```java
/**
 * SEF Assignment 2 - Student Management System
 *
 * @author Stuart Parker (s3390317)
 * @author Jason Hamilton (s3455196)
 * @author Aidan Cyr (s3471910)
 * @author Jake Seeary (s3430163)
 */

import java.util.*;
import java.io.*;
import java.text.*;
import java.time.*;
import java.time.format.*;

public class App
{
    HashMap<String, Course> courses = new HashMap<String, Course>();
    HashMap<String, Venue> venues = new HashMap<String, Venue>();
    HashMap<String, Lecturer> lecturers = new HashMap<String, Lecturer>();
    HashMap<String, Tutor> tutors = new HashMap<String, Tutor>();
    HashMap<String, Applicant> applicants = new HashMap<String, Applicant>();
    HashMap<String, Student> students = new HashMap<String, Student>();
    Scanner scan = new Scanner(System.in);
    private int year = 2015;
    private int semester = 1;
    DateTimeFormatter timeFormat = DateTimeFormatter.ofPattern("h:mm a");
    DateTimeFormatter dateFormat = DateTimeFormatter.ofPattern("d/M/yyyy");
    LocalDate censusDate = LocalDate.of(2015, 3, 31); // Default census date
    String adminPassword = "admin"; // Default admin password

    public static void main(String[] args)
    {
        App a = new App();

        a.readDataFromFile();

        String mainMenuOpts[] = {"Student Menu", "Admin Menu", "View Menu",
                "Config Menu"};
        Menu m = new Menu("Main Menu", mainMenuOpts, Menu.Type.FULL);
        int n;
        int resp;
        do
        {
            if((resp = m.getResponse()) == 0)
            {
                break;
            }
            switch(resp)
            {
            case 1: // Student Menu
                Student student = a.studentLogin();
                if(student == null)
                {
                    break;
                }
                String studentMenuOpts[] = {"Enrol in course",
```

```
58                          "Withdraw from course", "Register in tutorial",
59                          "Deregister from tutorial", "Change password"};
60                  Menu studentMenu = new Menu("Student Submenu", studentMenuOpts,
61                          Menu.Type.FULL);
62                  while((n = studentMenu.getResponse()) != 0)
63                  {
64                      a.studentMenu(n, student);
65                  }
66                  break;
67              case 2: // Admin Menu
68                  if(a.adminLogin())
69                  {
70                      String adminMenuOpts[] = {"Add offering", "Add lecture",
71                              "Assign lecturer", "Add tutorial", "Appoint tutor",
72                              "Assign tutor", "Admit student",
73                              "Change admin password"};
74                      Menu adminMenu = new Menu("Admin Submenu", adminMenuOpts,
75                              Menu.Type.FULL);
76                      while((n = adminMenu.getResponse()) != 0)
77                      {
78                          a.adminMenu(n);
79                      }
80                  }
81                  break;
82              case 3: // View Menu
83                  String viewMenuOpts[] = {"View courses", "View venues",
84                          "View lecturers", "View tutors",
85                          "View venue timetable", "View lecturer timetable",
86                          "View tutor timetable", "View student timetable"};
87                  Menu viewMenu = new Menu("View Submenu", viewMenuOpts,
88                          Menu.Type.FULL);
89                  while((n = viewMenu.getResponse()) != 0)
90                  {
91                      a.viewMenu(n);
92                  }
93                  break;
94              case 4: // Config Menu
95                  String configMenuOpts[] = {"Change Census Date"};
96                  Menu configMenu = new Menu("Config Submenu", configMenuOpts,
97                          Menu.Type.FULL);
98                  while((n = configMenu.getResponse()) != 0)
99                  {
100                     a.configMenu(n);
101                 }
102                 break;
103             }
104         } while(true);
105
106         a.writeDataToFile();
107     }
108
109     public void studentMenu(int n, Student student)
110     {
111         switch(n)
112         {
113         case 1: // Enrol in course
114             handleEnrol(student);
```

```
115             break;
116         case 2: // Withdraw from course
117             handleWithdrawFromCourse(student);
118             break;
119         case 3: // Register in tutorial
120             handleRegisterTutorial(student);
121             break;
122         case 4: // Deregister from tutorial
123             handleDeregisterTutorial(student);
124             break;
125         case 5: // Change password
126             changeStudentPassword(student);
127             break;
128         }
129     }
130 
131     public void adminMenu(int n)
132     {
133         switch(n)
134         {
135         case 1: // Add offering
136             handleCreateOffering();
137             break;
138         case 2: // Add lecture
139             handleAddLecture();
140             break;
141         case 3: // Assign lecturer
142             handleAssignLecturer();
143             break;
144         case 4: // Add tutorial
145             handleAddTutorial();
146             break;
147         case 5: // Appoint tutor
148             handleAppointApplicantAsTutor();
149             break;
150         case 6: // Assign tutor
151             handleAssignTutor();
152             break;
153         case 7: // Admit student
154             handleAdmitStudent();
155             break;
156         case 8: // Change admin password
157             changeAdminPassword();
158             break;
159         }
160     }
161 
162     public void viewMenu(int n)
163     {
164         switch(n)
165         {
166         case 1: // View courses
167             displayMap(courses);
168             break;
169         case 2: // View venues
170             displayMap(venues);
171             break;
```

```
172             case 3: // View lecturers
173                 displayMap(lecturers);
174                 break;
175             case 4: // View tutors
176                 displayMap(tutors);
177                 break;
178             case 5: // View venue timetable
179                 handlePrintVenueTimetable();
180                 break;
181             case 6: // View lecturer timetable
182                 handlePrintLecturerTimetable();
183                 break;
184             case 7: // View tutor timetable
185                 handlePrintTutorTimetable();
186                 break;
187             case 8: // View student timetable
188                 handlePrintStudentTimetable();
189                 break;
190             }
191         }
192 
193         public void configMenu(int n)
194         {
195             switch(n)
196             {
197             case 1: // Change census date
198                 handleChangeCensusDate();
199                 break;
200             }
201         }
202 
203         public App()
204         {
205             initialiseCourses();
206             initialiseVenues();
207             initialiseLecturers();
208             initialiseTutors();
209             initialiseApplicants();
210             initialiseStudents();
211         }
212 
213         public void initialiseCourses()
214         {
215             Course course1 = new Course("P101", "Programming 1",
216                     "Teach Basic Programming");
217             Course course2 = new Course("P102", "Programming 2",
218                     "Teach Intermediate Programming");
219             Course course3 = new Course("S101", "Software Engineering",
220                     "Teach UML and Modelling");
221             Course course4 = new Course("WP1", "Web Programming",
222                     "Teach Web Technologies");
223             Course course5 = new Course("UI1", "User Interface",
224                     "Teach UI Principles");
225             Course course6 = new Course("MATH", "Discrete Maths",
226                     "Teach Maths needed for CS");
227             Course course7 = new Course("NET1", "Networking",
228                     "Teach networking principles");
```

```
229             Course course8 = new Course("CSYS", "Computer Systems",
230                     "Introduction to computer systems");
231 
232             course3.addPrereq(course1);
233             course2.addPrereq(course1);
234             course7.addPrereq(course2);
235             course7.addPrereq(course6);
236 
237             courses.put(course1.getID(), course1);
238             courses.put(course2.getID(), course2);
239             courses.put(course3.getID(), course3);
240             courses.put(course4.getID(), course4);
241             courses.put(course5.getID(), course5);
242             courses.put(course6.getID(), course6);
243             courses.put(course7.getID(), course7);
244             courses.put(course8.getID(), course8);
245 
246         }
247 
248         public void initialiseVenues()
249         {
250             venues.put("12.10.02",
251                     new Venue("12.10.02", 120, Venue.Purpose.LECTURE));
252             venues.put("12.10.03",
253                     new Venue("12.10.03", 200, Venue.Purpose.LECTURE));
254             venues.put("10.10.22",
255                     new Venue("10.10.22", 36, Venue.Purpose.TUTELAB));
256             venues.put("10.10.23",
257                     new Venue("10.10.23", 36, Venue.Purpose.TUTELAB));
258         }
259 
260         public void initialiseLecturers()
261         {
262             lecturers.put("e44556", new Lecturer("e44556", "Tim O'Connor",
263                     "Lecturer", "14.13.12"));
264             lecturers.put("e44321", new Lecturer("e44321", "Richard Cooper",
265                     "Professor", "14.13.12"));
266             lecturers.put("e54321", new Lecturer("e54321", "Jane Smith",
267                     "Lecturer", "11.9.10"));
268         }
269 
270         public void initialiseTutors()
271         {
272             tutors.put("e78965", new Tutor("e78965", "John Smith", "Tutor",
273                     "123 Fake Street, Fakeville", "01 2345 6789"));
274             tutors.put("e12345", new Tutor("e12345", "Jane Doe", "Tutor",
275                     "1 Fake Drive, Fakerston", "09 8765 4321"));
276         }
277 
278         public void initialiseApplicants()
279         {
280             applicants.put("e45612", new Applicant("e45612", "Harry Potter",
281                     "Applicant"));
282             applicants.put("e78945", new Applicant("e78945", "Hermione Granger",
283                     "Applicant"));
284         }
285 
```

```
286     public void initialiseStudents()
287     {
288         try
289         {
290             admitStudent("s1234567", "Test McTest", "test",
291                     "03 1234 5678", "123 Fake St, Fakeville VIC, 3000",
292                     LocalDate.of(1994, 1, 1), new ArrayList<Course>());
293         }
294         catch(DuplicateStudentException e)
295         {
296             // Do nothing
297         }
298     }
299
300     private void hold()
301     {
302         System.out.print("Press enter to continue...");
303         scan.nextLine();
304     }
305
306     public void handleChangeCensusDate()
307     {
308         System.out.println("Current census date is: "
309                 + dateFormat.format(censusDate));
310
311         LocalDate newCensusDate = null;
312         while(newCensusDate == null)
313         {
314             try
315             {
316                 System.out.print("Enter new census date (d/M/yyyy): ");
317                 String newCensusDateStr = scan.nextLine();
318                 newCensusDate = LocalDate.parse(newCensusDateStr, dateFormat);
319             }
320             catch(DateTimeParseException e)
321             {
322                 System.out.println("Invalid date format");
323             }
324         }
325
326         changeCensusDate(newCensusDate);
327
328         System.out.println("Success! New census date is: "
329                 + dateFormat.format(censusDate));
330     }
331
332     public void changeCensusDate(LocalDate date)
333     {
334         censusDate = date;
335     }
336
337     /**
338      * a) Administrators can add new course offerings. Only one per course in
339      * any one semester
340      */
341     public void handleCreateOffering()
342     {
```

```
343             System.out.print("Enter Course ID: ");
344             String courseID = scan.nextLine().toUpperCase();
345             Course course = getCourse(courseID);
346 
347             if(course == null)
348             {
349                 System.out.println("Course ID is invalid");
350                 hold();
351             }
352             else
353             {
354                 System.out.print("Enter Expected Number: ");
355                 int expectedNum = scan.nextInt();
356                 scan.nextLine();
357                 try
358                 {
359                     createOffering(course, expectedNum, year, semester);
360                 }
361                 catch(PreExistException e)
362                 {
363                     System.out.println(e.getMessage());
364                     hold();
365                 }
366             }
367         }
368 
369         public CourseOffering createOffering(Course course, int expectedNum,
370                 int year, int semester) throws PreExistException
371         {
372             return course.createOffering(expectedNum, year, semester);
373         }
374 
375         /**
376          * b) Administrator can add lectures specifying day, time and venue
377          */
378         public void handleAddLecture()
379         {
380             System.out.print("Enter course ID: ");
381             String courseID = scan.nextLine().toUpperCase();
382             CourseOffering offering = getCourseOffering(courseID, year, semester);
383             if(offering == null)
384             {
385                 System.out.println("No course offering for this course");
386                 hold();
387                 return;
388             }
389 
390             System.out.print("Enter venue location: ");
391             String location = scan.nextLine();
392             Venue venue = getVenue(location);
393             if(venue == null)
394             {
395                 System.out.println("No such venue");
396                 hold();
397                 return;
398             }
399 
```

```
400             DayOfWeek day = null;
401             while(day == null)
402             {
403                 try
404                 {
405                     System.out.print("Enter day of lecture (e.g. Monday): ");
406                     day = DayOfWeek.valueOf(scan.nextLine().toUpperCase());
407                 }
408                 catch(IllegalArgumentException e)
409                 {
410                     System.out.println("Invalid day");
411                 }
412             }
413 
414             LocalTime startTime = null;
415             while(startTime == null)
416             {
417                 try
418                 {
419                     System.out.print("Enter start time (h:mm AM/PM): ");
420                     startTime = LocalTime.parse(scan.nextLine(), timeFormat);
421 
422                 }
423                 catch(DateTimeParseException e)
424                 {
425                     System.out.println("Invalid time format");
426                 }
427             }
428 
429             System.out.print("Enter duration (in minutes): ");
430             int duration = scan.nextInt();
431             scan.nextLine();
432 
433             try
434             {
435                 addLecture(offering, day, startTime, duration, venue);
436                 System.out.println("Lecture successfully added");
437             }
438             catch(ClashException | PreExistException | UnsuitableVenueException
439                     | LessonTimeOutOfBoundsException e)
440             {
441                 System.out.println(e.getMessage());
442                 hold();
443             }
444         }
445 
446         public void addLecture(CourseOffering co, DayOfWeek day,
447                 LocalTime startTime, int duration, Venue venue)
448                 throws ClashException, PreExistException, UnsuitableVenueException,
449                 LessonTimeOutOfBoundsException
450         {
451             co.addLecture(day, startTime, duration, venue);
452         }
453 
454         /**
455          * c) Administrator can assign lecturers to the lectures
456          */
```

```
457         public void handleAssignLecturer()
458         {
459             System.out.print("Enter Course ID: ");
460             String courseID = scan.nextLine().toUpperCase();
461             CourseOffering courseOffering = getCourseOffering(courseID, year,
462                     semester);
463             if(courseOffering == null)
464             {
465                 System.out.println("No course offering yet");
466                 hold();
467                 return;
468             }
469 
470             Lecture lecture = courseOffering.getLecture();
471             if(lecture == null)
472             {
473                 System.out.println("No lecture assigned to this course offering "
474                         + "yet");
475                 hold();
476                 return;
477             }
478 
479             System.out.print("Enter Lecturer ID: ");
480             String lecID = scan.nextLine();
481             Lecturer lecturer = getLecturer(lecID);
482             if(lecturer == null)
483             {
484                 System.out.println("No lecturer with such ID");
485                 hold();
486                 return;
487             }
488 
489             try
490             {
491                 assignLecturer(lecture, lecturer);
492                 System.out.println("Lecturer successfully assigned");
493             }
494             catch(ClashException | PreExistException e)
495             {
496                 System.out.println(e.getMessage());
497                 hold();
498             }
499         }
500 
501         public void assignLecturer(Lecture lecture, Lecturer lecturer)
502                 throws ClashException, PreExistException
503         {
504             lecturer.assign(lecture);
505         }
506 
507         /**
508          * d) Administrator can add tutorials specifying date, time and venue.
509          * Tutorials for each course offering should be labelled T1, T2, ... (?)
510          */
511         public void handleAddTutorial()
512         {
513             System.out.print("Enter Course ID: ");
```

```
514             String courseID = scan.nextLine().toUpperCase();
515
516             if(!courses.containsKey(courseID))
517             {
518                 System.out.println("Course does not exist");
519                 hold();
520                 return;
521             }
522
523             CourseOffering offering = getCourseOffering(courseID, year, semester);
524             if(offering == null)
525             {
526                 System.out.println("No course offering for this course yet");
527                 hold();
528                 return;
529             }
530
531             System.out.print("Enter venue location: ");
532             String location = scan.nextLine();
533             Venue venue = getVenue(location);
534             if(venue == null)
535             {
536                 System.out.println("No such venue");
537                 hold();
538                 return;
539             }
540
541             DayOfWeek day = null;
542             while(day == null)
543             {
544                 try
545                 {
546                     System.out.print("Enter day of tutorial (e.g. Monday): ");
547                     day = DayOfWeek.valueOf(scan.nextLine().toUpperCase());
548                 }
549                 catch(IllegalArgumentException e)
550                 {
551                     System.out.println("Invalid day format");
552                 }
553             }
554
555             LocalTime startTime = null;
556             while(startTime == null)
557             {
558                 try
559                 {
560                     System.out.print("Enter start time (h:mm AM/PM): ");
561                     startTime = LocalTime.parse(scan.nextLine(), timeFormat);
562
563                 }
564                 catch(DateTimeParseException e)
565                 {
566                     System.out.println("Invalid time format");
567                 }
568             }
569
570             System.out.print("Enter duration (in minutes): ");
```

```
571             int duration = scan.nextInt();
572 
573             scan.nextLine();
574 
575             try
576             {
577                 addTutorial(offering, day, startTime, duration, venue);
578                 System.out.println("Tutorial successfully added");
579             }
580             catch(ClashException | UnsuitableVenueException |
581                     LessonTimeOutOfBoundsException e)
582             {
583                 System.out.println(e.getMessage());
584                 hold();
585             }
586         }
587 
588         public Tutorial addTutorial(CourseOffering offering, DayOfWeek day,
589                 LocalTime startTime, int duration, Venue venue)
590                 throws ClashException, UnsuitableVenueException,
591                 LessonTimeOutOfBoundsException
592         {
593             Tutorial tutorial = offering.addTutorial(day, startTime, duration,
594                     venue);
595             return tutorial;
596         }
597 
598         /**
599          * e) Administrator can appoint suitable applicants as tutors (Note:
600          * shortlisting/deleting not handled in this iteration)
601          */
602         public void handleAppointApplicantAsTutor()
603         {
604             System.out.print("Enter employee ID of tutor applicant: ");
605             String eNo = scan.nextLine();
606             Applicant applicant = getApplicant(eNo);
607 
608             if(applicant == null)
609             {
610                 System.out.println("No tutor applicant with that ID");
611                 hold();
612                 return;
613             }
614 
615             System.out.print("Enter phone number: ");
616             String phone = scan.nextLine();
617 
618             System.out.print("Enter address: ");
619             String address = scan.nextLine();
620 
621             try
622             {
623                 appointApplicantAsTutor(applicant, phone, address);
624             }
625             catch(DuplicateTutorException e)
626             {
627                 System.out.println(e.getMessage());
```

```
628             hold();
629         }
630     }
631
632     public void appointApplicantAsTutor(Applicant applicant, String phone,
633             String address) throws DuplicateTutorException
634     {
635
636         // Check if a tutor with this ID already exists.
637         if(tutors.get(applicant.getENo()) == null)
638         {
639             // Convert applicant to tutor and add to tutors HashMap
640             tutors.put(applicant.getENo(), new Tutor(applicant.getENo(),
641                     applicant.getName(), "Tutor", phone, address));
642
643             // Remove applicant from applicants HashMap
644             applicants.remove(applicant.getENo());
645         }
646         else
647         {
648             throw new DuplicateTutorException("A tutor with that ID already "
649                     + "exists.");
650         }
651     }
652
653     /**
654      * f) Administrator can assign tutor to specific tutorials
655      */
656     public void handleAssignTutor()
657     {
658         // Get course offering
659         System.out.print("Enter Course ID: ");
660         String courseID = scan.nextLine().toUpperCase();
661         CourseOffering courseOffering = getCourseOffering(courseID, year,
662                 semester);
663         if(courseOffering == null)
664         {
665             System.out.println("No course offering yet");
666             hold();
667             return;
668         }
669
670         // Get tutor
671         System.out.print("Enter tutor ID: ");
672         String tutorID = scan.nextLine();
673         Tutor tutor = getTutor(tutorID);
674         if(tutor == null)
675         {
676             System.out.println("No tutor with ID " + tutorID + " exists");
677             hold();
678             return;
679         }
680
681         // Get list of tutorials
682         ArrayList<Tutorial> tutorials = courseOffering.getTutorials();
683         if(tutorials.size() == 0)
684         {
```

```
685             System.out.println("No tutorials assigned to this course offering "
686                     + "yet");
687             hold();
688             return;
689         }
690 
691         // Create menu options "<day> <startTime>-<endTime> (<location>)"
692         String tutorialStrings[] = new String[tutorials.size()];
693         for(int i = 0; i < tutorials.size(); i++)
694         {
695             Tutorial thisTutorial = tutorials.get(i);
696 
697             tutorialStrings[i] = thisTutorial.getDay().getDisplayName(
698                     TextStyle.FULL, Locale.getDefault()) + " ";
699             tutorialStrings[i] += timeFormat.format(thisTutorial.getStart());
700             tutorialStrings[i] += "-" + timeFormat.format(
701                     thisTutorial.getEnd());
702             tutorialStrings[i] += " (" + thisTutorial.getVenue().getLocation()
703                     + ")";
704         }
705 
706         // Get tutorial
707         Menu tutorialMenu = new Menu("Select tutorial:", tutorialStrings,
708                 Menu.Type.SELECTION);
709         int n;
710         if((n = tutorialMenu.getResponse()) == 0)
711         {
712             hold();
713             return;
714         }
715         Tutorial chosenTutorial = tutorials.get(n - 1);
716 
717         // Assign tutor to chosen tutorial
718         try
719         {
720             assignTutor(chosenTutorial, tutor);
721             System.out.println("Tutor successfully assigned");
722         }
723         catch(ClashException | PreExistException e)
724         {
725             System.out.println(e.getMessage());
726             hold();
727         }
728     }
729 
730     public void assignTutor(Tutorial tutorial, Tutor tutor)
731             throws ClashException, PreExistException
732     {
733         tutor.assign(tutorial);
734     }
735 
736     /**
737      * g) Administrator can admit students and grant exemptions for those with
738      * advanced standing
739      */
740     public void handleAdmitStudent()
741     {
```

```
742             System.out.println("Enter new students details:");
743             System.out.print("Student no.: ");
744             String sNum = scan.nextLine();
745 
746             System.out.print("Name: ");
747             String name = scan.nextLine();
748 
749             System.out.print("Phone number: ");
750             String phone = scan.nextLine();
751 
752             System.out.print("Address: ");
753             String address = scan.nextLine();
754 
755             LocalDate dob = null;
756             while(dob == null)
757             {
758                 try
759                 {
760                     System.out.print("Date of birth (d/M/yyyy): ");
761                     dob = LocalDate.parse(scan.nextLine(), dateFormat);
762                 }
763                 catch(DateTimeParseException e)
764                 {
765                     System.out.println("Invalid date format");
766                 }
767             }
768 
769             System.out.print("Exempt courses (IDs comma-delimited): ");
770             String exemptCoursesStr = scan.nextLine();
771             String[] exemptCoursesArr = exemptCoursesStr.split(",");
772 
773             // Create ArrayList of exempt courses
774             ArrayList<Course> exemptCourses = new ArrayList<Course>();
775             // Check that at least one courseID has been entered (else exemptCourses
776             // should remain empty)
777             if(!(exemptCoursesArr.length == 1 && exemptCoursesArr[0].equals("")))
778             {
779                 for(String courseID : exemptCoursesArr)
780                 {
781                     Course course = courses.get(courseID);
782                     if(course == null)
783                     {
784                         System.out.println("Error: No course with ID '"
785                                 + courseID + "'");
786                         hold();
787                         return;
788                     }
789                     exemptCourses.add(course);
790                 }
791             }
792 
793             // Create default password ("p<year><month><day>", e.g. "p19940726"
794             DecimalFormat df = new DecimalFormat("00");
795             String password = "p" + String.valueOf(dob.getYear())
796                     + df.format(dob.getMonthValue())
797                     + df.format(dob.getDayOfMonth());
798 
```

```
799             try
800             {
801                 admitStudent(sNum, name, password, phone, address, dob,
802                         exemptCourses);
803                 System.out.println("Student successfully admitted");
804             }
805             catch(DuplicateStudentException e)
806             {
807                 System.out.println(e.getMessage());
808                 hold();
809             }
810         }
811 
812         public void admitStudent(String sNum, String name, String password,
813                 String phone, String address, LocalDate dob,
814                 ArrayList<Course> exemptCourses) throws DuplicateStudentException
815         {
816             if(students.containsKey(sNum))
817             {
818                 throw new DuplicateStudentException("A student with sNum '" + sNum
819                         + "' already exists");
820             }
821 
822             // A student is uniquely identified by full name and phone number,
823             // check that this student has not already been admitted
824             for(Student student : students.values())
825             {
826                 if(name.equals(student.getName())
827                         && phone.equals(student.getPhone()))
828                 {
829                     throw new DuplicateStudentException("Student already exists "
830                             + "in system");
831                 }
832             }
833 
834             Student newStudent = new Student(sNum, name, password, phone, address,
835                     dob, exemptCourses);
836 
837             students.put(sNum, newStudent);
838 
839         }
840 
841         /**
842          * h) Students can enrol into courses for which they meet the necessary
843          * prerequisites
844          */
845         public void handleEnrol(Student student)
846         {
847             System.out.print("Enter Course ID: ");
848             String courseID = scan.nextLine().toUpperCase();
849 
850             if(!courses.containsKey(courseID))
851             {
852                 System.out.println("Course does not exist");
853                 hold();
854                 return;
855             }
```

```
856
857             CourseOffering offering = getCourseOffering(courseID, year, semester);
858             if(offering == null)
859             {
860                 System.out.println("No course offering for this course yet");
861                 hold();
862                 return;
863             }
864
865             try
866             {
867                 enrol(student, offering);
868                 System.out.println("Enrolment successful");
869             }
870             catch(OverEnrolmentException | CensusDateExceededException
871                     | AlreadyEnrolledException | IncompletePrerequisitesException e)
872             {
873                 System.out.println(e.getMessage());
874                 hold();
875             }
876         }
877
878         public CourseEnrolment enrol(Student student, CourseOffering offering)
879                 throws OverEnrolmentException, CensusDateExceededException,
880                 AlreadyEnrolledException, IncompletePrerequisitesException
881         {
882             if(censusDate.isBefore(LocalDate.now()))
883             {
884                 throw new CensusDateExceededException("The census date for "
885                         + "this course has already passed, you can no longer "
886                         + "enrol.");
887             }
888
889             CourseEnrolment newEnrolment = student.enrol(offering);
890
891             return newEnrolment;
892         }
893
894         /**
895          * i) Students can withdraw from a course until census date (after
896          * deregistering from any tute)
897          */
898         public void handleWithdrawFromCourse(Student student)
899         {
900             ArrayList<CourseEnrolment> currentEnrolments = student
901                     .getCurrentEnrolments();
902             String enrolmentNames[] = student.getCurrentEnrolmentNames();
903
904             Menu enrolmentMenu = new Menu("Choose course:", enrolmentNames,
905                     Menu.Type.SELECTION);
906
907             int n;
908             if((n = enrolmentMenu.getResponse()) == 0)
909             {
910                 hold();
911                 return;
912             }
```

```
913             CourseEnrolment chosenCourse = currentEnrolments.get(n - 1);
914
915             try
916             {
917                 withdrawFromCourse(student, chosenCourse);
918                 System.out.println("Successfully withdrawn from course");
919
920             }
921             catch(TutorialEnrolledException | CensusDateExceededException e)
922             {
923                 System.out.println(e.getMessage());
924                 hold();
925             }
926         }
927
928         public void withdrawFromCourse(Student student, CourseEnrolment chosenCourse)
929                 throws TutorialEnrolledException, CensusDateExceededException
930         {
931             if(chosenCourse.getTutorial() != null)
932             {
933                 // Still enrolled in tutorial
934                 throw new TutorialEnrolledException("Still enrolled in tutorial, "
935                         + "you cannot withdraw from course.");
936             }
937
938             if(censusDate.isBefore(LocalDate.now()))
939             {
940                 throw new CensusDateExceededException("The census date for "
941                         + "this course has already passed, you can no longer "
942                         + "withdraw.");
943             }
944
945             student.getCurrentEnrolments().remove(chosenCourse);
946         }
947
948         /**
949          * j) Students can register (as long as tute is not full) and deregister
950          * from tutorials
951          */
952         public void handleRegisterTutorial(Student student)
953         {
954             ArrayList<CourseEnrolment> currentEnrolments = student
955                     .getCurrentEnrolments();
956             String enrolmentNames[] = student.getCurrentEnrolmentNames();
957
958             Menu enrolmentMenu = new Menu("Select course:",
959                     enrolmentNames, Menu.Type.SELECTION);
960
961             int n;
962             if((n = enrolmentMenu.getResponse()) == 0)
963             {
964                 hold();
965                 return;
966             }
967             CourseEnrolment chosenCourse = currentEnrolments.get(n - 1);
968
969             // Create menu options "<day> <startTime>-<endTime> (<location>)"
```

```
970             ArrayList<Tutorial> tutorials = chosenCourse.getCourseOffering()
971                     .getTutorials();
972             String tutorialStrings[] = new String[tutorials.size()];
973             for(int i = 0; i < tutorials.size(); i++)
974             {
975                 Tutorial thisTutorial = tutorials.get(i);
976 
977                 tutorialStrings[i] = thisTutorial.getDay().getDisplayName(
978                         TextStyle.FULL, Locale.getDefault()) + " ";
979                 tutorialStrings[i] += timeFormat.format(thisTutorial.getStart());
980                 tutorialStrings[i] += "-" + timeFormat.format(
981                         thisTutorial.getEnd());
982                 tutorialStrings[i] += " (" + thisTutorial.getVenue().getLocation()
983                         + ")";
984             }
985 
986             Menu tutorialMenu = new Menu("Select tutorial:",
987                     tutorialStrings, Menu.Type.SELECTION);
988             Tutorial chosenTutorial = null;
989 
990             int m;
991             if((m = tutorialMenu.getResponse()) == 0)
992             {
993                 hold();
994                 return;
995             }
996             chosenTutorial = tutorials.get(m - 1);
997 
998             try
999             {
1000                 registerTutorial(student, chosenCourse, chosenTutorial);
1001                 System.out.println("Successfully registered in tutorial");
1002             }
1003             catch(OverCapacityException | TutorialAlreadyEnrolledException e)
1004             {
1005                 System.out.println(e.getMessage());
1006                 hold();
1007             }
1008         }
1009 
1010         public void registerTutorial(Student student, CourseEnrolment enrolment,
1011                 Tutorial tutorial) throws OverCapacityException,
1012                 TutorialAlreadyEnrolledException
1013         {
1014             student.registerTutorial(enrolment, tutorial);
1015         }
1016 
1017         public void handleDeregisterTutorial(Student student)
1018         {
1019             ArrayList<CourseEnrolment> currentEnrolments = student
1020                     .getCurrentEnrolments();
1021             String enrolmentNames[] = student.getCurrentEnrolmentNames();
1022 
1023             Menu enrolmentMenu = new Menu("Select course:",
1024                     enrolmentNames, Menu.Type.SELECTION);
1025 
1026             int n;
```

```
1027            if((n = enrolmentMenu.getResponse()) == 0)
1028            {
1029                hold();
1030                return;
1031            }
1032            CourseEnrolment chosenCourse = currentEnrolments.get(n - 1);
1033
1034            try
1035            {
1036                deregisterTutorial(student, chosenCourse);
1037                System.out.println("Successfully deregistered from tutorial");
1038            }
1039            catch(TutorialNotEnrolledException e)
1040            {
1041                System.out.println(e.getMessage());
1042            }
1043        }
1044
1045        public void deregisterTutorial(Student student, CourseEnrolment enrolment)
1046                throws TutorialNotEnrolledException
1047        {
1048            student.deregisterTutorial(enrolment);
1049        }
1050
1051        /**
1052         * k) Users can view timetables for any venue, lecturer, tutor or
1053         * student
1054         */
1055        public void handlePrintVenueTimetable()
1056        {
1057            System.out.print("Enter  Venue Location: ");
1058            String location = scan.nextLine();
1059            Venue venue = getVenue(location);
1060            if(venue == null)
1061            {
1062                System.out.println("No Venue at this location");
1063                hold();
1064                return;
1065            }
1066
1067            printVenueTimetable(venue);
1068            hold();
1069        }
1070
1071        public void printVenueTimetable(Venue venue)
1072        {
1073            if(venue.getLessons().size() == 0)
1074            {
1075                System.out.println("No lessons assigned to this venue");
1076            }
1077
1078            venue.printTimetable();
1079        }
1080
1081        public void handlePrintLecturerTimetable()
1082        {
1083            System.out.print("Enter Lecturer ID: ");
```

```
1084            Lecturer lecturer = getLecturer(scan.nextLine());
1085            if(lecturer == null)
1086            {
1087                System.out.println("No lecturer with this ID");
1088                hold();
1089                return;
1090            }
1091
1092            printLecturerTimeTable(lecturer);
1093            hold();
1094        }
1095
1096        public void printLecturerTimeTable(Lecturer lecturer)
1097        {
1098            ArrayList<Lecture> lectures = getLectures(lecturer);
1099            if(lectures == null || lectures.size() == 0)
1100            {
1101                System.out.println("No lectures assigned");
1102            }
1103
1104            lecturer.printTimetable(lectures);
1105        }
1106
1107        public void handlePrintTutorTimetable()
1108        {
1109            System.out.print("Enter Tutor ID: ");
1110            Tutor tutor = getTutor(scan.nextLine());
1111            if(tutor == null)
1112            {
1113                System.out.println("No tutor with this ID");
1114                hold();
1115                return;
1116            }
1117
1118            printTutorTimetable(tutor);
1119            hold();
1120        }
1121
1122        public void printTutorTimetable(Tutor tutor)
1123        {
1124            ArrayList<Tutorial> tutorials = getTutorials(tutor);
1125            if(tutorials == null || tutorials.size() == 0)
1126            {
1127                System.out.println("No tutorials assigned");
1128            }
1129
1130            tutor.printTimetable(tutorials);
1131        }
1132
1133        public void handlePrintStudentTimetable()
1134        {
1135            System.out.print("Enter Student ID: ");
1136            Student student = getStudent(scan.nextLine());
1137            if(student == null)
1138            {
1139                System.out.println("No student with this ID");
1140                hold();
```

```
1141            return;
1142        }
1143
1144        printStudentTimetable(student);
1145        hold();
1146    }
1147
1148    public void printStudentTimetable(Student student)
1149    {
1150        if(student.getCurrentEnrolments().size() == 0)
1151        {
1152            System.out.println("No enrolments for this period");
1153        }
1154
1155        student.printTimetable();
1156    }
1157
1158    public void displayMap(Map m)
1159    {
1160        Set keySet = m.keySet();
1161        Iterator iterator = keySet.iterator();
1162        while(iterator.hasNext())
1163        {
1164            String key = (String) iterator.next();
1165            System.out.println(m.get(key) + "\n");
1166        }
1167        hold();
1168    }
1169
1170    /**
1171     * Bonus 1) Separate menus for separate actors (with distinct passwords)
1172     */
1173    public Student studentLogin()
1174    {
1175        Student student = null;
1176        while(student == null)
1177        {
1178            System.out.print("Enter Student ID: ");
1179            String sNum = scan.nextLine();
1180            student = students.get(sNum);
1181            if(sNum.equals(""))
1182            {
1183                System.out.println("Returning...\n");
1184                return null;
1185            }
1186            if(student == null)
1187            {
1188                System.out.println("Student ID not found\n");
1189            }
1190        }
1191
1192        boolean correctPassword = false;
1193        while(!correctPassword)
1194        {
1195            System.out.print("Enter Password: ");
1196            String password = scan.nextLine();
1197            if(password.equals(""))
```

```
            {
                System.out.println("Returning...\n");
                return null;
            }

            if(password.equals(student.getPassword()))
            {
                return student;
            }
            else
            {
                System.out.println("Incorrect Password\n");
            }
        }

        return null;
    }

    public void changeStudentPassword(Student student)
    {
        boolean currentPassValid = false;
        while(!currentPassValid)
        {
            System.out.print("Enter current passowrd: ");
            String currentPass = scan.nextLine();

            if(currentPass.equals(""))
            {
                System.out.println("Returning...\n");
                return;
            }

            if(currentPass.equals(student.getPassword()))
            {
                currentPassValid = true;
            }
            else
            {
                System.out.println("Incorrect password\n");
            }
        }

        boolean newPassConfirmed = false;
        while(!newPassConfirmed)
        {
            System.out.print("\nEnter new password: ");
            String newPass = scan.nextLine();

            if(newPass.equals(""))
            {
                System.out.println("Returning...\n");
                return;
            }

            System.out.print("Confirm new password: ");
            String newPassConf = scan.nextLine();

```

```
1255            if(newPassConf.equals(""))
1256            {
1257                return;
1258            }
1259
1260            if(newPass.equals(newPassConf))
1261            {
1262                student.setPassword(newPass);
1263                System.out.println("\nPassword successfully changed");
1264                return;
1265            }
1266            else
1267            {
1268                System.out.println("Passwords don't match");
1269            }
1270        }
1271    }
1272
1273    public boolean adminLogin()
1274    {
1275        boolean correctPassword = false;
1276        while(!correctPassword)
1277        {
1278            System.out.print("Enter Admin Password: ");
1279            String password = scan.nextLine();
1280            if(password.equals(""))
1281            {
1282                System.out.println("Returning...\n");
1283                return false;
1284            }
1285
1286            if(password.equals(adminPassword))
1287            {
1288                return true;
1289            }
1290            else
1291            {
1292                System.out.println("Incorrect Password\n");
1293            }
1294        }
1295
1296        return false;
1297    }
1298
1299    public void changeAdminPassword()
1300    {
1301        boolean currentPassValid = false;
1302        while(!currentPassValid)
1303        {
1304            System.out.print("Enter current admin passowrd: ");
1305            String currentPass = scan.nextLine();
1306
1307            if(currentPass.equals(""))
1308            {
1309                System.out.println("Returning...\n");
1310                return;
1311            }
```

```
1312
1313            if(currentPass.equals(adminPassword))
1314            {
1315                currentPassValid = true;
1316            }
1317            else
1318            {
1319                System.out.println("Incorrect password\n");
1320            }
1321        }
1322
1323        boolean newPassConfirmed = false;
1324        while(!newPassConfirmed)
1325        {
1326            System.out.print("\nEnter new admin password: ");
1327            String newPass = scan.nextLine();
1328
1329            if(newPass.equals(""))
1330            {
1331                System.out.println("Returning...\n");
1332                return;
1333            }
1334
1335            System.out.print("Confirm new admin password: ");
1336            String newPassConf = scan.nextLine();
1337
1338            if(newPassConf.equals(""))
1339            {
1340                return;
1341            }
1342
1343            if(newPass.equals(newPassConf))
1344            {
1345                adminPassword = newPass;
1346                System.out.println("\nAdmin password successfully changed");
1347                return;
1348            }
1349            else
1350            {
1351                System.out.println("Passwords don't match");
1352            }
1353        }
1354    }
1355
1356    /**
1357     * Bonus 2) Ability to save and retrieve the objects
1358     */
1359    public void writeDataToFile()
1360    {
1361        String fileStr = "data.dat";
1362        FileOutputStream fOut = null;
1363        ObjectOutputStream objOut = null;
1364
1365        try
1366        {
1367            fOut = new FileOutputStream(fileStr);
1368            objOut = new ObjectOutputStream(fOut);
```

```
1369
1370            objOut.writeObject(courses);
1371            objOut.writeObject(venues);
1372            objOut.writeObject(lecturers);
1373            objOut.writeObject(tutors);
1374            objOut.writeObject(students);
1375            objOut.writeObject(censusDate);
1376            objOut.writeObject(adminPassword);
1377
1378            fOut.close();
1379        }
1380        catch(Exception e)
1381        {
1382            System.out.println("Error writing data to file '" + fileStr + "'");
1383            return;
1384        }
1385    }
1386
1387    public void readDataFromFile()
1388    {
1389        String fileStr = "data.dat";
1390        FileInputStream fIn = null;
1391        ObjectInputStream objIn = null;
1392
1393        try
1394        {
1395            fIn = new FileInputStream(fileStr);
1396            objIn = new ObjectInputStream(fIn);
1397
1398            courses = (HashMap<String, Course>) objIn.readObject();
1399            venues = (HashMap<String, Venue>) objIn.readObject();
1400            lecturers = (HashMap<String, Lecturer>) objIn.readObject();
1401            tutors = (HashMap<String, Tutor>) objIn.readObject();
1402            students = (HashMap<String, Student>) objIn.readObject();
1403            censusDate = (LocalDate) objIn.readObject();
1404            adminPassword = (String) objIn.readObject();
1405        }
1406        catch(Exception e)
1407        {
1408            System.out.println("Error reading data from file '"
1409                    + fileStr + "'");
1410        }
1411    }
1412
1413    public Course getCourse(String ID)
1414    {
1415        return courses.get(ID);
1416    }
1417
1418    public Venue getVenue(String location)
1419    {
1420        return venues.get(location);
1421    }
1422
1423    public Lecturer getLecturer(String eNo)
1424    {
1425        return lecturers.get(eNo);
```

```
1426        }
1427
1428        public Tutor getTutor(String eNo)
1429        {
1430            return tutors.get(eNo);
1431        }
1432
1433        public Applicant getApplicant(String eNo)
1434        {
1435            return applicants.get(eNo);
1436        }
1437
1438        public Student getStudent(String sNo)
1439        {
1440            return students.get(sNo);
1441        }
1442
1443        public ArrayList<Lecture> getLectures(Lecturer lecturer)
1444        {
1445            return lecturer.getLectures();
1446        }
1447
1448        public ArrayList<Tutorial> getTutorials(Tutor tutor)
1449        {
1450            return tutor.getTutorials();
1451        }
1452
1453        public ArrayList<Lesson> getLessons(Venue venue)
1454        {
1455            return venue.getLessons();
1456        }
1457
1458        public Lecture getLecture(CourseOffering offering)
1459        {
1460            return offering.getLecture();
1461        }
1462
1463        public ArrayList<Tutorial> getTutorials(CourseOffering offering)
1464        {
1465            return offering.getTutorials();
1466        }
1467
1468        public CourseOffering getCourseOffering(String ID, int year, int sem)
1469        {
1470            Course c = courses.get(ID);
1471            if(c == null)
1472            {
1473                return null;
1474            }
1475            return c.getOffering(year, semester);
1476        }
1477    }
1478
```