1. The `APCalendar` class contains methods used to calculate information about a calendar. You will write two methods of the class.

```
public class APCalendar
{

    /** Returns true if year is a leap year and false otherwise. */
    private static boolean isLeapYear(int year)
    {  /* implementation not shown */  }

    /** Returns the number of leap years between year1 and year2, inclusive.
     *  Precondition: 0 <= year1 <= year2
     */
    public static int numberOfLeapYears(int year1, int year2)
    {  /* to be implemented in part (a) */  }

    /** Returns the value representing the day of the week for the first day of year,
     *  where 0 denotes Sunday, 1 denotes Monday, ..., and 6 denotes Saturday.
     */
    private static int firstDayOfYear(int year)
    {  /* implementation not shown */  }

    /** Returns n, where month, day, and year specify the nth day of the year.
     *  Returns 1 for January 1 (month = 1, day = 1) of any year.
     *  Precondition: The date represented by month, day, year is a valid date.
     */
    private static int dayOfYear(int month, int day, int year)
    {  /* implementation not shown */  }

    /** Returns the value representing the day of the week for the given date
     *  (month, day, year), where 0 denotes Sunday, 1 denotes Monday, ...,
     *  and 6 denotes Saturday.
     *  Precondition: The date represented by month, day, year is a valid date.
     */
    public static int dayOfWeek(int month, int day, int year)
    {  /* to be implemented in part (b) */  }

    // There may be instance variables, constructors, and other methods not shown.

}
```

GO ON TO THE NEXT PAGE.

(a) Write the static method `numberOfLeapYears`, which returns the number of leap years between `year1` and `year2`, inclusive.

In order to calculate this value, a helper method is provided for you.

- `isLeapYear(year)` returns `true` if `year` is a leap year and `false` otherwise.

Complete method `numberOfLeapYears` below. You must use `isLeapYear` appropriately to receive full credit.

```
/** Returns the number of leap years between year1 and year2, inclusive.
 *  Precondition: 0 <= year1 <= year2
 */
public static int numberOfLeapYears(int year1, int year2)
```

(b) Write the static method `dayOfWeek`, which returns the integer value representing the day of the week for the given date `(month, day, year)`, where `0` denotes Sunday, `1` denotes Monday, ..., and `6` denotes Saturday. For example, 2019 began on a Tuesday, and January 5 is the fifth day of 2019. As a result, January 5, 2019, fell on a Saturday, and the method call `dayOfWeek(1, 5, 2019)` returns `6`.

As another example, January 10 is the tenth day of 2019. As a result, January 10, 2019, fell on a Thursday, and the method call `dayOfWeek(1, 10, 2019)` returns `4`.

In order to calculate this value, two helper methods are provided for you.

- `firstDayOfYear(year)` returns the integer value representing the day of the week for the first day of `year`, where `0` denotes Sunday, `1` denotes Monday, ..., and `6` denotes Saturday. For example, since 2019 began on a Tuesday, `firstDayOfYear(2019)` returns `2`.
- `dayOfYear(month, day, year)` returns *n*, where `month`, `day`, and `year` specify the *n*th day of the year. For the first day of the year, January 1 `(month = 1, day = 1)`, the value `1` is returned. This method accounts for whether `year` is a leap year. For example, `dayOfYear(3, 1, 2017)` returns `60`, since 2017 is not a leap year, while `dayOfYear(3, 1, 2016)` returns `61`, since 2016 is a leap year.

Class information for this question

```
public class APCalendar

private static boolean isLeapYear(int year)
public static int numberOfLeapYears(int year1, int year2)
private static int firstDayOfYear(int year)
private static int dayOfYear(int month, int day, int year)
public static int dayOfWeek(int month, int day, int year)
```

GO ON TO THE NEXT PAGE.

Complete method `dayOfWeek` below. You must use `firstDayOfYear` and `dayOfYear` appropriately to receive full credit.

```
/** Returns the value representing the day of the week for the given date
 *  (month, day, year), where 0 denotes Sunday, 1 denotes Monday, ...,
 *  and 6 denotes Saturday.
 *  Precondition: The date represented by month, day, year is a valid date.
 */
public static int dayOfWeek(int month, int day, int year)
```

GO ON TO THE NEXT PAGE.

2. This question involves the implementation of a fitness tracking system that is represented by the `StepTracker` class. A `StepTracker` object is created with a parameter that defines the minimum number of steps that must be taken for a day to be considered *active*.

The `StepTracker` class provides a constructor and the following methods.

- `addDailySteps`, which accumulates information about steps, in readings taken once per day
- `activeDays`, which returns the number of active days
- `averageSteps`, which returns the average number of steps per day, calculated by dividing the total number of steps taken by the number of days tracked

The following table contains a sample code execution sequence and the corresponding results.

Statements and Expressions	**Value Returned (blank if no value)**	**Comment**
`StepTracker tr = new StepTracker(10000);`		Days with at least 10,000 steps are considered active. Assume that the parameter is positive.
`tr.activeDays();`	`0`	No data have been recorded yet.
`tr.averageSteps();`	`0.0`	When no step data have been recorded, the `averageSteps` method returns `0.0`.
`tr.addDailySteps(9000);`		This is too few steps for the day to be considered active.
`tr.addDailySteps(5000);`		This is too few steps for the day to be considered active.
`tr.activeDays();`	`0`	No day had at least 10,000 steps.
`tr.averageSteps();`	`7000.0`	The average number of steps per day is (14000 / 2).
`tr.addDailySteps(13000);`		This represents an active day.
`tr.activeDays();`	`1`	Of the three days for which step data were entered, one day had at least 10,000 steps.
`tr.averageSteps();`	`9000.0`	The average number of steps per day is (27000 / 3).
`tr.addDailySteps(23000);`		This represents an active day.
`tr.addDailySteps(1111);`		This is too few steps for the day to be considered active.
`tr.activeDays();`	`2`	Of the five days for which step data were entered, two days had at least 10,000 steps.
`tr.averageSteps();`	`10222.2`	The average number of steps per day is (51111 / 5).

In this question, you will write two methods in the following `Delimiters` class.

```
public class Delimiters
{

   /** The open and close delimiters. */
   private String openDel;
   private String closeDel;

   /** Constructs a Delimiters object where open is the open delimiter and close is the
    *  close delimiter.
    *  Precondition: open and close are non-empty strings.
    */
   public Delimiters(String open, String close)
   {
      openDel = open;
      closeDel = close;
   }

   /** Returns an ArrayList of delimiters from the array tokens, as described in part (a). */
   public ArrayList<String> getDelimitersList(String[] tokens)
   {  /* to be implemented in part (a) */  }

   /** Returns true if the delimiters are balanced and false otherwise, as described in part (b).
    *  Precondition: delimiters contains only valid open and close delimiters.
    */
   public boolean isBalanced(ArrayList<String> delimiters)
   {  /* to be implemented in part (b) */  }

   // There may be instance variables, constructors, and methods that are not shown.

}
```

(a) A string containing text and possibly delimiters has been split into *tokens* and stored in `String[] tokens`. Each token is either an open delimiter, a close delimiter, or a substring that is not a delimiter. You will write the method `getDelimitersList`, which returns an `ArrayList` containing all the open and close delimiters found in `tokens` in their original order.

The following examples show the contents of an `ArrayList` returned by `getDelimitersList` for different open and close delimiters and different `tokens` arrays.

Example 1

openDel: `"("`

closeDel: `")"`

tokens:	"("	"x + y"	")"	" * 5"

ArrayList of delimiters:	"("	")"

Example 2

openDel: `"<q>"`

closeDel: `"</q>"`

tokens:	"<q>"	"yy"	"</q>"	"zz"	"</q>"

ArrayList of delimiters:	"<q>"	"</q>"	"</q>"

Class information for this question

```
public class Delimiters

private String openDel
private String closeDel

public Delimiters(String open, String close)
public ArrayList<String> getDelimitersList(String[] tokens)
public boolean isBalanced(ArrayList<String> delimiters)
```

AP® COMPUTER SCIENCE A
2019 SCORING GUIDELINES

Question 1: Calendar

Part (a)	`numberOfLeapYears`	5 points

Intent: *Return the number of leap years in a range*

+1 Initializes a numeric variable

+1 Loops through each necessary year in the range

+1 Calls `isLeapYear` on some valid year in the range

+1 Updates count based on result of calling `isLeapYear`

+1 Returns count of leap years

Part (b)	`dayOfWeek`	4 points

Intent: *Return an integer representing the day of the week for a given date*

+1 Calls `firstDayOfYear`

+1 Calls `dayOfYear`

+1 Calculates the value representing the day of the week

+1 Returns the calculated value

Question-Specific Penalties

-1 (t) Static methods called with `this`.

Question 1: Scoring Notes

Part (a) `numberOfLeapYears`			**5 points**
Points	Rubric Criteria	Responses earn the point even if they…	Responses will not earn the point if they…
+1	Initializes a numeric variable		• use the variable for loop control only
+1	Loops through each necessary year in the range		• consider years outside the range
+1	Calls `isLeapYear` on some valid year in the range	• do not use a loop	
+1	Updates count based on result of calling `isLeapYear`	• do not use a loop • do not initialize the counter	• use result as a non-`boolean`
+1	Returns count of leap years	• loop from `year1` to `year2` incorrectly • do not initialize the counter	• do not use a loop • update or initialize the counter incorrectly • return early inside the loop
Part (b) `dayOfWeek`			**4 points**
Points	Rubric Criteria	Responses earn the point even if they…	Responses will not earn the point if they…
+1	Calls `firstDayOfYear`		• do not use the given year
+1	Calls `dayOfYear`		• have arguments out of order
+1	Calculates the value representing the day of the week		• make any error in the calculation
+1	Returns the calculated value	• return the value from calling `firstDayOfYear` or `dayOfYear`	• return a constant value

Question 1: Calendar

Part (a)

```
public static int numberOfLeapYears(int year1, int year2)
{
   int count = 0;
   for (int y = year1; y <= year2; y++)
   {
      if (isLeapYear(y))
      {
         count++;
      }
   }
   return count;
}
```

Part (b)

```
public static int dayOfWeek(int month, int day, int year)
{
   int startDay = firstDayOfYear(year);
   int nthDay = dayOfYear(month, day, year);
   int returnDay = (startDay + nthDay - 1) % 7;
   return returnDay;
}
```

These canonical solutions serve an expository role, depicting general approaches to solution. Each reflects only one instance from the infinite set of valid solutions. The solutions are presented in a coding style chosen to enhance readability and facilitate understanding.