```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),
                120, venue);
    }

    // Checking lecture clashes are detected
    @Test(expected = ClashException.class)
    public void testLectureClashes() throws PreExistException, ClashException,
            UnsuitableVenueException, LessonTimeOutOfBoundsException
    {

        CourseOffering offering1 = app.createOffering(course1, 200, 2015, 1);
        CourseOffering offering2 = app.createOffering(course2, 200, 2015, 1);
        Venue venue1 = app.getVenue("12.10.02");
        Venue venue2 = app.getVenue("12.10.03");
        app.addLecture(offering1, DayOfWeek.WEDNESDAY, LocalTime.of(10, 00),
                120, venue1);
        app.addLecture(offering2, DayOfWeek.WEDNESDAY, LocalTime.of(11, 30),
                120, venue2);
        Lecturer lecturer = app.getLecturer("e44556");
        Lecture lecture1 = offering1.getLecture();
        Lecture lecture2 = offering2.getLecture();
        app.assignLecturer(lecture1, lecturer);
        app.assignLecturer(lecture2, lecturer);
    }

    // Checks when adding a lectures Venue assignments are correctly made
    @Test
    public void testVenueAssignments() throws PreExistException,
            ClashException, UnsuitableVenueException,
            LessonTimeOutOfBoundsException
    {
        CourseOffering offering1 = app.createOffering(course1, 200, 2015, 1);
        CourseOffering offering2 = app.createOffering(course2, 200, 2015, 1);
        CourseOffering offering3 = app.createOffering(course3, 200, 2015, 1);
        CourseOffering offering4 = app.createOffering(course4, 200, 2015, 1);
        Venue venue1 = app.getVenue("12.10.02");
        Venue venue2 = app.getVenue("12.10.03");
        app.addLecture(offering1, DayOfWeek.WEDNESDAY, LocalTime.of(10, 00),
                120, venue1);
        app.addLecture(offering2, DayOfWeek.WEDNESDAY, LocalTime.of(14, 30),
                120, venue2);
        app.addLecture(offering3, DayOfWeek.THURSDAY, LocalTime.of(9, 30),
                120, venue2);
        app.addLecture(offering4, DayOfWeek.FRIDAY, LocalTime.of(18, 30),
                120, venue2);
        assertEquals(app.getLessons(venue1).size(), 1);
        assertEquals(app.getLessons(venue2).size(), 3);
    }

    /**
     * BEGIN Jason Hamilton (s3455196) test cases
     */
    // System should prevent students from enrolling in more than 4 course
    // offerings
    @Test(expected = OverEnrolmentException.class)
    public void testStudentEnrollment() throws OverEnrolmentException,
            PreExistException, CensusDateExceededException,
            AlreadyEnrolledException, IncompletePrerequisitesException
```

```java
    {
        // Create a student and course offerings for the student to enrol in
        Student student = new Student("s1234567", "Jane Doe", "password",
                "0412345678", "1 Swanston Street, Melbourne 3000",
                LocalDate.of(1994, 1, 1), null);
        CourseOffering offering1 = app.createOffering(course1, 200, 2015, 1);
        CourseOffering offering2 = app.createOffering(course2, 200, 2015, 1);
        CourseOffering offering3 = app.createOffering(course3, 200, 2015, 1);
        CourseOffering offering4 = app.createOffering(course4, 200, 2015, 1);

        // Create a fifth course offering for the student to attempt to enrol in
        CourseOffering offering5 = app.createOffering(course5, 200, 2015, 1);

        // Avoid CensusDateExceededException
        app.changeCensusDate(LocalDate.now().plusDays(1));

        app.enrol(student, offering1);
        app.enrol(student, offering2);
        app.enrol(student, offering3);
        app.enrol(student, offering4);
        app.enrol(student, offering5);
    }

    // System should prevent not allow a student (identified uniquely by
    // first-name, surname and phone) to be admitted twice
    @Test(expected = DuplicateStudentException.class)
    public void testDuplicateStudentAdmittance()
            throws DuplicateStudentException
    {

        app.admitStudent("s3455196", "Jane Doe", "password",
                "0412345678", "1 Swanston Street, Melbourne 3000",
                LocalDate.of(1994, 1, 1), new ArrayList<Course>());

        app.admitStudent("s3455197", "Jane Doe", "password",
                "0412345678", "1 Swanston Street, Melbourne 3000",
                LocalDate.of(1994, 1, 1), new ArrayList<Course>());

    }

    // System should not allow students to enrol into a course offering without
    // the necessary prerequisites
    @Test
    public void testStudentPrerequisites()
            throws IncompletePrerequisitesException, PreExistException
    {
        // Create a student and course offerings for the student to enrol in
        Student student = new Student("s1234567", "Jane Doe", "password",
                "0412345678", "1 Swanston Street, Melbourne 3000",
                LocalDate.of(1994, 1, 1), new ArrayList<Course>());

        CourseOffering offering = app.createOffering(course6, 200, 2015, 1);

        // This student hasn't completed Programming 1 so they shouldn't be able
        // to enrol into Programming 2
        assertFalse(student.meetsPrerequisites(offering.getCourse()));
    }

    // System should prevent a tutor being appointed with the same ID as an
    // existing tutor.
    @Test(expected = DuplicateTutorException.class)
    public void testDuplicateTutor() throws DuplicateTutorException
    {

        app.tutors.put("e1234567", new Tutor("e1234567", "Don Draper",
                "Creative Director", "Madison Avenue",
                "0412345678"));

        Applicant applicant = new Applicant("e1234567", "Don Draper",
                "Creative Director");

        if(app.tutors.get(applicant.getENo()) != null)
        {
            throw new DuplicateTutorException("A tutor with that ID already "
                    + "exists.");
        }
    }

    // System should prevent a lesson booking on the weekend
    @Test(expected = LessonTimeOutOfBoundsException.class)
    public void testLessonOnWeekend() throws PreExistException,
            ClashException, UnsuitableVenueException,
            LessonTimeOutOfBoundsException
    {
        CourseOffering offering = app.createOffering(course1, 200, 2015, 1);
        Venue venue = app.getVenue("12.10.02");

        app.addLecture(offering, DayOfWeek.SATURDAY, LocalTime.of(12, 00),
                120, venue);
    }

    /**
     * END Jason Hamilton (s3455196) test cases
     */

    /**
     * BEGIN Stuart Parker (s3390317) test cases
     */

    // System should prevent students from enrolling or withdrawing after the
    // census date
    @Test(expected = CensusDateExceededException.class)
    public void testWithdrawAfterCensusDate() throws PreExistException,
            OverEnrolmentException, CensusDateExceededException,
            AlreadyEnrolledException, IncompletePrerequisitesException
    {
        Student student = app.students.get("s1234567");
        CourseOffering offering = app.createOffering(course1, 200, 2015, 1);

        // Make census date yesterday
        app.changeCensusDate(LocalDate.now().minusDays(1));

        app.enrol(student, offering);
    }

    @Test(expected = CensusDateExceededException.class)
    public void testEnrolAfterCensusDate() throws PreExistException,
            OverEnrolmentException, CensusDateExceededException,
            TutorialEnrolledException, AlreadyEnrolledException,
            IncompletePrerequisitesException
    {
        Student student = app.students.get("s1234567");
        CourseOffering offering = app.createOffering(course1, 200, 2015, 1);

        // Make census date tomorrow
        app.changeCensusDate(LocalDate.now().plusDays(1));

        app.enrol(student, offering);

        // Make census date yesterday
        app.changeCensusDate(LocalDate.now().minusDays(1));

        ArrayList<CourseEnrolment> enrolments = student.getCurrentEnrolments();

        app.withdrawFromCourse(student, enrolments.get(0));
    }

    // System should prevent any tutor from being assigned two tutorials at the
    // same time
    @Test(expected = ClashException.class)
    public void testOneTutorTwoTutorialsSameTime() throws PreExistException,
            ClashException, UnsuitableVenueException,
            LessonTimeOutOfBoundsException
    {
        CourseOffering offering1 = app.createOffering(course1, 200, 2015, 1);
        CourseOffering offering2 = app.createOffering(course2, 100, 2015, 1);

        Tutor tutor = app.tutors.get("e12345");

        Venue venue1 = app.venues.get("10.10.22");
        Venue venue2 = app.venues.get("10.10.23");

        Tutorial tutorial1 = app.addTutorial(offering1, DayOfWeek.MONDAY,
                LocalTime.of(10, 00), 120, venue1);
        Tutorial tutorial2 = app.addTutorial(offering2, DayOfWeek.MONDAY,
                LocalTime.of(11, 00), 120, venue2);

        app.assignTutor(tutorial1, tutor);
        app.assignTutor(tutorial2, tutor);
    }

    // System should prevent students from registering into a tutorial that is
    // already full
    @Test(expected = OverCapacityException.class)
    public void testTutorialOverEnrollment() throws PreExistException,
            ClashException, UnsuitableVenueException, OverEnrolmentException,
            CensusDateExceededException, OverCapacityException,
            LessonTimeOutOfBoundsException, AlreadyEnrolledException,
            IncompletePrerequisitesException, TutorialAlreadyEnrolledException
    {
        CourseOffering offering = app.createOffering(course1, 200, 2015, 1);
        Venue venue = new Venue("08.09.42", 1, Venue.Purpose.TUTELAB);
        Student student1 = app.students.get("s1234567");
        Student student2 = new Student("s1234568", "Test McTesterson", "test",
                "03 5678 1324", "125 Fake St, Fakeville VIC, 3000",
                LocalDate.of(1994, 1, 2), new ArrayList<Course>());

        Tutorial tutorial = app.addTutorial(offering, DayOfWeek.FRIDAY,
                LocalTime.of(14, 00), 120, venue);

        // Enrol students in course offering
        app.changeCensusDate(LocalDate.now().plusDays(1));
        CourseEnrolment s1enrolment = app.enrol(student1, offering);
        CourseEnrolment s2enrolment = app.enrol(student2, offering);

        student1.registerTutorial(s1enrolment, tutorial);
        student2.registerTutorial(s2enrolment, tutorial);
    }

    // System should prevent a lecture being assigned a tutelab venue
    @Test(expected = UnsuitableVenueException.class)
    public void testInvalidLectureVenue() throws PreExistException,
            ClashException, UnsuitableVenueException,
            LessonTimeOutOfBoundsException
    {
        CourseOffering offering = app.createOffering(course1, 200, 2015, 1);
        Venue venue = app.venues.get("10.10.22");

        app.addLecture(offering, DayOfWeek.MONDAY, LocalTime.of(10, 00), 120,
                venue);
    }

    // System should prevent a tutorial being assigned a lecture venue
    @Test(expected = UnsuitableVenueException.class)
    public void testInvalidTutorialVenue() throws PreExistException,
            ClashException, UnsuitableVenueException,
            LessonTimeOutOfBoundsException
    {
        CourseOffering offering = app.createOffering(course1, 200, 2015, 1);
        Venue venue = app.venues.get("12.10.02");

        app.addTutorial(offering, DayOfWeek.MONDAY, LocalTime.of(10, 00), 120,
                venue);
    }

    /**
     * END Stuart Parker (s3390317) test cases
     */

    /**
     * BEGIN Aidan Cyr (s3471910) test cases
     */

    // TODO 3 more from assignment spec list
    // TODO 2 more not from assignment spec list

    /**
     * END Aidan Cyr (s3471910) test cases
```

```
         */

        /**
         * BEGIN Jake Seeary (s3430163) test cases
         */

        // TODO 3 more from assignment spec list
        // TODO 2 more not from assignment spec list

        /**
         * END Jake Seeary (s3430163) test cases
         */

        /**
         * BEGIN Test cases for anyone to claim
         */

        // System should prevent more than one offering per course in any semester
        // ** from given list **
        @Test(expected = PreExistException.class)
        public void testNoDuplicateOfferings() throws PreExistException,
                ClashException
        {
            assertEquals("P101", course1.getID());
            app.createOffering(course1, 200, 2015, 1);
            app.createOffering(course1, 200, 2015, 1);
        }

        // System should prevent a lesson booking outside of allowable venue times
        // ** NOT from given list **
        @Test(expected = LessonTimeOutOfBoundsException.class)
        public void testLessonStartTooEarly() throws PreExistException,
                ClashException, UnsuitableVenueException,
                LessonTimeOutOfBoundsException
        {
            CourseOffering offering = app.createOffering(course1, 200, 2015, 1);
            Venue venue = app.getVenue("12.10.02");

            app.addLecture(offering, DayOfWeek.WEDNESDAY, LocalTime.of(6, 00),
                    120, venue);
        }

        @Test(expected = LessonTimeOutOfBoundsException.class)
        public void testLessonStartTooLate() throws PreExistException,
                ClashException, UnsuitableVenueException,
                LessonTimeOutOfBoundsException
        {
            CourseOffering offering = app.createOffering(course1, 200, 2015, 1);
            Venue venue = app.getVenue("12.10.02");

            app.addLecture(offering, DayOfWeek.WEDNESDAY, LocalTime.of(22, 00),
                    120, venue);
        }

        @Test(expected = LessonTimeOutOfBoundsException.class)
        public void testLessonEndTooLate() throws PreExistException,
                ClashException, UnsuitableVenueException,
```

```java
                LessonTimeOutOfBoundsException
        {
            CourseOffering offering = app.createOffering(course1, 200, 2015, 1);
            Venue venue = app.getVenue("12.10.02");

            app.addLecture(offering, DayOfWeek.WEDNESDAY, LocalTime.of(20, 00),
                    120, venue);
        }

        // System should ensure each timetable is complete and consistent (matches
        // with others)
        // ** from given list **
        @Test()
        public void testTimetableConsistency() throws PreExistException,
                ClashException, LessonTimeOutOfBoundsException,
                OverEnrolmentException, OverCapacityException,
                CensusDateExceededException, AlreadyEnrolledException,
                IncompletePrerequisitesException, TutorialAlreadyEnrolledException,
                UnsuitableVenueException
        {
            CourseOffering offering1 = app.createOffering(course1, 200, 2015, 1);
            CourseOffering offering2 = app.createOffering(course2, 200, 2015, 1);
            Venue lectureVenue = app.venues.get("12.10.02");
            Venue tutorialVenue = app.venues.get("10.10.22");
            Student student = app.students.get("s1234567");
            Lecturer lecturer = app.lecturers.get("e54321");
            Tutor tutor = app.tutors.get("e12345");

            // Avoid CensusDateExceededException
            app.changeCensusDate(LocalDate.now().plusDays(1));

            offering1.addLecture(DayOfWeek.MONDAY, LocalTime.of(10, 00), 120,
                    lectureVenue);
            lecturer.assign(offering1.getLecture());
            Tutorial tutorial1 = offering1.addTutorial(DayOfWeek.TUESDAY,
                    LocalTime.of(12, 00), 120, tutorialVenue);
            tutor.assign(tutorial1);
            CourseEnrolment enrolment1 = app.enrol(student, offering1);
            student.registerTutorial(enrolment1, tutorial1);

            offering2.addLecture(DayOfWeek.WEDNESDAY, LocalTime.of(14, 00), 120,
                    lectureVenue);
            lecturer.assign(offering2.getLecture());
            Tutorial tutorial2 = offering2.addTutorial(DayOfWeek.THURSDAY,
                    LocalTime.of(12, 00), 120, tutorialVenue);
            tutor.assign(tutorial2);
            CourseEnrolment enrolment2 = app.enrol(student, offering2);
            student.registerTutorial(enrolment2, tutorial2);

            app.printLecturerTimeTable(lecturer);
            app.printTutorTimetable(tutor);
            app.printStudentTimetable(student);
            app.printVenueTimetable(lectureVenue);
            app.printVenueTimetable(tutorialVenue);
        }

        /**
```

```
         * END Test cases for anyone to claim
         */
}

```