```
/**
 *
 */
package edu.ncsu.csc216.wolf_scheduler.course;

/**
 * Creates Course class for WolfScheduler
 *
 * @author Abby Finan
 */
public class Course extends Activity {
        /** Course's name. */
        private String name;
        /** Course's section. */
        private String section;
        /** Course's credit hours */
        private int credits;
        /** Course's instructor */
        private String instructorId;
        /** Course's minimum name length */
        private static final int MIN_NAME_LENGTH = 5;
        /** Course's maximum name length */
        private static final int MAX_NAME_LENGTH = 8;
        /** Course's section length */
        private static final int SECTION_LENGTH = 3;
        /** Course's minimum credit hours */
        private static final int MIN_CREDITS = 1;
        /** Course's maximum credit hours */
        private static final int MAX_CREDITS = 5;

        /**
         * Constructs a Course object with values for all fields.
         *
         * @param name         name of Course
         * @param title        title of Course
         * @param section      section of Course
         * @param credits      credit hours for Course
         * @param instructorId instructorID instructor's unity id
         * @param meetingDays  meetingDays for Course
         * @param startTime    startTime for Course
         * @param endTime      endTime for Course
         */
        public Course(String name, String title, String section, int credits, String instructorId,
String meetingDays,
                        int startTime, int endTime) {
                super(title, meetingDays, startTime, endTime);
                setName(name);
                setSection(section);
                setCredits(credits);
                setInstructorId(instructorId);
        }

        /**
         * Creates a Course with the given name, title, section, credits, instructorId,
         * and meetingDays for courses that are arranged.
         *
         * @param name         name of Course
         * @param title        title of Course
         * @param section      section of Course
         * @param credits      credit hours for Course
         * @param instructorId instructor's unity id
         * @param meetingDays  meeting days for Course as series of chars
         */
        public Course(String name, String title, String section, int credits, String instructorId,
String meetingDays) {
                this(name, title, section, credits, instructorId, meetingDays, 0, 0);
        }
```

```
/**
 * Returns the Course's name.
 *
 * @return the name
 */
public String getName() {
	return name;
}

/**
 * Sets the Course's name. If the name is null, has a length less than 5 or more
 * than 8, does not contain a space between letter characters and number
 * characters, has less than 1 or more than 4 letter characters, and not exactly
 * three trailing digit characters, an IllegalArgumentException is thrown.
 *
 * @param name the name to set
 * @throws IllegalArgumentException if the name parameter is invalid
 */
private void setName(String name) {
	// Throw exception if the name is null
	if (name == null) {
		throw new IllegalArgumentException("Invalid course name.");
	}
	// Throw exception if the name is an empty string
	// Throw exception if the name contains less than 5 character or greater than 8
	// characters
	if (name.length() < MIN_NAME_LENGTH || name.length() > MAX_NAME_LENGTH) {
		throw new IllegalArgumentException("Invalid course name.");
	}
	// Check for pattern of L[LLL] NNN
	int letter = 0;
	int digit = 0;
	boolean flag = false;
	for (int i = 0; i < name.length(); i++) {
		if (!flag) {
			if (Character.isLetter(name.charAt(i)))
				letter++;
			else if (Character.isSpaceChar(name.charAt(i)))
				flag = true;
			else {
				throw new IllegalArgumentException("Invalid course name.");
			}
		} else if (flag) {
			if (Character.isDigit(name.charAt(i)))
				digit++;
			else {
				throw new IllegalArgumentException("Invalid course name.");
			}
		}
	}
	// Check that the number of letters is correct
	if (letter < 1 || letter > 4) {
		throw new IllegalArgumentException("Invalid course name.");
	}
	// Check that the number of digits is correct
	if (digit != 3) {
		throw new IllegalArgumentException("Invalid course name.");
	}
	this.name = name;
}

/**
 * Returns the Course's section.
 *
 * @return the section
 */
public String getSection() {
	return section;
```

```
}

/**
 * Sets the Course's section.
 *
 * @param section the section to set
 */
public void setSection(String section) {
        // Conditional 1 checks for if section is null:
        if (section == null) {
                throw new IllegalArgumentException("Invalid section.");
        }
        // Conditional 2 checks for if section is not 3 characters:
        if (section.length() != SECTION_LENGTH) {
                throw new IllegalArgumentException("Invalid section.");
        }
        // Conditional 3 checks for if character is not digit:
        for (char c : section.toCharArray()) {
                if (!Character.isDigit(c)) {
                        throw new IllegalArgumentException("Invalid section.");
                }
        }
        this.section = section;
}

/**
 * Returns the Course's credit hours.
 *
 * @return the credits
 */
public int getCredits() {
        return credits;
}

/**
 * Sets the Course's credit hours.
 *
 * @param credits the credits to set
 */
public void setCredits(int credits) {
        if (credits < MIN_CREDITS || credits > MAX_CREDITS) {
                throw new IllegalArgumentException("Invalid credits.");
        }
        this.credits = credits;
}

/**
 * Returns the Course's instructor.
 *
 * @return the instructorId
 */
public String getInstructorId() {
        return instructorId;
}

/**
 * Sets the Course's instructor.
 *
 * @param instructorId the instructorId to set
 */
public void setInstructorId(String instructorId) {
        // Conditional 1 checks for if instructorID is null:
        if (instructorId == null || "".equals(instructorId)) {
                throw new IllegalArgumentException("Invalid instructor id.");
        }
        // Conditional 2 checks for if instructorID is empty:
        if (instructorId == null || instructorId.length() == 0) {
                throw new IllegalArgumentException("Invalid instructor id.");
```

```
                }
                this.instructorId = instructorId;
        }

        /**
         * Returns a comma separated value String of all Course fields.
         *
         * @return String representation of Course
         */
        @Override
        public String toString() {
                if ("A".equals(getMeetingDays())) {
                        return name + "," + getTitle() + "," + section + "," + credits + "," +
instructorId + ","
                                        + getMeetingDays();
                }
                return name + "," + getTitle() + "," + section + "," + credits + "," + instructorId
+ "," + getMeetingDays()
                                + "," + getStartTime() + "," + getEndTime();
        }

        /**
         * Creates main method for Course.
         *
         * @param args an array of command-line arguments for Course
         */
        public static void main(String[] args) {
                // TODO Auto-generated method stub

        }

        /**
         * Generates a hashCode for Course using all fields.
         *
         * @return hashCode for Activity
         */
        @Override
        public int hashCode() {
                final int prime = 31;
                int result = super.hashCode();
                result = prime * result + credits;
                result = prime * result + ((instructorId == null) ? 0 : instructorId.hashCode());
                result = prime * result + ((name == null) ? 0 : name.hashCode());
                result = prime * result + ((section == null) ? 0 : section.hashCode());
                return result;
        }

        /**
         * Compares a given object to this object for equality on all fields.
         *
         * @param obj the Object to compare
         * @return true if the objects are the same on all fields.
         */
        @Override
        public boolean equals(Object obj) {
                if (this == obj)
                        return true;
                if (!super.equals(obj))
                        return false;
                if (getClass() != obj.getClass())
                        return false;
                Course other = (Course) obj;
                if (credits != other.credits)
                        return false;
                if (instructorId == null) {
                        if (other.instructorId != null)
                                return false;
                } else if (!instructorId.equals(other.instructorId))
```

```
                        return false;
                if (name == null) {
                        if (other.name != null)
                                return false;
                } else if (!name.equals(other.name))
                        return false;
                if (section == null) {
                        if (other.section != null)
                                return false;
                } else if (!section.equals(other.section))
                        return false;
                return true;
        }

        /**
         * returns an array of length 4 containing the Course name, section, title, and
         * meeting string.
         */
        @Override
        public String[] getShortDisplayArray() {
                String[] stringArray = { name, section, getTitle(), getMeetingString() };
                return stringArray;
        }

        /**
         * returns an array of length 7 containing the Course name, section, title,
         * credits, instructorId, meeting string, empty string (for a field that Event
         * will have that Course does not).
         */
        @Override
        public String[] getLongDisplayArray() {
                String[] stringArray = { name, section, getTitle(), String.valueOf(credits),
instructorId, getMeetingString(),
                                "" };
                return stringArray;
        }

        /**
         * Sets meeting days and times
         */
        @Override
        public void setMeetingDaysAndTime(String meetingDays, int startTime, int endTime) {
                if (meetingDays == null || meetingDays.length() == 0) {
                        throw new IllegalArgumentException("Invalid meeting days and times.");
                }
                if (meetingDays.contains("A")) {
                        if (meetingDays.length() != 1) {
                                throw new IllegalArgumentException("Invalid meeting days and
times.");
                        }

                        if (startTime != 0 || endTime != 0) {
                                throw new IllegalArgumentException("Invalid meeting days and
times.");
                        }

                        super.setMeetingDaysAndTime(meetingDays, 0, 0);

                }

                else {
                        int m = 0;
                        int t = 0;
                        int w = 0;
                        int h = 0;
                        int f = 0;
                        int a = 0;
```

```
                        for (int i = 0; i < meetingDays.length(); i++) {
                                if (meetingDays.charAt(i) == 'M') {
                                        m++;
                                } else if (meetingDays.charAt(i) == 'T') {
                                        t++;
                                } else if (meetingDays.charAt(i) == 'W') {
                                        w++;
                                } else if (meetingDays.charAt(i) == 'H') {
                                        h++;
                                } else if (meetingDays.charAt(i) == 'F') {
                                        f++;
                                } else if (meetingDays.charAt(i) == 'A') {
                                        a++;
                                } else {
                                        throw new IllegalArgumentException("Invalid meeting days and
times.");
                                }
                        }
                        if (m > 1 || t > 1 || w > 1 || h > 1 || f > 1 || a > 1) {
                                throw new IllegalArgumentException("Invalid meeting days and
times.");
                        }
                }
                int h1 = startTime / 100;
                int h2 = endTime / 100;
                int m1 = startTime % 100;
                int m2 = endTime % 100;
                if (h1 < 0 || h1 > 23)
                        throw new IllegalArgumentException("Invalid meeting days and times.");
                if (h2 < 0 || h2 > 23)
                        throw new IllegalArgumentException("Invalid meeting days and times.");
                if (m1 < 0 || m1 > 59)
                        throw new IllegalArgumentException("Invalid meeting days and times.");
                if (m2 < 0 || m2 > 59)
                        throw new IllegalArgumentException("Invalid meeting days and times.");
                if (startTime > endTime)
                        throw new IllegalArgumentException("Invalid meeting days and times.");

                super.setMeetingDaysAndTime(meetingDays, startTime, endTime);
        }

        /**
         * Figures out if there is a duplicate activity
         *
         * @return true if there is a duplicate and false if not
         */
        @Override
        public boolean isDuplicate(Activity activity) {
                if (activity instanceof Course) {
                        Course c = (Course) activity;
                        if (getName().equals(c.getName())) {
                                return true;
                        }
                }

                return false;
        }

}
```