```
import java.sql.*;
import oracle.jdbc.*;
import java.security.MessageDigest;
import java.io.*;
import psb.*;

// s04-cs389-s20/8Qi5Ys10@DELPHI

/**
 * This class handles all interaction with the database.
 *
 * @author      Viktor Geller
 * @version     %I%, %G%
 * @since       JDK1.4
 */

public class Database {

    public static void main(String[] args) {
        Database d = new Database();
        System.out.println(d.login("admin","psb2004"));
        System.out.println(d.encrypt("psb2004"));
    }

    public Database() {

        }

    /**
     * Gets largest Schedule ID.
     */
    public String getNextSchedId() throws Exception {
        String ret;
        Class.forName("oracle.jdbc.driver.OracleDriver");
        Connection conn =
DriverManager.getConnection("jdbc:oracle:thin:@DELPHI:1521:ORA1","s04-cs389-s20","8Qi5Ys10");
        Statement stmt = conn.createStatement();
        String strSQL = "select max(scheduleid) as mid from schedule";
        ResultSet rs = stmt.executeQuery (strSQL);

        if (rs.next() && rs.getString("mid") != null) ret = String.valueOf(Integer.parseInt(rs.getString("mid")) + 1);
        else ret = "0";
        stmt.close();
        conn.close();
        return ret;

    }

    /**
     * Deletes a student. Used for administrator functionality.
     */
    public void deleteStudent(String uname) {
        try {
        Class.forName("oracle.jdbc.driver.OracleDriver");
        Connection conn =
DriverManager.getConnection("jdbc:oracle:thin:@DELPHI:1521:ORA1","s04-cs389-s20","8Qi5Ys10");
        Statement stmt = conn.createStatement();
        stmt.executeUpdate ("delete from student where login = '" + uname + "'");
        stmt.close();
        stmt = conn.createStatement();
        stmt.executeUpdate ("commit");
        stmt.close();
        conn.close();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    /**
```



```
     * Gets a list of students. Used for administrator functionality.
     */
    public Students getStudentList() throws Exception {
        Students ret = new Students();
        try {
            Class.forName("oracle.jdbc.driver.OracleDriver");
            Connection conn =
DriverManager.getConnection("jdbc:oracle:thin:@DELPHI:1521:ORA1","s04-cs389-s20","8Qi5Ys10");
            Statement stmt = conn.createStatement();
            String strSQL = "select * from Student";
            ResultSet rs = stmt.executeQuery (strSQL);
            while(rs.next()) {
                ret.add(new Student(rs.getString("login"),rs.getString("password")));

            }
            stmt.close();
            conn.close();
        } catch (Exception e) {
            e.printStackTrace();
        }
        return ret;
    }

    /**
     * Gets the Days/Times of when a student does NOT want to have any classes
     * on.
     */
    public MeetingTimes getRestrictedTimesForStudent(String uname) {
        MeetingTimes ret = new MeetingTimes();
        try {
            Class.forName("oracle.jdbc.driver.OracleDriver");
            Connection conn =
DriverManager.getConnection("jdbc:oracle:thin:@DELPHI:1521:ORA1","s04-cs389-s20","8Qi5Ys10");
            Statement stmt = conn.createStatement();
            String strSQL = "select * from StudentPreferencesTimeDetails where login = '" + uname + "'";
            ResultSet rs = stmt.executeQuery (strSQL);
            while(rs.next()) {
                ret.add(new MeetingTime(rs.getString("daysofweek").charAt(0),new psb.Time(rs.getInt("starttime")), new
psb.Time(rs.getInt("endtime"))));

            }
            stmt.close();
            conn.close();
        } catch (Exception e) {
            e.printStackTrace();
        }
        return ret;
    }

    /**
     * Returns Student information.
     */
    public Student getStudent(String uname) {
        try {
            Class.forName("oracle.jdbc.driver.OracleDriver");
            Connection conn =
DriverManager.getConnection("jdbc:oracle:thin:@DELPHI:1521:ORA1","s04-cs389-s20","8Qi5Ys10");
            Statement stmt = conn.createStatement();
            String strSQL = "select * from Student where login = '" + uname + "'";
            ResultSet rs = stmt.executeQuery (strSQL);
            Student s = null;
            while(rs.next()) {
                s = new
Student(rs.getString("login"),rs.getString("password"),rs.getString("firstname"),rs.getString("lastname"),rs.getString("email"));
                Profile p = new Profile(rs.getString("timeofday"),rs.getString("gapsize"),rs.getString("classlength"));
                p.setRestrictedTimes(getRestrictedTimesForStudent(uname));
                s.setProfile(p);
            }

            stmt.close();
```

```
            conn.close();
            return s;
        } catch (Exception e) {
            e.printStackTrace();
        }
        return null;
    }

    /**
    * Returns meeting times for a particular course.
    */
    public MeetingTimes getMeetingTimesForCrn(String crn)  throws Exception {
        System.out.println("getMeetingTimesForCrn(" + crn + ")");
        MeetingTimes mt = new MeetingTimes();

        Class.forName("oracle.jdbc.driver.OracleDriver");
        Connection conn =
DriverManager.getConnection("jdbc:oracle:thin:@DELPHI:1521:ORA1","s04-cs389-s20","8Qi5Ys10");
        Statement stmt = conn.createStatement();
        String strSQL = "select * from MEETING_TIMES where crn = '" + crn + "'";
        ResultSet rs = stmt.executeQuery (strSQL);
        while(rs.next()) {
            MeetingTime m = new MeetingTime(rs.getString("day").charAt(0),new psb.Time(rs.getInt("start_time")), new
psb.Time(rs.getInt("end_time")));
            mt.add(m);
        }
        stmt.close();
        conn.close();

        return mt;
    }

    /**
    * Returns course for a particular CRN.
    */
    public Course getCourseForCrn(String crn) throws Exception {
        System.out.println("getCourseForCrn(" + crn + ")");
        Class.forName("oracle.jdbc.driver.OracleDriver");
        Connection conn =
DriverManager.getConnection("jdbc:oracle:thin:@DELPHI:1521:ORA1","s04-cs389-s20","8Qi5Ys10");
        Statement stmt = conn.createStatement();
        String strSQL = "select * from CLASS where crn = '" + crn + "'";
        ResultSet rs = stmt.executeQuery (strSQL);
        while(rs.next()) {
            Course c = new Course(rs.getString("crn"),
                        rs.getString("coursename"),
                        rs.getString("fullcoursename"),
                        rs.getString("status"),
                        getMeetingTimesForCrn(crn),
                        rs.getString("room"),
                        rs.getString("site"),
                        rs.getString("fee"),
                        rs.getString("remark"),
                        rs.getString("professor"));
            stmt.close();
            conn.close();
            System.out.println(c);
            return c;
        }
        return null;
    }

    /**
    * Returns all courses for a particular schedule.
    */
    public Courses getCourseListForSchedule(String scheduleid) {
        System.out.println("getCourseListForSchedule(" + scheduleid + ")");
        Courses ret = new Courses();
        try {
            Class.forName("oracle.jdbc.driver.OracleDriver");
```



```
            Connection conn =
DriverManager.getConnection("jdbc:oracle:thin:@DELPHI:1521:ORA1","s04-cs389-s20","8Qi5Ys10");
            Statement stmt = conn.createStatement();
            String strSQL = "select crn from schedule_crn where scheduleid = '" + scheduleid + "' order by crn";
            ResultSet rs = stmt.executeQuery (strSQL);
            while(rs.next()) {
                ret.add(getCourseForCrn(rs.getString("crn")));

            }
            stmt.close();
            conn.close();
        } catch (Exception e) {
            e.printStackTrace();
        }
        return ret;
    }

    /**
    * Gets a list of saved schedules for a particular user.
    */
    public Schedules getSavedScheduleList(String uname) {
        System.out.println("getSavedScheduleList(" + uname + ")");
        Schedules ret = new Schedules();
        int count=0;
        try {
            Class.forName("oracle.jdbc.driver.OracleDriver");
            Connection conn =
DriverManager.getConnection("jdbc:oracle:thin:@DELPHI:1521:ORA1","s04-cs389-s20","8Qi5Ys10");
            Statement stmt = conn.createStatement();
            String strSQL = "select scheduleid, semester from schedule where login = '" + uname + "' order by scheduleid";
            ResultSet rs = stmt.executeQuery (strSQL);
            while(rs.next()) {
                Schedule s = new Schedule();
                s.setCourseList(getCourseListForSchedule(rs.getString("scheduleid")));
                s.setSchedNumber(Integer.parseInt(rs.getString("scheduleid")));
                s.setSemester(rs.getString("semester"));
                count++;
                ret.add(s);
            }

            stmt.close();
            conn.close();

            }catch (Exception e) {
            e.printStackTrace();
        }

        System.out.println(ret.size()+ "}}}}}}}}}}}");
        return ret;
    }

    /**
    * Deletes a course from saved schedule. Used in "Edit schedule" mode.
    */
    public void deleteCrnFromSchedule(String scheduleid, String crn) {
        try {
        Class.forName("oracle.jdbc.driver.OracleDriver");
        Connection conn =
DriverManager.getConnection("jdbc:oracle:thin:@DELPHI:1521:ORA1","s04-cs389-s20","8Qi5Ys10");
        Statement stmt = conn.createStatement();
        stmt.executeUpdate ("delete from schedule_crn where scheduleid = '" + scheduleid + "' and crn = '" + crn + "'");
        stmt.close();
        stmt = conn.createStatement();
        stmt.executeUpdate ("commit");
        stmt.close();
        conn.close();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
```

```
    /**
    * Deletes a schedule. Used in "My Schedules" mode.
    */
    public void deleteSchedule(String scheduleid) {
        try {
        Class.forName("oracle.jdbc.driver.OracleDriver");
        Connection conn =
DriverManager.getConnection("jdbc:oracle:thin:@DELPHI:1521:ORA1","s04-cs389-s20","8Qi5Ys10");
        Statement stmt = conn.createStatement();
        stmt.executeUpdate ("delete from schedule where scheduleid = '" + scheduleid + "'");
        stmt.close();
        stmt = conn.createStatement();
        stmt.executeUpdate ("delete from schedule_crn where scheduleid = '" + scheduleid + "'");
        stmt.close();
        stmt = conn.createStatement();
        stmt.executeUpdate ("commit");
        stmt.close();
        conn.close();
        } catch (Exception e) {
            e.printStackTrace();
        }

    }

    /**
    * Saves a schedule.
    */
    public void saveSchedule(Schedule s, String uname) {
        try {
        Class.forName("oracle.jdbc.driver.OracleDriver");
        Connection conn =
DriverManager.getConnection("jdbc:oracle:thin:@DELPHI:1521:ORA1","s04-cs389-s20","8Qi5Ys10");
        Statement stmt = conn.createStatement();
        Courses cs = s.getCourseList();

        String scheduleId = getNextSchedId();

        stmt.executeUpdate("insert into schedule values ('" + scheduleId + "','" + uname + "','" + s.getSemester() + "')");
        stmt.close();
        stmt = conn.createStatement();

        for (int i=0; i< cs.size();i++) {
            Course c = (Course) cs.get(i);
            stmt.executeUpdate ("delete from CLASS where crn = '" + c.getCrn() + "'");
            stmt.close();
            stmt = conn.createStatement();
            System.out.println("0");
            stmt.executeUpdate ("delete from MEETING_TIMES where crn = '" + c.getCrn() + "'");
            stmt.close();
            stmt = conn.createStatement();

            stmt.executeUpdate("insert into schedule_crn values ('" + scheduleId + "','" + c.getCrn() + "')");
            stmt.close();
            stmt = conn.createStatement();

            System.out.println("1");
            System.out.println("insert into CLASS values('" + c.getCrn() + "','" + c.getCourse() + "','" + c.getTitle() + "','" +
c.getStatus() + "','" + c.getInstructor() + "','" + c.getRoom() + "','" + c.getSite() + "','" + c.getFee() + "','" + c.getRemark() +
"',0)");
            stmt.executeUpdate ("insert into CLASS values('" + c.getCrn() + "','" + c.getCourse() + "','" +
c.getTitle().replaceAll("&","And").replaceAll("'","") + "','" + c.getStatus() + "','" + c.getInstructor() + "','" + c.getRoom() + "','" +
c.getSite() + "','" + c.getFee() + "','" + c.getRemark() + "',0)");
            stmt.close();
            stmt = conn.createStatement();

            System.out.println("2");
            MeetingTimes mt = c.getMeetingTimes();
            for (int j=0; j<mt.size();j++) {
                System.out.println("3");
```



```
                MeetingTime m = (MeetingTime) mt.get(j);
                stmt.executeUpdate ("insert into MEETING_TIMES values('" + c.getCrn() + "'," + m.getStartTime().getTime() + "," +
m.getEndTime().getTime() + "," + m.getDay() + ")");
                stmt.close();
                stmt = conn.createStatement();
            }

        }
        System.out.println("4");
        stmt.executeUpdate ("commit");
        System.out.println("5");
        stmt.close();
        conn.close();
        System.out.println("6");
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    /**
    * Checks whether login/password  entered by a user correspond to those
    *of any existing accounts.
    */
    public int login(String uname, String pass) {
        try {
        Class.forName("oracle.jdbc.driver.OracleDriver");
        Connection conn =
DriverManager.getConnection("jdbc:oracle:thin:@DELPHI:1521:ORA1","s04-cs389-s20","8Qi5Ys10");
        Statement stmt = conn.createStatement();
        uname = sqlProtect(uname);
        pass = encrypt(pass);
        String strSQL = "select * from student";
        ResultSet rs = stmt.executeQuery (strSQL);
        while(rs.next()) {
        if (rs.getString("login").equalsIgnoreCase(uname) && rs.getString("Password").equalsIgnoreCase(pass)) {

            if (rs.getInt("ISADMINISTRATOR") > 0) return 2;

            //System.out.println("ok!");
            return 1;
        } else {
            //System.out.println("not good!");

        }

        }
        stmt.close();
        conn.close();
        } catch (Exception e) {
            e.printStackTrace();
        }
        //conn.close();
        return 0;
    }

    /**
    * Creates account entry in the Student table.
    */
    public void createAccount(String uname, String pass, String fname, String lname, String email) {
        try {
                Class.forName("oracle.jdbc.driver.OracleDriver");
                Connection conn =
DriverManager.getConnection("jdbc:oracle:thin:@DELPHI:1521:ORA1","s04-cs389-s20","8Qi5Ys10");
                Statement stmt = conn.createStatement();
                uname = sqlProtect(uname);
                pass = encrypt(pass);
                String strSQL = "insert into student values('" + uname  + "','" + pass + "','" + fname + "','" + lname + "','" + email +
"',0,'NO_PREF','NO_PREF','NO_PREF')";
                stmt.executeUpdate (strSQL);
                stmt.close();
```

```
            conn.close();
            //System.out.println(strSQL);
            } catch (Exception e) {
                  e.printStackTrace();
      }

   }

   /**
   * Deletes time student does not want to have classes on. Used in "Modify Profile".
   */
   public void deleteTimePreferences(String uname) {
   try {

         Class.forName("oracle.jdbc.driver.OracleDriver");
         Connection conn =
DriverManager.getConnection("jdbc:oracle:thin:@DELPHI:1521:ORA1","s04-cs389-s20","8Qi5Ys10");

         Statement deleteStmt = conn.createStatement();

         deleteStmt.executeUpdate ("delete from StudentPreferencesTimeDetails where LOGIN = '" + uname + "'");
         deleteStmt.close();

          conn.close();

      } catch (Exception e) {
          e.printStackTrace();
      }
   }



   /**
   * Saves user preferences.
   */
public void saveProfile(Student s) {
      try {

         Class.forName("oracle.jdbc.driver.OracleDriver");
         Connection conn =
DriverManager.getConnection("jdbc:oracle:thin:@DELPHI:1521:ORA1","s04-cs389-s20","8Qi5Ys10");

         Profile p = s.getProfile();
         String uname = s.getUname();
         MeetingTimes mt = p.getRestrictedTimes();

         Statement stmt = conn.createStatement();
         stmt.executeUpdate ("update student set password = '" + encrypt(s.getPass()) + "',firstname = '" + s.getFname() + "',
lastname = '" + s.getLname() + "', email = '" + s.getEmail() + "', timeofday = '" + p.getTimeOfDay() + "', gapsize = '" +
p.getGaps() + "', classlength = '" + p.getLengthOfClasses() + "' where login = '" + uname + "'");
         stmt.close();

         stmt = conn.createStatement();
         stmt.executeUpdate ("delete from StudentPreferencesTimeDetails where login = '" + uname + "'");
         stmt.close();


         for (int i=0;i<mt.size();i++) {
            MeetingTime m = (MeetingTime) mt.get(i);

            stmt = conn.createStatement();
            stmt.executeUpdate ("insert into StudentPreferencesTimeDetails values('" + uname  + "','" + m.getDay() + "','" +
m.getStartTime().getTime()  + "','" + m.getEndTime().getTime() + "','W')");
            stmt.close();
         }



         conn.close();
         //System.out.println(strSQL);
```



```
         } catch (Exception e) {
                  e.printStackTrace();
      }
}

   /**
   * Saves profile
   */
   public void saveProfile(String uname, String restrictedDay, int startTime, int endTime, BufferedWriter bufWrtrNick) {


      try {
            String strSQL = "insert into StudentPreferencesTimeDetails values('" + uname  + "','" + restrictedDay + "','" +
startTime  + "','" + endTime+ "','W')";
            Class.forName("oracle.jdbc.driver.OracleDriver");
            Connection conn =
DriverManager.getConnection("jdbc:oracle:thin:@DELPHI:1521:ORA1","s04-cs389-s20","8Qi5Ys10");
            Statement stmt = conn.createStatement();
            stmt.executeUpdate (strSQL);
            stmt.close();
            conn.close();
        } catch (Exception e) {
        e.printStackTrace();
      }



   }

   private String sqlProtect(String s) {
      return s.replace('-',' ').replace('\'',' ').replace('"',' ');
   }

   private String encrypt(String s) {
      try {
      MessageDigest md = MessageDigest.getInstance("SHA");
      md.update(s.getBytes());
      return toHexString(md.digest()).substring(1,21);
      } catch (Exception e) {
         e.printStackTrace();
      }
      return null;
   }

   private String toHexString ( byte[] b )
   {
   StringBuffer sb = new StringBuffer( b.length * 2 );
   for ( int i=0 ; i<b.length ; i++ )
      {
   // look up high nibble char
   sb.append( hexChar [ ( b[ i] & 0xf0 ) >>> 4 ] ) ;

   // look up low nibble char
   sb.append( hexChar [ b[ i] & 0x0f ] ) ;
   }
   return sb.toString() ;
   }


// table to convert a nibble to a hex char.
static char[] hexChar =
{
'0' , '1' , '2' , '3' ,
'4' , '5' , '6' , '7' ,
'8' , '9' , 'a' , 'b' ,
'c' , 'd' , 'e' , 'f' }
;
}
```

```
package psb;

public class TimeSlot implements Comparable {
    private MeetingTime meetingTime;
    private String description;
    private String displayColor;
    private String crn;
    private int schedNumber;

    public TimeSlot(MeetingTime meetingTime, String crn, int schedNumber) {
        this.meetingTime = meetingTime;
        this.crn = crn;
        this.schedNumber = schedNumber;
    }

    public MeetingTime getMeetingTime() {
        return meetingTime;
    }

    public void setDescription(String description) {
        this.description = description;
    }

    public void setDisplayColor(String displayColor) {
        this.displayColor = displayColor;
    }

    public int compareTo(Object o) {
        TimeSlot t = (TimeSlot) o;
        return this.getMeetingTime().getStartTime().compareTo(t.getMeetingTime().getStartTime());
    }


    public String getDescription() { return description; }
    public String getDisplayColor() { return displayColor; }

    public String toString() {
        String ret = "";
        ret += "<table cellpadding=0 cellspacing=0 style=\"border: 1px black dotted;\" width=100% height=" +
(meetingTime.getEndTime().getTime() - meetingTime.getStartTime().getTime())/120 + ">";
        ret += "<tr bgcolor='" + this.displayColor + "'>";
        ret += "<td style=\"font-size:12px;font-family:Arial\" valign=top>";
        ret += "<div style=\"cursor: hand\" onClick=\"openpopup('" + crn + "','" + schedNumber + "');\">";
        ret += this.description;
        ret += "</div></td>";
        ret += "</tr>";
        ret += "</table>";
        return ret;
    }
}
```

SemesterXRef.java 05/May/2004

```
package psb.dm;

public class SemesterXRef {

   public static void main(String[] args) {
      System.out.println(toCharBased("20047"));
   }

   private static String[] xref = {"Jan. Int. ", "Spring ", "May Int. ", "Summer I ", "Summer II ", "August Int ", "Fall "};

   public static String toNumeric(String s) {
      String year = s.substring(s.length()-4);
      for (int i=0;i<xref.length;i++) {
         if (s.replaceAll("%20"," ").indexOf(xref[i]) >= 0) return year + (i+1);
      }
      return null;
   }

   public static String toCharBased(String s) {
      String year = s.substring(0,4);
      String s1 = s.substring(s.length()-1);
      //System.out.println(year + "|" + s1);
      return xref[Integer.parseInt(s1)-1] + year;
   }

}
```

```
package psb;

import java.util.StringTokenizer;

public class Time implements Comparable {

   private int seconds;

   public static void main(String[] args) {
      Time t = new Time("04:30", "pm");
      System.out.println(t);
   }

   public int compareTo(Object o) {
      Time t = (Time) o;
      //System.out.println("\t\tComparing " + this.getTime() + " to " + t.getTime());

      if(this.getTime() > t.getTime()) {
         return 1;
      } else if (this.getTime() < t.getTime()) {
         return -1;
      } else { return 0; }
   }

   public Time(int seconds) {
      if (seconds < 0 || seconds > 86400) throw new RuntimeException("Invalid time specified");
      this.seconds = seconds;
   }

   public Time(int hours, int minutes, String a) {
      this.seconds = (hours * 60 + minutes) * 60;
      if (a.equalsIgnoreCase("PM") && hours != 12) this.seconds += 43200;
   }

   public Time(String val, String a) {
      StringTokenizer st = new StringTokenizer(val,":");
      if (st.countTokens() != 2) throw new RuntimeException("Invalid time format.  Try HH:MM");
      int hours = Integer.parseInt(st.nextToken());
      this.seconds = (hours * 60 + Integer.parseInt(st.nextToken())) * 60;
      if (a.equalsIgnoreCase("PM") && hours != 12) this.seconds += 43200;
   }

   public int getHour() {
      int seconds = this.seconds;
      int minutes = seconds / 60;
      int hours = minutes / 60;
      String m = "AM";

      seconds %= 60;
      minutes %= 60;

      if (hours >= 12) {
         hours %= 12;
         m = "PM";
      }

      if (hours == 0) hours = 12;
      return hours;
   }

   public int getMinute() {
      int seconds = this.seconds;
      int minutes = seconds / 60;
      int hours = minutes / 60;
      String m = "AM";

      seconds %= 60;
      minutes %= 60;

      if (hours >= 12) {
```



```
         hours %= 12;
         m = "PM";
      }

      if (hours == 0) hours = 12;
      return minutes;
   }

   public String getAMPM() {
      int seconds = this.seconds;
      int minutes = seconds / 60;
      int hours = minutes / 60;
      String m = "AM";

      seconds %= 60;
      minutes %= 60;

      if (hours >= 12) {
         hours %= 12;
         m = "PM";
      }

      if (hours == 0) hours = 12;

      return m;

   }

   public String toString() {
      int seconds = this.seconds;
      int minutes = seconds / 60;
      int hours = minutes / 60;
      String m = "AM";

      seconds %= 60;
      minutes %= 60;

      if (hours >= 12) {
         hours %= 12;
         m = "PM";
      }

      if (hours == 0) hours = 12;

      return hours + ":" + pad0(minutes)  + " " + m;
   }

   public int getTime() {
      return seconds;
   }

   private String pad0(String s) {
      if (s.length() == 1) return "0"+s;
      return s;
   }

   private String pad0(int s) {
      if (s < 10) return "0"+s;
      return String.valueOf(s);
   }
}
```

```
package psb;

import java.util.StringTokenizer;

public class MeetingTime {
   private char day;
   private Time startTime;
   private Time endTime;

   public static void main(String[] args) {
      System.out.println(new MeetingTime('R',"10:35 - 12:35 pm "));
      /* System.out.println(new MeetingTime('R',"03:35 - 05:35 am "));
      System.out.println(new MeetingTime('R',"12:35 - 12:50 pm "));
      System.out.println(new MeetingTime('R',"03:35 - 12:35 pm "));
      System.out.println(new MeetingTime('R',"03:35 - 02:35 pm "));
      System.out.println(new MeetingTime('R',"01:35 - 02:35 pm ")); */
   }

   public MeetingTime() {}

   public MeetingTime(char day, Time startTime, Time endTime) {
      this.day = day;
      this.startTime = startTime;
      this.endTime = endTime;
   }

   public char getDay() { return day; }
   public Time getStartTime() { return startTime; }
   public Time getEndTime() { return endTime; }

   public boolean isWholeDay() {
      return this.startTime.getTime() == 0 && this.endTime.getTime() == 86400;
   }

   public MeetingTime(char day, String val) {
      // 10:10 - 12:10 pm
      this.day = day;
      StringTokenizer st = new StringTokenizer(val.replaceAll(" "," "),":- ");
      int t1 = Integer.parseInt(st.nextToken());
      int t2 = Integer.parseInt(st.nextToken());
      int t3 = Integer.parseInt(st.nextToken());
      int t4 = Integer.parseInt(st.nextToken());
      String a = st.nextToken();


      if (a.equalsIgnoreCase("am")) {

         startTime = new Time(t1,t2,a);
         endTime = new Time(t3,t4,a);
      } else if (a.equalsIgnoreCase("pm")) {
         if (t1 == 12) {
            startTime = new Time(t1,t2,"pm");
            endTime = new Time(t3,t4,"pm");
         } else if (t1 > t3 || t3 == 12) {
            startTime = new Time(t1,t2,"am");
            endTime = new Time(t3,t4,"pm");
            //System.out.println(endTime.getTime());
         } else {
            startTime = new Time(t1,t2,"pm");
            endTime = new Time(t3,t4,"pm");
         }
      }
   }

   public String toString() {
      return day + " " + startTime + "-" + endTime;
   }

   public int lengthHours() {
      return (endTime.getTime() - startTime.getTime())/3600;
```



```
   }


   public boolean overlaps(MeetingTime m) {
      //System.out.println("Comparing " + this + " to " + m);
      //System.out.println("Comparing " + this.getTime() + " to " + m.getTime());
      boolean before = (this.getEndTime().compareTo(m.getStartTime()) <= 0);
      boolean after = (this.getStartTime().compareTo(m.getEndTime()) >= 0);
      //System.out.println("\t" + before + " " + after);
      return (m.getDay() == this.getDay()) && !(before || after);
   }

   public MeetingTime getOverlapPeriod(MeetingTime m) {
      //System.out.println("Calculating overlap for " + this + " to " + m);
      boolean before = (this.getEndTime().compareTo(m.getStartTime()) <= 0);
      boolean after = (this.getStartTime().compareTo(m.getEndTime()) >= 0);
      if ((m.getDay() == this.getDay()) && !(before || after)) {
         Time startTime1 = m.getStartTime();
         Time endTime1 = m.getEndTime();

         if (this.getStartTime().compareTo(m.getStartTime()) >= 0) {
            startTime1 = this.getStartTime();
            //System.out.println(this.getStartTime() + " >= " + m.getStartTime());
            }
         if (this.getEndTime().compareTo(m.getEndTime()) <= 0) {
            endTime1 = this.getEndTime();
            //System.out.println(this.getEndTime() + " <= " + m.getEndTime());
         }

         return new MeetingTime(this.getDay(),startTime1,endTime1);
      }

      return null;
   }
}
```

```
package psb.dm;

import psb.*;
import java.io.*;
import java.util.*;
import java.net.*;
import java.text.SimpleDateFormat;

/**
 * This class extracts data from Pace University's Schedule web-page.
 *
 * @author     Viktor Geller
 * @version    %I%, %G%
 * @since      JDK1.4
 */
public class DataMiner {

    public final String PACE_CS_HREF =
"http://programs.pace.edu/classschedule/CSresults2.cfm?ClassSchedule__SectListDiv=All&ClassSchedule__SectListCamp=
    public final String PACE_CS_CRN_HREF =
"http://programs.pace.edu/classschedule/CSDetails2.CFM?ClassSchedule__CRN=[CRN]&ClassSchedule__YYT=[SEMESTE
    public final String PACE_CS_MAIN_HREF = "http://programs.pace.edu/classschedule/classschedule2.cfm";

    public final String SEMESTERS_START = "<select NAME=\"Semester__YearTerm\" size=\"1\">";
    public final String SEMESTERS_END = "</select>";

    public final String COURSE_START = "<table border=\"2\" background=\"backgrnd.gif\">";
    public final String COURSE_END = "</TABLE>";

    public final String CRN_START = "Course:</b></td><td colspan=\"3\"><font Color=\"navy\"> ";
    public final String CRN_END = " </td></tr>";

    public static void main (String[] args)
    {
        boolean flag = false;
    DataMiner me = new DataMiner();
    //System.out.println(me.getDataForCourse(args[0] + "%20" + args[1], "Spring%202004"));

    //Course c = me.getDataForCrn("28555", "Fall%202004");
    //System.out.println(c.toString());
    System.out.println(me.getSemesterSelectBox());
    //Courses c2 = me.getDataForCourse("ACC%20203", "Fall%202004");

    /*for (int i=0;i<c1.size();i++) {
        for (int j=0;j<c2.size();j++) {
            Course cr1 = (Course) c1.get(i);
            Course cr2 = (Course) c2.get(j);
            if (cr1.overlaps(cr2)) {
                System.out.println(cr1.getJavaScriptLink() + " conflicts with " + cr2.getCm() + ", on " + cr1.getOverlapPeriods(cr2));
                //if (!flag) System.out.println(cr1+""+cr2);
                flag = true;
            } else {
                System.out.println(cr1.getCm() + " OK with " + cr2.getCrn());
            }

        }
    } */

    }


    private String readTextFile(String f) {
        String ret = "";
        try {
        BufferedReader in = new BufferedReader(new FileReader(f));
        String str;
        while ((str = in.readLine()) != null) {
            ret += str;
        }
        in.close();
```



```
        } catch (IOException e) {
        }
        return ret;

    }

    public String getSemesterSelectBox() {
        try {
            SimpleDateFormat sdt = new SimpleDateFormat("yyyyMMdd");
            String filename = "semester_" + sdt.format(new Date()) + ".txt";
            File f = new File(filename);
            if (f.exists()) {
                System.out.println(f.getAbsolutePath());
                return readTextFile(filename);
            }

            BufferedWriter bufWrtr = new BufferedWriter(new FileWriter(filename));
            String output = "", str;
            URL url = new URL(PACE_CS_MAIN_HREF);
            BufferedReader in = new BufferedReader(new InputStreamReader(url.openStream()));
            while ((str = in.readLine()) != null) { output += str; }
            in.close();
            //System.out.println(output);
            int start = output.indexOf(SEMESTERS_START);
            int end = output.indexOf(SEMESTERS_END,start);
            output = output.substring(start, end + SEMESTERS_END.length()).replaceAll("<option selected>","");
            bufWrtr.write(output);
            bufWrtr.close();
            return output;
        } catch (Exception e) { e.printStackTrace(); }
        return null;
    }

    public psb.Course getDataForCrn(String crn, String semester) {
        try {
            String output = "", str;
            URL url = new
URL(PACE_CS_CRN_HREF.replaceAll("\\[CRN\\]",crn).replaceAll("\\[SEMESTER\\]",SemesterXRef.toNumeric(semester)));
            BufferedReader in = new BufferedReader(new InputStreamReader(url.openStream()));
            while ((str = in.readLine()) != null) { output += str; }
            in.close();
            int start = output.indexOf(CRN_START);
            int end = output.indexOf(CRN_END,start);
            //System.out.println(output);
            if (start == -1) throw new RuntimeException("HTML Parsing via CRN failed!");
            output = output.substring(start + CRN_START.length(), end);

            String course = output.replaceAll(" ","%20");
            //System.out.println(course + "|" + semester);
            Courses c = getDataForCourse(course, semester);

            for (int i=0;i<c.size();i++) {
                Course c1=(Course) c.get(i);
                if (c1.getCm().equals(cm)) return c1;
            }
            } catch (Exception e) { e.printStackTrace(); }
            return null;
    }

    public psb.Courses getDataForCourse(String course, String semester) {
        return getDataForCourse(course, semester, "All");
    }

    public psb.Courses getDataForCourse(String course, String semester, String campus) {
        Courses retVal = new Courses();
        try {

        String output = "", str;
        int i = 0;
```

```
        URL url = new
URL(PACE_CS_HREF.replaceAll("\\[COURSE\\]",course).replaceAll("\\[SEMESTER\\]",semester).replaceAll("\\[CAMPUS\\]",c

        BufferedReader in = new BufferedReader(new InputStreamReader(url.openStream()));
        while ((str = in.readLine()) != null) { output += str; }
        in.close();

        int start = output.indexOf(COURSE_START);
        int end = output.indexOf(COURSE_END,start);

        //System.out.println(output);

        if (start == -1) throw new RuntimeException("HTML Parsing via Course failed!");
        output = output.substring(start, end);

        boolean hasRooms = (output.indexOf("<td>Room</td>") >= 0);
        int skips = 11;
        if (!hasRooms) skips = 10;

        output = output.replaceAll("<td.*?>","|");

        //System.out.println(output);

        StringTokenizer st = new StringTokenizer(output,"|");
        while(st.hasMoreTokens()) {
           if (i>skips) {
              String crn, course2, title, status, days, meetingTimes, room, site, fee, remark, instructor;

              crn = reformat(st.nextToken());
              course2 = reformat(st.nextToken());
              title = reformat(st.nextToken());
              status = reformat(st.nextToken());
              days = reformat(st.nextToken().replaceAll("\\<br\\>","|"));
              meetingTimes = reformat(st.nextToken().replaceAll("\\<br\\>","|"));

              if (hasRooms) room = reformat(st.nextToken().replaceAll("\\<br\\>","|"));
              else room = null;

              site = reformat(st.nextToken());
              fee = reformat(st.nextToken());
              remark = reformat(st.nextToken());
              instructor = reformat(st.nextToken());

                 System.out.println(crn);
              //System.out.println("Data Miner: course="+course+"; courseReplaced="+course.replaceAll("%20"," ")+";
course2="+course2);
              if (course.replaceAll("%20"," ").equalsIgnoreCase(course2)) {
                 retVal.add(new Course(crn, course2, title, status, days, meetingTimes, room, site, fee, remark, instructor));
              }
           } else { st.nextToken(); }
           i++;
        }


     } catch (Exception e) { e.printStackTrace(); }
        return retVal;
     }

     public void parseData(String str, PrintWriter out) {
        String captions[] = {"CRN","Course","Title","Status","Days","Meeting Times","Room","Site","Fee","Remark","Instructor"};
        int i = 0;


     }

     public static String reformat(Object o) { return reformat(o.toString()); }
     public static String reformat(String s) { return s.replaceAll("\\<.*?\\>","").replaceAll(" "," "); }

     }
```

```
package psb;

import java.util.*;

/**
 * This class reperesents a course offered by Pace University.
 *
 * @author     Viktor Geller
 * @version    %I%, %G%
 * @since      JDK1.4
 */

public class Course implements Comparable {


    private String cm,course,title,status,room,site,fee,remark,instructor;
    int rank;
    private MeetingTimes meetingTimes;


    public static void main(String[] args) {
//      Schedule s = ((Schedule) ((Schedules) request.getSession().getAttribute("s")).get(Integer.parseInt(sched_number)));

        Course c = new Course("M <br>W ", " 01:25 - 03:25 pm  <br>\n01:25 - 02:20 pm ");
        Course d = new Course(" TF ", " 09:00 - 11:00 am ");
        System.out.println(c);
    }


    /**
    * This method checks if this course overlaps another course
    */
        public boolean overlaps(Course c) {
        MeetingTimes cm = c.getMeetingTimes();
        return overlaps(cm);
    }

    /**
    * This method checks if this course overlaps another course
    */

    public boolean overlaps(MeetingTimes cm) {
        //MeetingTimes cm = c.getMeetingTimes();
        MeetingTimes tm = this.getMeetingTimes();

        for (int i=0; i<cm.size(); i++) {
            MeetingTime cmm = (MeetingTime) cm.get(i);
            for (int j=0; j<tm.size(); j++) {
                MeetingTime tmm = (MeetingTime) tm.get(j);
                if (cmm.overlaps(tmm)) {
                    //System.out.println("overlap at " + cmm + "|" + tmm);
                    return true;
                }
            }
        }

        return false;
    }

    /**
    * This method returns "LONG" if a course meets one day a week,
    *"SHORT" otherwise. Used to compare generated courses' structure to
    *user's preferences
    */

public String getClassLength() {
    if (meetingTimes.size() == 1) return "LONG";
    else return "SHORT";
}
```



```
    /**
    * This method returns "EARLY","MORNING","AFTERNOON" or "EVENING", Used to
    *compare generated courses' meeting time to user's preferences
    */
public String getTimeOfDay() {
    for (int j=0;j<meetingTimes.size();j++) {
        MeetingTime m = (MeetingTime) meetingTimes.get(j);
        int a = m.getStartTime().getTime();
        if (a > 57600) return "EVENING";
        if (a > 43200) return "AFTERNOON";
        if (a > 32400) return "MORNING";
        if (a > 25200) return "EARLY";
    }
    return "";
}


    /**
    * Constructor.
    */

    public Course(String days, String mT) {
        meetingTimes = processMTimes(days, mT);
    }

    public String getTitle() { return title; }

    /**
    * This method returns the course (CRN) number.
    */
    public String getCm() { return cm; }
    /**
    * This method returns the course (e.g. CS 389)
    */
    public String getCourse() { return course; }
    /**
    * This method returns the course status (closed, canceled, etc)
    */
    public String getStatus() { return status; }
    /**
    * This method returns the room where the course is offered.
    */
    public String getRoom() { return room; }
    /**
    * This method returns the site where the course is offered.
    */
    public String getSite() { return site; }
    /**
    * This method returns the fee for the course
    */
    public String getFee() { return fee; }
    /**
    * This method returns any remarks for the course
    */
    public String getRemark() { return remark; }
    /**
    * This method returns the name of the instructor who teaches the course
    */
    public String getInstructor() { return instructor; }

    /**
    * This method returns the rank for the course
    */
    public int getRank() { return rank; }

    /**
    * This method sets the rank for the course
    */
    public void setRank(int rank) { this.rank = rank; }
```

```
    /**
    * This method returns the times when the class meets
    */
    public MeetingTimes getMeetingTimes() {
        return this.meetingTimes;
    }

    /**
    * Constructor.
    */

    public Course(String crn, String course, String title, String status, String days, String meetingTimes,
            String room, String site, String fee, String remark, String instructor) {
        //System.out.println("new course " + crn);
        this.crn = crn;
        this.course = course;
        this.title = title;
        this.status = status;
        this.meetingTimes = processMTimes(days, meetingTimes);
        this.room = processRoom(room);
        this.site = site;
        this.fee = fee;
        this.remark = remark;
        this.instructor = instructor;

    }


    /**
    * Constructor.
    */      public Course(String crn, String course, String title, String status, MeetingTimes meetingTimes,
            String room, String site, String fee, String remark, String instructor) {
        //System.out.println("new course " + crn);
        this.crn = crn;
        this.course = course;
        this.title = title;
        this.status = status;
        this.meetingTimes = meetingTimes;
        this.room = processRoom(room);
        this.site = site;
        this.fee = fee;
        this.remark = remark;
        this.instructor = instructor;

    }

    /**
     *
    * This method hecks if there is a room specified for a class (sometimes
    * there is not).
    * */
    private String processRoom(String room) {
        if (room == null) return "";
        StringTokenizer dst = new StringTokenizer(room + "|","|");
        return dst.nextToken();
    }

    /**
    * This method creates a list of Meeting times based on 2 strings from the HTML code.
    */
    private MeetingTimes processMTimes(String days, String meetingTimes) {
        MeetingTimes m = new MeetingTimes();
        //days = days.replaceAll("\\<br\\>","|");
        days = days.replaceAll("\\s", "");

            //System.out.println("mtimes: " + meetingTimes);

        meetingTimes = meetingTimes.replace('\n',' ');
        //meetingTimes = meetingTimes.replaceAll("\\<br\\>","|");
```



```
        //System.out.println("mtimes: " + meetingTimes);
        //System.out.println("days: " + days);

        StringTokenizer dst = new StringTokenizer(days,"|");
        StringTokenizer mst = new StringTokenizer(meetingTimes,"|");

        if (dst.countTokens() != mst.countTokens()) throw new RuntimeException("General MTimes parsing failure.  Please
investigate");

        while (dst.hasMoreTokens()) {
            String currday = dst.nextToken();
            String currtime = mst.nextToken();

            for (int i=0; i<currday.length(); i++) {
                //System.out.println("adding " + currday.charAt(i) + " " + currtime);
                m.add(new MeetingTime(currday.charAt(i),currtime));
            }

        }

        return m;
    }



    /**
    * This method returns the time period of where an overlap occurs.
    */
    public MeetingTimes getOverlapPeriods(Course c) {
        MeetingTimes overlaps = new MeetingTimes();
        MeetingTimes cm = c.getMeetingTimes();
        MeetingTimes tm = this.getMeetingTimes();

        for (int i=0; i<cm.size(); i++) {
            MeetingTime cmm = (MeetingTime) cm.get(i);
            for (int j=0; j<tm.size(); j++) {
                MeetingTime tmm = (MeetingTime) tm.get(j);
                if (cmm.overlaps(tmm)) {
                    //System.out.println("overlap at " + cmm + "|" + tmm);
                    //return true;
                    overlaps.add(cmm.getOverlapPeriod(tmm));
                }
            }
        }
        return overlaps;
    }

/*  public String toString() {
        String ret = "";
        for (int i=0; i<meetingTimes.size(); i++) {
            ret += meetingTimes.get(i).toString() + "\n";
        }
        return ret;
    } */

    /**
    * This method returns the URL to Pace Course Info website.
    */
    public String HTMLLinkToPaceCourseInfo() {
        return "<a href=\"http://programs.pace.edu/classschedule/CSDetails2.CFM?ClassSchedule__CRN=" + getCrn() +
"&ClassSchedule__YYT=20042\">" + getCrn() + "</a>";
    }

    /**
    * This method returns the JavaScript link for onClick display of any
    * course on a generated schedule.
    */
    public String getJavaScriptLink() {
        char  s = 92;
```

```
        return "<a href=\"#\" onClick=\"alert('" + this.toString().replaceAll("\n","\\\\n") + "');\">" + getCrn() + "</a>";
    }

    /*
    public String toString() {
        String n = "\n";
        String ret = "";
        ret+= ("CRN: " + crn + n);
        ret+= ("course: " + course + n);
        ret+= ("title: " + title + n);
        ret+= ("status: " + status + n);
        ret+= ("meetingTimes: " + n + meetingTimes);
        ret+= ("room: " + room + n);
        ret+= ("site: " + site + n);
        ret+= ("fee: " + fee + n);
        ret+= ("remark: " + remark + n);
        ret+= ("instructor: " + instructor + n);
        ret += n+n;
        return ret;
    } */

    public String toString() {
        String ret = "";
        ret += "<tr>";
        ret += "<td>" + crn + "</td>";
        ret += "<td>" + course + "</td>";
        ret += "<td>" + title + "</td>";
        ret += "<td><input type=Submit value=\"Delete\" name=\"delete_" + crn + "\"></td>";
        ret += "<td><input type=Submit value=\"Regenerate\" name=\"regen_" + crn + "\"></td>";
        ret += "</tr>";
        return ret;
    }
    public String debugToString() {
        //this is here for debugging purposes - Nikita
        String n = "\n";
        String ret = "";
        ret+= ("CRN: " + crn + n);
        ret+= ("course: " + course + n);
        ret+= ("title: " + title + n);
        ret+= ("status: " + status + n);
        ret+= ("meetingTimes: " + n + meetingTimes);
        ret+= ("room: " + room + n);
        ret+= ("site: " + site + n);
        ret+= ("fee: " + fee + n);
        ret+= ("remark: " + remark + n);
        ret+= ("instructor: " + instructor + n);
        ret += n+n;
        return ret;
    }

    /**
    * This method is used to sort courses based on their rank. Here, rank is
    * based on how many sections are offered for the course. The lower the
    * number of sections, the higher the rank and the earlier course will be
    * put on the schedule, compared to other courses.
    */
    public int compareTo(Object o) {
        Course c= (Course) o;
        if (this.getRank() < c.getRank()) return 1;
        if (this.getRank() == c.getRank()) return 0;
        return -1;
        //return (this.getRank())==(c.getRank());
        //TimeSlot t = (TimeSlot) o;
        //return this.getMeetingTime().getStartTime().compareTo(t.getMeetingTime().getStartTime());
    }

}
```

```
//import Database;
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;
import psb.*;
import psb.dm.*;

/**
 * This servlet handles users' requests, directs flow and controls the main
 * functionality of the PSB system.
 *
 * @author     Viktor Geller
 * @version    %I%, %G%
 * @since      JDK1.4
 */

public class PSBCentralServlet extends HttpServlet {

    public void doPost(HttpServletRequest req, HttpServletResponse resp) {
        try {
        if (req.getRequestURL().indexOf("index.psb") > 0) {
            gotoIndex(req, resp);
        } else if (req.getRequestURL().indexOf("mainscreen.psb") > 0) {
            gotoMainScreen(req, resp);
        } else if (req.getRequestURL().indexOf("new_account.psb") > 0) {
                    gotoPage(req, resp,"/new_account.jsp");
        } else if (req.getRequestURL().indexOf("save_schedule.psb") > 0) {
                saveSchedule(req, resp);
        } else if (req.getRequestURL().indexOf("my_schedules.psb") > 0) {
            gotoMySchedules(req, resp);
        } else if (req.getRequestURL().indexOf("schedule.psb") > 0) {
                    gotoSchedule(req, resp);
        } else if (req.getRequestURL().indexOf("course_info.psb") > 0) {
            gotoPage(req, resp, "/course_info.jsp");
        } else if (req.getRequestURL().indexOf("profile_change.psb") > 0) {
            gotoProfileChange(req, resp);
        } else if (req.getRequestURL().indexOf("createaccount.psb") > 0) {
            createNewAccount(req, resp);
        } else if (req.getRequestURL().indexOf("schedule_action.psb") > 0) {
            mySchedulePageAction(req, resp);
        } else if (req.getRequestURL().indexOf("profile_action.psb") > 0) {
            myProfilePageAction(req, resp);
        } else if (req.getRequestURL().indexOf("admin_action.psb") > 0) {
            adminAction(req, resp);
        } else if (req.getRequestURL().indexOf("schedule_edit_action.psb") > 0) {
            scheduleEditPageAction(req, resp);
        } else if (req.getRequestURL().indexOf("regenerate.psb") > 0) {
            regenerate(req, resp);
        } else if (req.getRequestURL().indexOf("admin.psb") > 0) {
            gotoAdmin(req, resp);
        } else if (req.getRequestURL().indexOf("logout.psb") > 0) {
            req.getSession().setAttribute("s1",null);
            req.getSession().setAttribute("s",null);
            req.getSession().setAttribute("username",null);
            gotoPage(req, resp, "/logout.jsp");
        } else {
            resp.getWriter().println("ERROR: Invalid Command!");
        }
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    /**
    * Handles a "Regenerate schedule" event. Calls overloaded schedule
    * generator method with a schedule and a list of classes to be added to it.
    */
    public void regenerate(HttpServletRequest req, HttpServletResponse resp) throws Exception {
            ClassList cl = new ClassList();
```



```
            for (int i=1;i<=4;i++) {
                String class1 = req.getParameter("class" + i);
                if (class1 != null && !class1.equals("")) cl.add(class1);
            }

            Database db = new Database();
            String uname = (String) req.getSession().getAttribute("username");
            Profile p = db.getStudent(uname).getProfile();


            String semester = req.getParameter("Semester__YearTerm").replaceAll(" ", "%20");
            String campus = req.getParameter("campus");
            int ns = 1;

              Schedules ss = ((Schedules) req.getSession().getAttribute("s1"));

          int s_number = Integer.parseInt((String) req.getSession().getAttribute("scheduleid"));
          for (int i=0; i<ss.size(); i++) {
            Schedule s = (Schedule) ss.get(i);

            if (s.getSchedNumber() == s_number) {


            ScheduleGenerator sg  = new ScheduleGenerator();
            Schedule s2 = (Schedule) ((Schedules) sg.generateSchedulesMain(cl,ns, semester, campus,p,s)).get(0);
             s2.setSchedNumber(s_number);
             db.deleteSchedule(String.valueOf(s_number));
             req.getSession().setAttribute("scheduleid",db.getNextSchedId());
             db.saveSchedule(s2,uname);
             req.getSession().setAttribute("s1",db.getSavedScheduleList(uname));

             break;
             }
          }

             gotoPage(req,resp,"/edit_schedule.jsp");

    }

    /**
    * Directs a user to "profile change" screen.
    */
    public void gotoProfileChange(HttpServletRequest req, HttpServletResponse resp) throws Exception {
        Database db = new Database();
        String uname = (String) req.getSession().getAttribute("username");
        Student s = db.getStudent(uname);
        System.out.println(s.outputJSFiller());
        req.getSession().setAttribute("student",s);
        gotoPage(req, resp, "/profile_change.jsp");
    }

    /**
    * Handles deletion of a user by an admin.
    */
    public void adminAction(HttpServletRequest req, HttpServletResponse resp) throws Exception {
        Database db = new Database();
        Students s = db.getStudentList();
        for (int i=0;i<s.size();i++) {
            if (req.getParameter("delete_student_" + ((Student) s.get(i)).getUname()) != null) db.deleteStudent(((Student)
s.get(i)).getUname());
        }
        gotoAdmin(req, resp);
    }

    /**
    * Goes to "Administrative Actions" page.
    */
    public void gotoAdmin(HttpServletRequest req, HttpServletResponse resp) throws Exception {
        Database db = new Database();
        Students s = db.getStudentList();
```

```
        req.getSession().setAttribute("studentlist",s);
        gotoPage(req, resp, "/admin.jsp");
    }

    /**
    * Handles "Edit Schedule" and "Edit schedule" actions.
    */
    public void scheduleEditPageAction(HttpServletRequest req, HttpServletResponse resp) throws Exception {
        Database db = new Database();
        String uname = (String) req.getSession().getAttribute("username");
        String scheduleid = (String) req.getSession().getAttribute("scheduleid");
        Schedules s1 = (Schedules) req.getSession().getAttribute("s1");
        Schedule curr= null;
        for (int i=0;i<s1.size();i++) {
            Schedule s = (Schedule) s1.get(i);
            if (s.getSchedNumber() == Integer.parseInt(scheduleid)) {
                curr = s;
                break;
            }
        }

        Courses cc = (Courses) curr.getCourseList();
        for (int i=0;i<cc.size();i++) {
            Course c = (Course) cc.get(i);
            String crn = c.getCrn();
            if (req.getParameter("delete_" + crn) != null) {
                db.deleteCrnFromSchedule(scheduleid,crn);
                req.getSession().setAttribute("scheduleid",scheduleid);
                req.getSession().setAttribute("s1",db.getSavedScheduleList(uname));
                gotoPage(req, resp,"/edit_schedule.jsp");
                return;
            }
            if (req.getParameter("regen_" + crn) != null) {
                //regen ..... (scheduleid,crn);
            }
        }
    }

    /**
    * Handles "Delete Schedule" and "Edit schedule" actions.
    */
    public void mySchedulePageAction(HttpServletRequest req, HttpServletResponse resp) throws Exception {
        Database db = new Database();
        int max = Integer.parseInt(db.getNextSchedId()) + 5;

        for (int i=0; i<=max;i++) {
            if (req.getParameter("delete_" + i) != null) {
                db.deleteSchedule(String.valueOf(i));
                gotoMySchedules(req, resp);
                return;
            }
            if (req.getParameter("edit_" + i) != null) {
                req.getSession().setAttribute("scheduleid",String.valueOf(i));
                gotoPage(req, resp,"/edit_schedule.jsp");
                return;
            }
        }
    }

    /**
    * Handles the request to "Save Profile".
    */
    public void myProfilePageAction(HttpServletRequest req, HttpServletResponse resp) throws Exception {
        try {
        String uname = (String) req.getSession().getAttribute("username");
        String pass  = req.getParameter("pass");
        String fname = req.getParameter("fname");
        String lname = req.getParameter("lname");
        String email = req.getParameter("email");
        String timeofday = req.getParameter("timeOfDay");
```



```
        String gaps = req.getParameter("gaps");
        String classlength = req.getParameter("classlength");
        Database db = new Database();
        Profile p = new Profile(timeofday,gaps,classlength);

        db.deleteTimePreferences(uname);
        BufferedWriter bufWrtrNick = new BufferedWriter(new FileWriter("NikitaDebug.txt"));
        bufWrtrNick.write("Starting!!\n");
        System.out.println("NIKDEBUG");

        for (int i=0;i<8;i++) {
            String restrictedDay = req.getParameter("restrictedDay" + i);
            /* if (restrictedDay.equalsIgnoreCase("Mon")) {
                restrictedDay="M";
            }
            if (restrictedDay.equalsIgnoreCase("Tue")) {
                restrictedDay="T";
            }
            if (restrictedDay.equalsIgnoreCase("Wed")) {
                restrictedDay="W";
            }
            if (restrictedDay.equalsIgnoreCase("Thu")) {
                restrictedDay="R";
            }
            if (restrictedDay.equalsIgnoreCase("Fri")) {
                restrictedDay="F";
            }
            if (restrictedDay.equalsIgnoreCase("Sat")) {
                restrictedDay="S";
            }
            if (restrictedDay.equalsIgnoreCase("Sun")) {
                restrictedDay="U";
            } */

            bufWrtrNick.write("restrictedDay" + i+"="+restrictedDay+"; ");

            int startHour,startMinute,endHour,endMinute;
            String restrictedStartAMPM,restrictedEndAMPM;
            //System.out.println("STARTH="+req.getParameter("startHour" + i)+"");

            //bufWrtrNick.write("STARTH="+req.getParameter("startHour" + i)+"");
            //bufWrtrNick.write("length="+(req.getParameter("startHour" + i).length())+"");

            if ((req.getParameter("startHour" + i) != null)&& ((req.getParameter("startHour" + i)).length() != 0)
            && (req.getParameter("endHour" + i) != null)&& ((req.getParameter("endHour" + i)).length() != 0)) {
                bufWrtrNick.write("if ");

                startHour = Integer.parseInt(req.getParameter("startHour" + i));
                bufWrtrNick.write("startHour" + i+"="+startHour+"; ");

                if ((req.getParameter("startMinute" + i) != null)&& ((req.getParameter("startMinute" + i)).length() != 0)) {
                    startMinute = Integer.parseInt(req.getParameter("startMinute" + i));
                }
                else {
                    startMinute=0;
                }
                bufWrtrNick.write("startMinute" + i+"="+startMinute+"; ");

                restrictedStartAMPM = req.getParameter("restrictedStartAMPM" + i);
                bufWrtrNick.write("restrictedStartAMPM" + i+"="+restrictedStartAMPM+"; ");

                endHour = Integer.parseInt(req.getParameter("endHour" + i));
                bufWrtrNick.write("endHour" + i+"="+endHour+"; ");

                if ((req.getParameter("endMinute" + i) != null)&& ((req.getParameter("endMinute" + i)).length() != 0)) {
                    endMinute = Integer.parseInt(req.getParameter("endMinute" + i));
                }
                else {
                    endMinute=0;
                }
```

```
            bufWrtrNick.write("endMinute" + i+"="+endMinute+"; ");

            restrictedEndAMPM = req.getParameter("restrictedEndAMPM" + i);
            bufWrtrNick.write("restrictedEndAMPM" + i+"="+restrictedEndAMPM+"; ");


            Time startTime = new Time(startHour,startMinute,restrictedStartAMPM);
            bufWrtrNick.write(startTime.getTime()+"\n");
            Time endTime = new Time(endHour,endMinute,restrictedEndAMPM);
            bufWrtrNick.write(endTime.getTime()+"\n");


            //db.saveProfile(uname, restrictedDay, startTime.getTime(), endTime.getTime(), bufWrtrNick);
            p.addRestrictedTime(new MeetingTime(restrictedDay.toCharArray()[0], startTime, endTime));
          }
          else {
            bufWrtrNick.write("else ");
          }
          bufWrtrNick.write("\n");
        }
        String checkbox, Day;
        bufWrtrNick.write("before j\n");
        char[] days = {'M','T','W','R','F','S','U'};
        for (int j=0;j<days.length;j++) {

          if (req.getParameter(("D_" + days[j])) != null ) {


            //db.saveProfile(uname, Day, 0, 86400, bufWrtrNick);
            p.addRestrictedTime(new MeetingTime(days[j], new Time(0), new Time(86400)));
          }
        }

        bufWrtrNick.close();



        Student s = new Student(uname,pass,fname,lname,email);
         s.setProfile(p);
         db.saveProfile(s);

        } catch (Exception e) { e.printStackTrace(); }
         gotoProfileChange(req, resp);
      }
    /**
    * Directs user to a screen which displays all the saved schedules
    * for this particular user.
    */
       public void gotoMySchedules(HttpServletRequest req, HttpServletResponse resp) throws ServletException,
IOException {
          String uname = (String) req.getSession().getAttribute("username");
          Database db = new Database();
          req.getSession().setAttribute("s1",db.getSavedScheduleList(uname));
          gotoPage(req, resp,"/my_schedules.jsp");

       }

    /**
    * This method recieves the list of classes the user enters, launches
    * schedule generator and directs a user to the generated schedule display
    * screen.
    */
       public void gotoSchedule(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOException {

          ClassList cl = new ClassList();
          for (int i=1;i<=8;i++) {
             String class1 = req.getParameter("class" + i);
             if (class1 != null && !class1.equals("")) cl.add(class1);
          }
```



```
            Database db = new Database();
          String uname = (String) req.getSession().getAttribute("username");
          Profile p = db.getStudent(uname).getProfile();

          System.out.println("aaaaaaaa  " + (p == null));

          String semester = req.getParameter("Semester__YearTerm").replaceAll(" ", "%20");
          String campus = req.getParameter("campus");
          int ns = Integer.parseInt(req.getParameter("ns"));

          ScheduleGenerator sg  = new ScheduleGenerator();
          Schedules s = (Schedules) sg.generateSchedulesMain(cl,ns, semester, campus,p);

          req.getSession().setAttribute("s1",s);

          gotoPage(req,resp,"/schedule.jsp");
       }


     /**
     * Handles "Save Schedule" request.
     */
     public void saveSchedule(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOException {
        String uname = (String) req.getSession().getAttribute("username");
        Database db = new Database();
        String schedNumber = "0";

        for (int i=0;i<100;i++) {
          if (req.getParameter("s" + i) != null) {
            schedNumber = String.valueOf(i);
            Schedule s = (Schedule) (((Schedules) req.getSession().getAttribute("s1")).get(Integer.parseInt(schedNumber)));
            db.saveSchedule(s, uname);
          }
        }

        gotoMySchedules(req, resp);


     }

     /**
     * Redirects user to the url that is provided as a parameter.
     */
     public void gotoPage(HttpServletRequest req, HttpServletResponse resp, String url) throws ServletException, IOException
{
          RequestDispatcher dispatcher = getServletContext().getRequestDispatcher(url);
          dispatcher.forward(req, resp);

     }

     /**
     * Redirects user to the index (login) jsp.
     */
     public void gotoIndex(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOException {
        gotoPage(req, resp, "/index.jsp");

     }

     /**
     * Redirects user to the main screen of the application.
     */
      public void gotoMainScreen(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOException {
           String uname = req.getParameter("uname");
        String pass = req.getParameter("pass");
        Database db = new Database();
        //System.out.println("))))))" + db.login(uname,pass));
        String unameS = (String) req.getSession().getAttribute("username");
        int l = db.login(uname,pass);
        System.out.println(l + "////////");
        if (l > 0) {
```

```
            req.getSession().setAttribute("username",uname);
            if (i == 2) req.getSession().setAttribute("admin",new String("true"));
            gotoPage(req,resp,"/mainscreen.jsp");
        } else if (unameS != null) {
            gotoPage(req,resp,"/mainscreen.jsp");
        } else {
            gotoPage(req,resp,"/index_error.jsp");
        }
        }


    public void doGet(HttpServletRequest req, HttpServletResponse resp) {
        doPost(req,resp);
    }

    /**
    * Check to see if the user credentials specified are valid.
    */
    public boolean authenticateUser(String uname, String pass) {
        return true;
    }
    /**
    * Create a new user account
    */
    public void createNewAccount(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOException {
        String uname = req.getParameter("uname");
        String pass  = req.getParameter("pass");
        String fname = req.getParameter("fname");
        String lname = req.getParameter("lname");
        String email = req.getParameter("email");

        Database db = new Database();
        db.createAccount(uname, pass, fname, lname, email);
        gotoMainScreen(req, resp);
    }
    /**
    * Modify a user account
    */
    public void modifyAccount(String uname, String pass, String fname, String lname, String email){
    }
    /**
    * Delete a user account
    */
    public void deleteAccount(String uname){}
    /**
    * Modify a user's profile
    */
    public void modifyProfile(HttpServletRequest req){}
    /**
    * Create a new profile for a user
    */
    public void createProfile(HttpServletRequest req){}

}
```

```
package psb;

import psb.dm.*;
import psb.*;
import java.io.*;
import java.util.*;



/**
 * This class works together with the DataMiner to generate a list of
 * schedules based on the specified list of classes and the profile the user
 * has entered.
 *
 * @author     Nikita Lukish
 * @version    %I%, %G%
 * @since      JDK1.4
 */
public class ScheduleGenerator {



    public static void main(String[] args) {
        try {

        ClassList cl = new ClassList();
        cl.add("COM%20200");
        cl.add("MAT%20131");
        cl.add("UNV%20101");
        cl.add("CHE%20101");
        cl.add("CS%20389");
        ScheduleGenerator sg  = new ScheduleGenerator();
        BufferedWriter bufWrtr = new BufferedWriter(new FileWriter("c:\\psb_test.html"));
        Profile p = new Profile("NO_PREF", "NO_PREF", "LONG");
        Schedules s = sg.generateSchedulesMain(cl,6,"Spring%202004","All",p);

        for (int i=0;i<s.size();i++) {
            bufWrtr.write("\nSchedule #"+i);
            bufWrtr.write(s.get(i).toString() + "<BR><BR><BR><BR>");
        }
        bufWrtr.close();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    /**
     * This method removes duplicate schedules from a Schedules object
     */
    private Schedules purgeDuplicates(Schedules allSchedules) {
    for (int i=0; i<allSchedules.size();i++) {
        Schedule is = (Schedule) allSchedules.get(i);
        for (int j=0; j<allSchedules.size();j++) {
            Schedule js = (Schedule) allSchedules.get(j);
            if (i != j && is.equals(js)) allSchedules.removeElementAt(j);
        }
    }
    return allSchedules;
}
    /**
     * This overloaded method returnes generated schedules
     */
    public Schedules generateSchedulesMain(ClassList classList, int numberOfSchedules, String semester, String campus,
Profile p) {
        Schedule sch= new Schedule();
        return generateSchedulesMain(classList, numberOfSchedules, semester, campus, p, sch);
    }

    /**
     * This overloaded method returnes generated schedules
```



```
     */
    public Schedules generateSchedulesMain(ClassList classList, int numberOfSchedules, String semester, String campus,
Profile p, Schedule previousSched) {
        Schedules s = generateSchedules( classList,  numberOfSchedules * 30,  semester,  campus,  p, previousSched);

        for (int i=0;i<s.size();i++) {
            ((Schedule) s.get(i)).setProfile(p);
        }
        Collections.sort(s);
        Schedules ret = new Schedules();
        if (s.size() < numberOfSchedules) numberOfSchedules = s.size();
        for (int i=0;i<numberOfSchedules;i++) {
            ret.add(s.get(i));
        }
        //System.out.println("Size: " + s.size());
        return ret;
    }

    /**
     * This method contains the main logic for generating schedules
     */
    public Schedules generateSchedules(ClassList classList, int numberOfSchedules, String semester, String campus, Profile
p, Schedule previousSched) {

        // 1) Obtain a list of classes with class info for each course
        Courses allCourses=getListOfCourses(classList, semester, campus);
        // 2) Purge the list of classes deemed invalid due to profile constraints
        allCourses=purgeTheList(duplicateCourses(allCourses), p);
        // 3) Assign a numeric rank to each class
        //allCourses=assignRank(allCourses);
        allCourses=assignRank(duplicateCourses(allCourses));
        // 4) Sort the list based on the class rank
        allCourses=sortTheList(duplicateCourses(allCourses));
        // 5) Loop through the sorted list and generate NONRANDOM schedules

        Schedules allSchedules = new Schedules();
        int numOfSched=0;
        try {
        BufferedWriter bufWrtr = new BufferedWriter(new FileWriter("c:\\nikita.txt"));
        bufWrtr.write("START...\n");


        //Schedule sch= new Schedule();
        previousSched.setProfile(p);
        numOfSched=numberOfSchedules(previousSched, allCourses, allSchedules, numberOfSchedules, semester);
        bufWrtr.write("numOfSched="+numOfSched+"\n");
        bufWrtr.write("numberOfSchedules="+numberOfSchedules+"\n");
        } catch (Exception e) {
            e.printStackTrace();
        }
        // 5) If total number of all possible schedules is less than the number
        // of schedules the user requested, return all possible schedules.
        // If it is not, randomize the results by running a random generateSchedules function
        if (numOfSched<numberOfSchedules) {

            allSchedules=purgeDuplicates(allSchedules);
            return allSchedules;
        }
        else {
            Schedules d=generateSchedules(classList, allCourses, numberOfSchedules, semester,p);
            System.out.println("numOfSched="+numOfSched);
            System.out.println("numberOfSchedules="+numberOfSchedules);
            d=purgeDuplicates(d);
            return d;
        }
}

/**
* This method uses Data Miner to get a list of all Sections for courses
*entered by the user
```

```
   */
   public Courses getListOfCourses(ClassList c, String semester, String campus) {
      Courses recievedCourses = new Courses();

      for (int i=0; i<c.size();i++) {
         String courseName=(String) c.get(i);
         DataMiner dm = new DataMiner();
         //Courses c1 = dm.getDataForCourse(courseName.replaceAll(" ","%20"), "Spring%202004");
         Courses c1 = dm.getDataForCourse(courseName.replaceAll(" ","%20"), semester, campus);

         for (int j=0; j<c1.size();j++) {
            recievedCourses.add((Course) c1.get(j));
         }

      }
      return recievedCourses;
   }

   /**
   * This method assigns priority to generated sections. Courses which
   * have fewer sections and that are therefore, rare, will be put on the
   * schedule first
   */
   public Courses assignRank(Courses c) {
      try {
         BufferedWriter bufWrtrRank = new BufferedWriter(new FileWriter("c:\\nikitaRANK.txt"));
         bufWrtrRank.write("START...\n");
      for (int i=0; i<c.size();i++) {
         Course thisCourse= (Course) c.get(i);
         Courses allSectionsForThisClass=findAllSectionsForThisClass(thisCourse, c);
         //bufWrtrRank.write("thisCourse="+thisCourse.debugToString()+"; \n");
         //bufWrtrRank.write("frequency="+allSectionsForThisClass.size()+ " out of "+c.size()+"; \n");
         //bufWrtrRank.write("rank="+(100-(allSectionsForThisClass.size()))+"; \n");
         thisCourse.setRank(100-(allSectionsForThisClass.size()));//will insert code for assigning rank here
         c.setElementAt(thisCourse, i);
      }
      bufWrtrRank.close();
      } catch (Exception e) {
         e.printStackTrace();
      }
      return c;
   }

   /**
   * This method returns a Schedules object, which is a list of
   * generated schedules
   */
   public Schedules generateSchedules(ClassList c, Courses allTheCourses, int numberOfSchedules, String semester,
Profile p) {
      Schedules listOfSchedules = new Schedules();
      try {
         BufferedWriter bufWrtr = new BufferedWriter(new FileWriter("c:\\nikita.txt"));
         bufWrtr.write("START...\n");
         for (int scheduleCounter=0;scheduleCounter<numberOfSchedules; scheduleCounter++) {
            //Schedule thisSchedule = new Schedule();
            //Schedule thisSchedule = generateFirstSchedule(c, allCourses);
            bufWrtr.write("Schedule="+scheduleCounter+"\n");
            Schedule thisSchedule = generateOneSchedule(c, duplicateCourses(allTheCourses), bufWrtr);
            thisSchedule.setSchedNumber(scheduleCounter);
            thisSchedule.setSemester(semester);
            thisSchedule.setProfile(p);
            listOfSchedules.insertElementAt(thisSchedule, scheduleCounter);

         }
      bufWrtr.close();
      } catch (Exception e) {
         e.printStackTrace();
      }
      return listOfSchedules;
   }
```



```
   /**
   * This method removes duplicated courses from a list of courses
   */
   public Courses duplicateCourses(Courses c) {
      Courses ret = new Courses();
      for (int i=0;i<c.size();i++) {
         ret.add(c.get(i));
      }
      return ret;
   }


   /**
   * This method generates one schedule by randomly selecting a section for
   * the first course entered, then the second course entered, etc.
   */
   public Schedule generateOneSchedule(ClassList c, Courses allCourses, BufferedWriter bufWrtr) {
      Schedule thisSched= new Schedule();
      try {

      for (int k=0; k<c.size();k++) {

         bufWrtr.write("Inside For...\n");
         String thisClass= (String) c.get(k);

         Course firstCourse=(Course) allCourses.get(0);
         //1. In the list of generated courses,
         //find all sections for the first course
         Courses allSectionsForThisClass=findAllSectionsForThisClass(firstCourse, allCourses);
         //2.generate a random course among them, a course that does not overlap
         //meeting time of any courses that were already put on thisSched
         Course courseToBeInserted = randomCourseForThisClass(allSectionsForThisClass);
         Courses coursesAlreadyScheduled=thisSched.getCourseList();
         boolean Flag = true;
         while (Flag == true) {
            courseToBeInserted=randomCourseForThisClass(allSectionsForThisClass);
            Flag = false;
            for(int m=k; m>(0); m--) {
            //this for loops will set Flag to true if courseToBeInserted overlaps any
            //course already scheduled
               Course courseToBeChecked=(Course) coursesAlreadyScheduled.get((m-1));
               if (courseToBeChecked.overlaps(courseToBeInserted)) {
                  Flag=true;
               }
            }
         }

         //3.remove all sections for this class
         for (int j=0; j<allCourses.size();j++) {
            Course courseToInspect=(Course) allCourses.get(j);
            String str1=(String) courseToInspect.getCourse();
            String str2=(String) firstCourse.getCourse();
            if (str1.equalsIgnoreCase(str2)) {
               allCourses.removeElementAt(j);
               j--;
            }
         }

         //4. If the section time does not overlap with
         //meeting times of any of the previously scheduled classes,
         //insert the course into the schedule
         bufWrtr.write("COURSE NUMBER="+k+"\n");
         bufWrtr.write(courseToBeInserted.debugToString()+"\n");
         thisSched.addCourse(courseToBeInserted);
      }

      } catch (Exception e) {
         e.printStackTrace();
      }
```

```
        return thisSched;
    }

    /**
    * This method finds all sections for the particular course and removes
    * them, once a section for this course has been added to the schedule.
    */
    Courses findAllSectionsForThisClass(Course firstCourse, Courses allCourses) {
        Courses allSections= new Courses();
        for (int i=0; i<allCourses.size();i++) {
            Course thisCourse=(Course) allCourses.get(i);
            String s1=(String) firstCourse.getCourse();
            String s2=(String) thisCourse.getCourse();
            if (s1.equalsIgnoreCase(s2)) {
                allSections.add((Course) allCourses.get(i));
            }
        }
        return allSections;


    }

    /**
    * This method returns a random course for a list of generated sections
    */
    Course randomCourseForThisClass(Courses allSectionsForThisClass) {
        int randomCourseIndex= (int) (Math.random() * allSectionsForThisClass.size());
        Course generatedCourse=(Course) allSectionsForThisClass.get(randomCourseIndex);
        return generatedCourse;
    }

    /**
    * This recursive method returns a number of schedules. It goes through the
    * possible permutations in a systematic non-random way. The number of
    * schedules it attempts to generate is provided by parameter
    * numberOfSchedules. If the method cannot generate the requested number of
    * schedules, it returns the number of possible schedules.
    */
    int numberOfSchedules(Schedule thisSched, Courses allCourses, Schedules allSchedules, int numberOfSchedules,
String semester ) {
        Course firstCourse=(Course) allCourses.get(0);
        Courses allSectionsForThisClass=findAllSectionsForThisClass(firstCourse, allCourses);
        Courses coursesAlreadyScheduled=thisSched.getCourseList();
        if (allSectionsForThisClass.size()<allCourses.size()) {
            int i=0;
            while ( (i<allSectionsForThisClass.size()) && ((allSchedules.size())<numberOfSchedules)) {
                Course thisCourse=(Course) allSectionsForThisClass.get(i);
                Schedule copyOfThisSched= duplicateSchedule(thisSched);
                Courses copyOfAllCourses=duplicateCourses(allCourses);

                boolean Test = false;
                for(int m=0; m<coursesAlreadyScheduled.size(); m++) {
                    Course courseToBeChecked=(Course) coursesAlreadyScheduled.get((m));
                    if (courseToBeChecked.overlaps(thisCourse)) {
                        Test=true;
                    }

                }
                if (Test == false) {
                    int j=0;
                    while  ((j<copyOfAllCourses.size()) && ((allSchedules.size())<numberOfSchedules)) {
                        Course courseToInspect=(Course) copyOfAllCourses.get(j);

                        String str1=(String) courseToInspect.getCourse();
                        String str2=(String) thisCourse.getCourse();
                        if (str1.equalsIgnoreCase(str2)) {
                            copyOfAllCourses.removeElementAt(j);
                            j--;
                        }
                        j++;
```



```
                    }
                    copyOfThisSched.addCourse(thisCourse);
                    int numOfScheduleBelow=numberOfSchedules(copyOfThisSched, copyOfAllCourses, allSchedules,
numberOfSchedules, semester);
                }
                i++;
            }
            return allSchedules.size();

        }
        else {
            int count=0;

            Schedule copySched= duplicateSchedule(thisSched);
            int k=0;
            while (( k<allCourses.size()) && ((allSchedules.size())<numberOfSchedules)) {

                Course crs=(Course) allCourses.get(k);
                boolean Flag = false;
                for(int m=0; m<coursesAlreadyScheduled.size(); m++) {
                //this for loops will set Flag to true if courseToBeInserted overlaps any
                //course already scheduled
                    Course courseToBeChecked=(Course) coursesAlreadyScheduled.get((m));
                    if (courseToBeChecked.overlaps(crs)) {
                        Flag=true;
                    }

                }
                if (Flag == false) {
                    count=count+1;
                    copySched.addCourse(crs);
                    System.out.println("\n\nSCHEDULE #="+allSchedules.size());
                    copySched.setSchedNumber(allSchedules.size());
                    copySched.setSemester(semester);
                    allSchedules.add(copySched);
                    copySched= duplicateSchedule(thisSched);
                }
            }
            return count;

        }


    }

    /**
    * This method creates a copy of a schedule passed to it as a parameter.
    */
    public Schedule duplicateSchedule(Schedule s) {
        Schedule ret = new Schedule();
        Courses c=s.getCourseList();
        for (int i=0;i<c.size();i++) {
            ret.addCourse((Course) c.get(i));
        }
        return ret;
    }


    /**
    * This method removes from the list of courses the courses that do not meet
    * user's criteria.
    */
    Courses purgeTheList(Courses allCourses, Profile p) {

        for (int j=0; j<allCourses.size();j++) {
            Course courseToInspect=(Course) allCourses.get(j);
            MeetingTimes rt=p.getRestrictedTimes();
            if (courseToInspect.overlaps(rt)) {
                allCourses.removeElementAt(j);
                j--;
```

```
            }
        }
        return allCourses;

    }

    /**
    * This method sorts a list of courses provided to it. This is needed
    * so that the courses are inserted into schedules in a certain order -
    * rare courses first.
    */
    Courses sortTheList(Courses  copyOfAllCourses) {
        try {
            Courses rc=new Courses();

            BufferedWriter bufWrtrRank = new BufferedWriter(new FileWriter("c:\\nikitaSORT.txt"));
            Course firstCourse= (Course) copyOfAllCourses.get(0);
            rc.addCourse(firstCourse);
        for (int i=0; i<copyOfAllCourses.size();i++) {
            Course thisCourse= (Course) copyOfAllCourses.get(i);
        }

        Collections.sort(copyOfAllCourses);

        for (int i=0; i<copyOfAllCourses.size();i++) {
            Course thisCourse= (Course) copyOfAllCourses.get(i);
        }


        bufWrtrRank.close();
        } catch (Exception e) {
            e.printStackTrace();
        }

        return copyOfAllCourses;
    }

}
```