```java
import java.time.*;
import java.util.ArrayList;
import org.junit.*;
import static org.junit.Assert.*;
import org.junit.Test;

public class AppTest
{
    App app;
    Course course1;
    Course course2;
    Course course3;
    Course course4;
    Course course5;
    Course course6;

    @Before
    public void setUp()
    {
        app = new App();
        course1 = app.courses.get("P101");
        course2 = app.courses.get("WP1");
        course3 = app.courses.get("UI1");
        course4 = app.courses.get("MATH");
        course5 = app.courses.get("CSYS");
        course6 = app.courses.get("P102");

    }

    @After
    public void tearDown()
    {

    }

    // Checking offerings created can be accessed via getCourseOffering
    @Test
    public void testGetCourseOffering() throws PreExistException
    {
        app.createOffering(course1, 200, 2015, 1);
        assertNotEquals("Offering for 2015 semester 1 exists",
                app.getCourseOffering("P101", 2015, 1), null);
        assertEquals("Offering for 2016 does not exist",
                app.getCourseOffering("P101", 2016, 1), null);
    }

    // Checking venue clashes are detected
    @Test(expected = ClashException.class)
    public void testVenueClashes() throws PreExistException, ClashException,
            UnsuitableVenueException, LessonTimeOutOfBoundsException
    {
        CourseOffering offering1 = app.createOffering(course1, 200, 2015, 1);
        CourseOffering offering2 = app.createOffering(course2, 200, 2015, 1);
        Venue venue = app.getVenue("12.10.02");
        app.addLecture(offering1, DayOfWeek.WEDNESDAY, LocalTime.of(10, 00),
                120, venue);
        app.addLecture(offering2, DayOfWeek.WEDNESDAY, LocalTime.of(11, 30),
```

```
58                  120, venue);
59          }
60
61          // Checking lecture clashes are detected
62          @Test(expected = ClashException.class)
63          public void testLectureClashes() throws PreExistException, ClashException,
64                  UnsuitableVenueException, LessonTimeOutOfBoundsException
65          {
66
67              CourseOffering offering1 = app.createOffering(course1, 200, 2015, 1);
68              CourseOffering offering2 = app.createOffering(course2, 200, 2015, 1);
69              Venue venue1 = app.getVenue("12.10.02");
70              Venue venue2 = app.getVenue("12.10.03");
71              app.addLecture(offering1, DayOfWeek.WEDNESDAY, LocalTime.of(10, 00),
72                      120, venue1);
73              app.addLecture(offering2, DayOfWeek.WEDNESDAY, LocalTime.of(11, 30),
74                      120, venue2);
75              Lecturer lecturer = app.getLecturer("e44556");
76              Lecture lecture1 = offering1.getLecture();
77              Lecture lecture2 = offering2.getLecture();
78              app.assignLecturer(lecture1, lecturer);
79              app.assignLecturer(lecture2, lecturer);
80          }
81
82          // Checks when adding a lectures Venue assignments are correctly made
83          @Test
84          public void testVenueAssignments() throws PreExistException,
85                  ClashException, UnsuitableVenueException,
86                  LessonTimeOutOfBoundsException
87          {
88              CourseOffering offering1 = app.createOffering(course1, 200, 2015, 1);
89              CourseOffering offering2 = app.createOffering(course2, 200, 2015, 1);
90              CourseOffering offering3 = app.createOffering(course3, 200, 2015, 1);
91              CourseOffering offering4 = app.createOffering(course4, 200, 2015, 1);
92              Venue venue1 = app.getVenue("12.10.02");
93              Venue venue2 = app.getVenue("12.10.03");
94              app.addLecture(offering1, DayOfWeek.WEDNESDAY, LocalTime.of(10, 00),
95                      120, venue1);
96              app.addLecture(offering2, DayOfWeek.WEDNESDAY, LocalTime.of(14, 30),
97                      120, venue2);
98              app.addLecture(offering3, DayOfWeek.THURSDAY, LocalTime.of(9, 30),
99                      120, venue2);
100             app.addLecture(offering4, DayOfWeek.FRIDAY, LocalTime.of(18, 30),
101                     120, venue2);
102             assertEquals(app.getLessons(venue1).size(), 1);
103             assertEquals(app.getLessons(venue2).size(), 3);
104         }
105
106         /**
107          * BEGIN Jason Hamilton (s3455196) test cases
108          */
109         // System should prevent students from enrolling in more than 4 course
110         // offerings
111         @Test(expected = OverEnrolmentException.class)
112         public void testStudentEnrollment() throws OverEnrolmentException,
113                 PreExistException, CensusDateExceededException,
114                 AlreadyEnrolledException, IncompletePrerequisitesException
```

```
115     {
116         // Create a student and course offerings for the student to enrol in
117         Student student = new Student("s1234567", "Jane Doe", "password",
118                 "0412345678", "1 Swanston Street, Melbourne 3000",
119                 LocalDate.of(1994, 1, 1), null);
120         CourseOffering offering1 = app.createOffering(course1, 200, 2015, 1);
121         CourseOffering offering2 = app.createOffering(course2, 200, 2015, 1);
122         CourseOffering offering3 = app.createOffering(course3, 200, 2015, 1);
123         CourseOffering offering4 = app.createOffering(course4, 200, 2015, 1);
124 
125         // Create a fifth course offering for the student to attempt to enrol in
126         CourseOffering offering5 = app.createOffering(course5, 200, 2015, 1);
127 
128         // Avoid CensusDateExceededException
129         app.changeCensusDate(LocalDate.now().plusDays(1));
130 
131         app.enrol(student, offering1);
132         app.enrol(student, offering2);
133         app.enrol(student, offering3);
134         app.enrol(student, offering4);
135         app.enrol(student, offering5);
136     }
137 
138     // System should prevent not allow a student (identified uniquely by
139     // first-name, surname and phone) to be admitted twice
140     @Test(expected = DuplicateStudentException.class)
141     public void testDuplicateStudentAdmittance()
142             throws DuplicateStudentException
143     {
144 
145         app.admitStudent("s3455196", "Jane Doe", "password",
146                 "0412345678", "1 Swanston Street, Melbourne 3000",
147                 LocalDate.of(1994, 1, 1), new ArrayList<Course>());
148 
149         app.admitStudent("s3455197", "Jane Doe", "password",
150                 "0412345678", "1 Swanston Street, Melbourne 3000",
151                 LocalDate.of(1994, 1, 1), new ArrayList<Course>());
152 
153     }
154 
155     // System should not allow students to enrol into a course offering without
156     // the necessary prerequisites
157     @Test
158     public void testStudentPrerequisites()
159             throws IncompletePrerequisitesException, PreExistException
160     {
161         // Create a student and course offerings for the student to enrol in
162         Student student = new Student("s1234567", "Jane Doe", "password",
163                 "0412345678", "1 Swanston Street, Melbourne 3000",
164                 LocalDate.of(1994, 1, 1), new ArrayList<Course>());
165 
166         CourseOffering offering = app.createOffering(course6, 200, 2015, 1);
167 
168         // This student hasn't completed Programming 1 so they shouldn't be able
169         // to enrol into Programming 2
170         assertFalse(student.meetsPrerequisites(offering.getCourse()));
171     }
```

```
172
173         // System should prevent a tutor being appointed with the same ID as an
174         // existing tutor.
175         @Test(expected = DuplicateTutorException.class)
176         public void testDuplicateTutor() throws DuplicateTutorException
177         {
178
179             app.tutors.put("e1234567", new Tutor("e1234567", "Don Draper",
180                     "Creative Director", "Madison Avenue",
181                     "0412345678"));
182
183             Applicant applicant = new Applicant("e1234567", "Don Draper",
184                     "Creative Director");
185
186             if(app.tutors.get(applicant.getENo()) != null)
187             {
188                 throw new DuplicateTutorException("A tutor with that ID already "
189                         + "exists.");
190             }
191         }
192
193         // System should prevent a lesson booking on the weekend
194         @Test(expected = LessonTimeOutOfBoundsException.class)
195         public void testLessonOnWeekend() throws PreExistException,
196                 ClashException, UnsuitableVenueException,
197                 LessonTimeOutOfBoundsException
198         {
199             CourseOffering offering = app.createOffering(course1, 200, 2015, 1);
200             Venue venue = app.getVenue("12.10.02");
201
202             app.addLecture(offering, DayOfWeek.SATURDAY, LocalTime.of(12, 00),
203                     120, venue);
204         }
205
206         /**
207          * END Jason Hamilton (s3455196) test cases
208          */
209
210         /**
211          * BEGIN Stuart Parker (s3390317) test cases
212          */
213
214         // System should prevent students from enrolling or withdrawing after the
215         // census date
216         @Test(expected = CensusDateExceededException.class)
217         public void testWithdrawAfterCensusDate() throws PreExistException,
218                 OverEnrolmentException, CensusDateExceededException,
219                 AlreadyEnrolledException, IncompletePrerequisitesException
220         {
221             Student student = app.students.get("s1234567");
222             CourseOffering offering = app.createOffering(course1, 200, 2015, 1);
223
224             // Make census date yesterday
225             app.changeCensusDate(LocalDate.now().minusDays(1));
226
227             app.enrol(student, offering);
228         }
```

```
229
230         @Test(expected = CensusDateExceededException.class)
231         public void testEnrolAfterCensusDate() throws PreExistException,
232                 OverEnrolmentException, CensusDateExceededException,
233                 TutorialEnrolledException, AlreadyEnrolledException,
234                 IncompletePrerequisitesException
235         {
236             Student student = app.students.get("s1234567");
237             CourseOffering offering = app.createOffering(course1, 200, 2015, 1);
238 
239             // Make census date tomorrow
240             app.changeCensusDate(LocalDate.now().plusDays(1));
241 
242             app.enrol(student, offering);
243 
244             // Make census date yesterday
245             app.changeCensusDate(LocalDate.now().minusDays(1));
246 
247             ArrayList<CourseEnrolment> enrolments = student.getCurrentEnrolments();
248 
249             app.withdrawFromCourse(student, enrolments.get(0));
250         }
251 
252         // System should prevent any tutor from being assigned two tutorials at the
253         // same time
254         @Test(expected = ClashException.class)
255         public void testOneTutorTwoTutorialsSameTime() throws PreExistException,
256                 ClashException, UnsuitableVenueException,
257                 LessonTimeOutOfBoundsException
258         {
259             CourseOffering offering1 = app.createOffering(course1, 200, 2015, 1);
260             CourseOffering offering2 = app.createOffering(course2, 100, 2015, 1);
261 
262             Tutor tutor = app.tutors.get("e12345");
263 
264             Venue venue1 = app.venues.get("10.10.22");
265             Venue venue2 = app.venues.get("10.10.23");
266 
267             Tutorial tutorial1 = app.addTutorial(offering1, DayOfWeek.MONDAY,
268                     LocalTime.of(10, 00), 120, venue1);
269             Tutorial tutorial2 = app.addTutorial(offering2, DayOfWeek.MONDAY,
270                     LocalTime.of(11, 00), 120, venue2);
271 
272             app.assignTutor(tutorial1, tutor);
273             app.assignTutor(tutorial2, tutor);
274         }
275 
276         // System should prevent students from registering into a tutorial that is
277         // already full
278         @Test(expected = OverCapacityException.class)
279         public void testTutorialOverEnrollment() throws PreExistException,
280                 ClashException, UnsuitableVenueException, OverEnrolmentException,
281                 CensusDateExceededException, OverCapacityException,
282                 LessonTimeOutOfBoundsException, AlreadyEnrolledException,
283                 IncompletePrerequisitesException, TutorialAlreadyEnrolledException
284         {
285             CourseOffering offering = app.createOffering(course1, 200, 2015, 1);
```

```
286             Venue venue = new Venue("08.09.42", 1, Venue.Purpose.TUTELAB);
287             Student student1 = app.students.get("s1234567");
288             Student student2 = new Student("s1234568", "Test McTesterson", "test",
289                     "03 5678 1324", "125 Fake St, Fakeville VIC, 3000",
290                     LocalDate.of(1994, 1, 2), new ArrayList<Course>());
291
292             Tutorial tutorial = app.addTutorial(offering, DayOfWeek.FRIDAY,
293                     LocalTime.of(14, 00), 120, venue);
294
295             // Enrol students in course offering
296             app.changeCensusDate(LocalDate.now().plusDays(1));
297             CourseEnrolment s1enrolment = app.enrol(student1, offering);
298             CourseEnrolment s2enrolment = app.enrol(student2, offering);
299
300             student1.registerTutorial(s1enrolment, tutorial);
301             student2.registerTutorial(s2enrolment, tutorial);
302         }
303
304         // System should prevent a lecture being assigned a tutelab venue
305         @Test(expected = UnsuitableVenueException.class)
306         public void testInvalidLectureVenue() throws PreExistException,
307                 ClashException, UnsuitableVenueException,
308                 LessonTimeOutOfBoundsException
309         {
310             CourseOffering offering = app.createOffering(course1, 200, 2015, 1);
311             Venue venue = app.venues.get("10.10.22");
312
313             app.addLecture(offering, DayOfWeek.MONDAY, LocalTime.of(10, 00), 120,
314                     venue);
315         }
316
317         // System should prevent a tutorial being assigned a lecture venue
318         @Test(expected = UnsuitableVenueException.class)
319         public void testInvalidTutorialVenue() throws PreExistException,
320                 ClashException, UnsuitableVenueException,
321                 LessonTimeOutOfBoundsException
322         {
323             CourseOffering offering = app.createOffering(course1, 200, 2015, 1);
324             Venue venue = app.venues.get("12.10.02");
325
326             app.addTutorial(offering, DayOfWeek.MONDAY, LocalTime.of(10, 00), 120,
327                     venue);
328         }
329
330         /**
331          * END Stuart Parker (s3390317) test cases
332          */
333
334         /**
335          * BEGIN Aidan Cyr (s3471910) test cases
336          */
337
338         // TODO 3 more from assignment spec list
339         // TODO 2 more not from assignment spec list
340
341         /**
342          * END Aidan Cyr (s3471910) test cases
```

```
343         */
344
345        /**
346         * BEGIN Jake Seeary (s3430163) test cases
347         */
348
349        // TODO 3 more from assignment spec list
350        // TODO 2 more not from assignment spec list
351
352        /**
353         * END Jake Seeary (s3430163) test cases
354         */
355
356        /**
357         * BEGIN Test cases for anyone to claim
358         */
359
360        // System should prevent more than one offering per course in any semester
361        // ** from given list **
362        @Test(expected = PreExistException.class)
363        public void testNoDuplicateOfferings() throws PreExistException,
364                ClashException
365        {
366            assertEquals("P101", course1.getID());
367            app.createOffering(course1, 200, 2015, 1);
368            app.createOffering(course1, 200, 2015, 1);
369        }
370
371        // System should prevent a lesson booking outside of allowable venue times
372        // ** NOT from given list **
373        @Test(expected = LessonTimeOutOfBoundsException.class)
374        public void testLessonStartTooEarly() throws PreExistException,
375                ClashException, UnsuitableVenueException,
376                LessonTimeOutOfBoundsException
377        {
378            CourseOffering offering = app.createOffering(course1, 200, 2015, 1);
379            Venue venue = app.getVenue("12.10.02");
380
381            app.addLecture(offering, DayOfWeek.WEDNESDAY, LocalTime.of(6, 00),
382                    120, venue);
383        }
384
385        @Test(expected = LessonTimeOutOfBoundsException.class)
386        public void testLessonStartTooLate() throws PreExistException,
387                ClashException, UnsuitableVenueException,
388                LessonTimeOutOfBoundsException
389        {
390            CourseOffering offering = app.createOffering(course1, 200, 2015, 1);
391            Venue venue = app.getVenue("12.10.02");
392
393            app.addLecture(offering, DayOfWeek.WEDNESDAY, LocalTime.of(22, 00),
394                    120, venue);
395        }
396
397        @Test(expected = LessonTimeOutOfBoundsException.class)
398        public void testLessonEndTooLate() throws PreExistException,
399                ClashException, UnsuitableVenueException,
```

```
400             LessonTimeOutOfBoundsException
401     {
402         CourseOffering offering = app.createOffering(course1, 200, 2015, 1);
403         Venue venue = app.getVenue("12.10.02");
404 
405         app.addLecture(offering, DayOfWeek.WEDNESDAY, LocalTime.of(20, 00),
406                 120, venue);
407     }
408 
409     // System should ensure each timetable is complete and consistent (matches
410     // with others)
411     // ** from given list **
412     @Test()
413     public void testTimetableConsistency() throws PreExistException,
414             ClashException, LessonTimeOutOfBoundsException,
415             OverEnrolmentException, OverCapacityException,
416             CensusDateExceededException, AlreadyEnrolledException,
417             IncompletePrerequisitesException, TutorialAlreadyEnrolledException,
418             UnsuitableVenueException
419     {
420         CourseOffering offering1 = app.createOffering(course1, 200, 2015, 1);
421         CourseOffering offering2 = app.createOffering(course2, 200, 2015, 1);
422         Venue lectureVenue = app.venues.get("12.10.02");
423         Venue tutorialVenue = app.venues.get("10.10.22");
424         Student student = app.students.get("s1234567");
425         Lecturer lecturer = app.lecturers.get("e54321");
426         Tutor tutor = app.tutors.get("e12345");
427 
428         // Avoid CensusDateExceededException
429         app.changeCensusDate(LocalDate.now().plusDays(1));
430 
431         offering1.addLecture(DayOfWeek.MONDAY, LocalTime.of(10, 00), 120,
432                 lectureVenue);
433         lecturer.assign(offering1.getLecture());
434         Tutorial tutorial1 = offering1.addTutorial(DayOfWeek.TUESDAY,
435                 LocalTime.of(12, 00), 120, tutorialVenue);
436         tutor.assign(tutorial1);
437         CourseEnrolment enrolment1 = app.enrol(student, offering1);
438         student.registerTutorial(enrolment1, tutorial1);
439 
440         offering2.addLecture(DayOfWeek.WEDNESDAY, LocalTime.of(14, 00), 120,
441                 lectureVenue);
442         lecturer.assign(offering2.getLecture());
443         Tutorial tutorial2 = offering2.addTutorial(DayOfWeek.THURSDAY,
444                 LocalTime.of(12, 00), 120, tutorialVenue);
445         tutor.assign(tutorial2);
446         CourseEnrolment enrolment2 = app.enrol(student, offering2);
447         student.registerTutorial(enrolment2, tutorial2);
448 
449         app.printLecturerTimeTable(lecturer);
450         app.printTutorTimetable(tutor);
451         app.printStudentTimetable(student);
452         app.printVenueTimetable(lectureVenue);
453         app.printVenueTimetable(tutorialVenue);
454     }
455 
456     /**
```

```
457         * END Test cases for anyone to claim
458         */
459    }
460
```