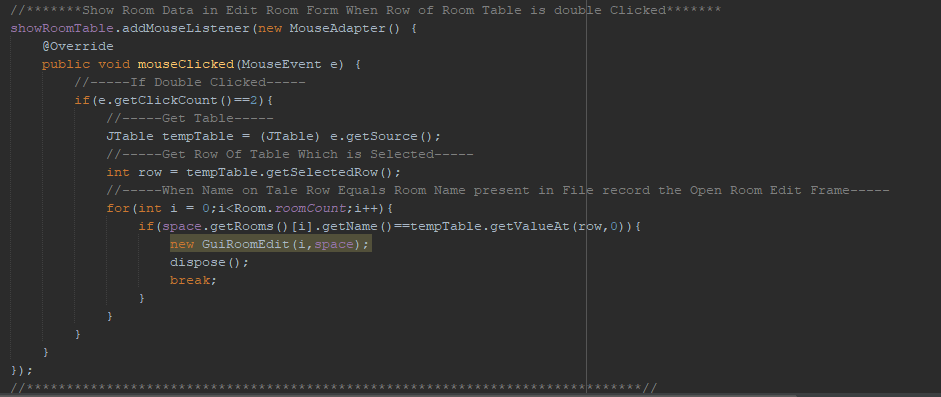
}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

}

**GUI Listeners on Room, Class Tables:**

****



**Showing Time Table:**

//\*\*\*\*\*\*\*Changing Time Table For Each Day\*\*\*\*\*\*\*

public void changeTimeTableDay(int day,TimeTableCreator timeTableCreator,TimeTableSpace space){

//-----Columns of Time Table Will Represent Rooms-----

String[] columns = new String[timeTableCreator.getRoomCount()+1];

//-----Add Time in first Column-----

columns[0] = "Time";

//-----Add Rooms Name to Columns-----

for(int i = 1;i<timeTableCreator.getRoomCount()+1;i++){

columns[i] = space.getRooms()[i-1].getName();

}

//-----Add data to Time Table-----

String[][] data = new String[timeTableCreator.getDailyClassesPerRoom()][timeTableCreator.getRoomCount()+1];

//-----Get Start Time for Classes-----

for(int i = 0;i<timeTableCreator.getDailyClassesPerRoom();i++){

data[i][0] = timeTableCreator.getRoomSlots()[i].getStartTime()+"";

}

//-----Alloat Classes to all Slots Available in Room-----

for(int i = 0;i<timeTableCreator.getRoomSlotCount();i++){

//-----If day matches the day of room slot-----

if(timeTableCreator.getRoomSlots()[i].getDay()==day){

//-----Set row where data to be added at -1-----

int horizontal = -1;

//-----Set row where Room name is equal to any of the room name present in record-----

for(int j = 0;j<timeTableCreator.getRoomCount()+1;j++){

if(timeTableCreator.getRoomSlots()[i].getName().equals(columns[j])){

horizontal = j;

}

}

//-----Set column where data to be added at -1-----

int vertical = -1;

//-----Set coloumn where start time matches with start time of room slot present in record-----

for(int j = 0;j<timeTableCreator.getDailyClassesPerRoom();j++){

if(data[j][0].equals(timeTableCreator.getRoomSlots()[i].getStartTime().toString())){

vertical = j;

}

}

//-----If specific row, column is not get start the loop again to find correct place for data entry-----

if(vertical==-1||horizontal==-1){

continue;

}

//-----If Class and Room is present add data at specific location in table-----

if(timeTableCreator.getRoomSlots()[i].getClass()!=null && timeTableCreator.getRoomSlots()[i].getCourse()!=null){

data[vertical][horizontal] = "Class: "+timeTableCreator.getRoomSlots()[i].getsClass().getName()+" , Course: "+timeTableCreator.getRoomSlots()[i].getCourse().getName()+" , Teacher: "+timeTableCreator.getRoomSlots()[i].getCourse().getTeacher();

}

//-----If Class or room is not present leave that cell as empty-----

else{

data[vertical][horizontal] = "";

}

}

}

//-----Set Time Table Model to not editable-----

DefaultTableModel tableModel = new DefaultTableModel(data,columns){

@Override

public boolean isCellEditable(int row, int column) {

return false;

}

};

timeTableToShow.setModel(tableModel);

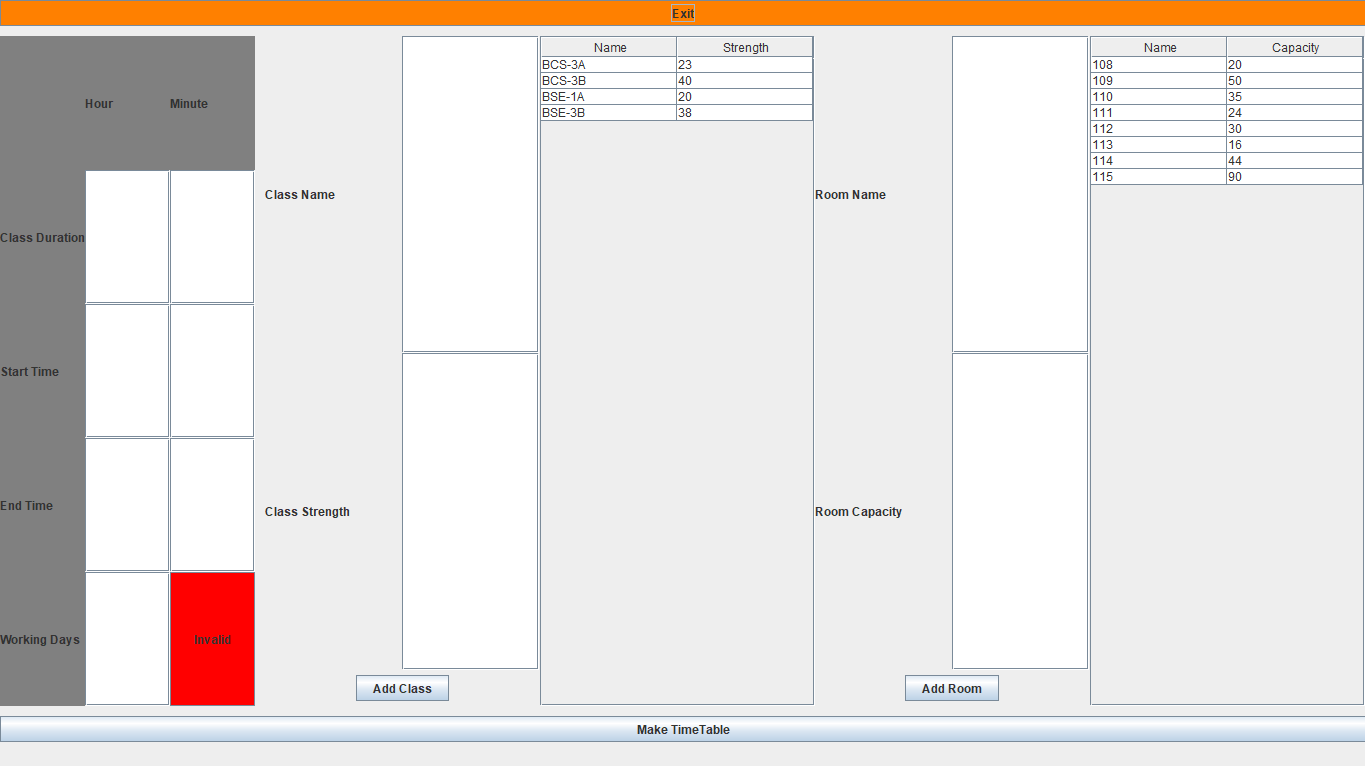
}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

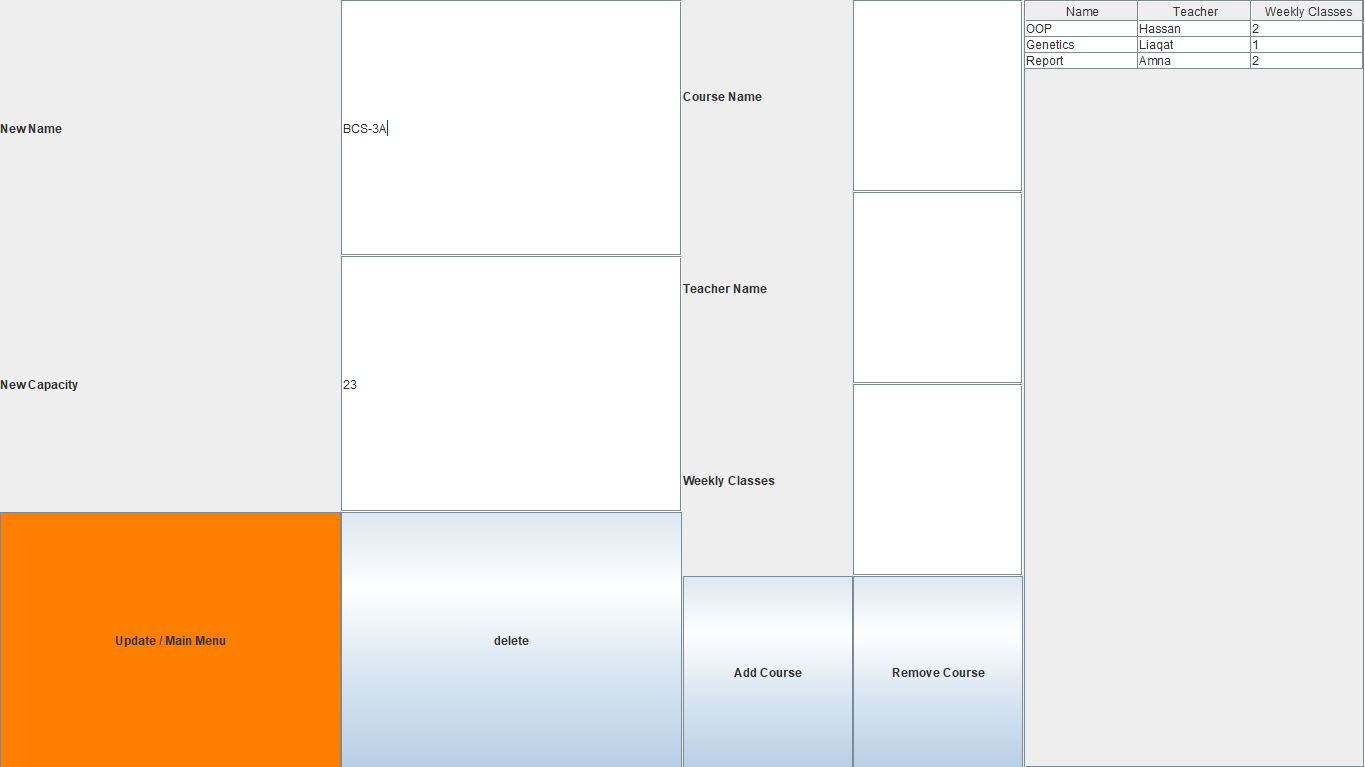
**GUI Output:**

Double click on table to add, remove or update specific Room, Class.

**Main** is



By **double clicking on any row in class table**, we get form like this



On clicking Make Time Table we have **Time Table** as

