

1. This question involves dog walkers who are paid through a dog-walking company to take dogs for one-hour walks. A dog-walking company has a varying number of dogs that need to be walked during each hour of the day. A dog-walking company is represented by the following `DogWalkCompany` class.

```
public class DogWalkCompany
{
    /**
     * Returns the number of dogs, always greater than 0, that are available
     * for a walk during the time specified by hour
     * Precondition: 0 <= hour <= 23
     */
    public int numAvailableDogs(int hour)
    { /* implementation not shown */ }

    /**
     * Decreases, by numberDogsWalked, the number of dogs available for a walk
     * during the time specified by hour
     * Preconditions: 0 <= hour <= 23
     *                 numberDogsWalked > 0
     */
    public void updateDogs(int hour, int numberDogsWalked)
    { /* implementation not shown */ }

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

A dog walker is associated with a dog-walking company and is represented by the `DogWalker` class. You will write two methods of the `DogWalker` class.

```
public class DogWalker
{
    /** The maximum number of dogs this walker can walk simultaneously
        per hour */
    private int maxDogs;

    /** The dog-walking company this dog walker is associated with */
    private DogWalkCompany company;

    /**
     * Assigns max to maxDogs and comp to company
     * Precondition: max > 0
     */
    public DogWalker(int max, DogWalkCompany comp)
    { /* implementation not shown */ }

    /**
     * Takes at least one dog for a walk during the time specified by
     * hour, as described in part (a)
     * Preconditions: 0 <= hour <= 23
     *                 maxDogs > 0
     */
    public int walkDogs(int hour)
    { /* to be implemented in part (a) */ }
```

```
/**
 * Performs an entire dog-walking shift and returns the amount
 * earned, in dollars, as described in part (b)
 * Preconditions: 0 <= startHour <= endHour <= 23
 *                maxDogs > 0
 */
public int dogWalkShift(int startHour, int endHour)
{ /* to be implemented in part (b) */ }

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

- A. Write the `walkDogs` method, which updates and returns the number of dogs this dog walker walks during the time specified by `hour`. Values of `hour` range from 0 to 23, inclusive.

A helper method, `numAvailableDogs`, has been provided in the `DogWalkCompany` class. The method returns the number of dogs available to be taken for a walk at a given hour. The dog walker will always walk as many dogs as the dog-walking company has available to walk, as long as the available number of dogs is not greater than the maximum number of dogs that the dog walker can handle, represented by the value of `maxDogs`.

Another helper method, `updateDogs`, has also been provided in the `DogWalkCompany` class. So that multiple dog walkers do not sign up to walk the same dogs, the `walkDogs` method should use `updateDogs` to update the dog-walking company with the number of dogs this dog walker will walk during the given hour. The parameters of the `updateDogs` method indicate how many dogs this dog walker will walk during the time specified by `hour`.

For example, if the dog-walking company has 10 dogs that need to be walked at the given hour but the dog walker's maximum is 4, the `updateDogs` method should be used to indicate that this dog walker will walk 4 of the 10 dogs during the given hour. As another example, if the dog-walking company has 3 dogs that need to be walked at the given hour and the dog walker's maximum is 4, the `updateDogs` method should be used to indicate that this dog walker will walk all 3 of the available dogs during the given hour.

The `walkDogs` method should return the number of dogs to be walked by this dog walker during the time specified by `hour`.

Complete method `walkDogs`. You must use `numAvailableDogs` and `updateDogs` appropriately to receive full credit.

```
/**
 * Takes at least one dog for a walk during the time specified by
 * hour, as described in part (a)
 * Preconditions: 0 <= hour <= 23
 *                maxDogs > 0
 */
public int walkDogs(int hour)
```

- B. Write the `dogWalkShift` method, which performs a dog-walking shift of the hours in the range `startHour` to `endHour`, inclusive, and returns the total amount earned. For example, a dog-walking shift from 14 to 16 is composed of three one-hour dog walks starting at hours 14, 15, and 16.

For each hour, the base pay is \$5 per dog walked plus a bonus of \$3 if at least one of the following is true.

- `maxDogs` dogs are walked
- the walk occurs between the peak hours of 9 and 17, inclusive

The following table shows an example of the calculated amount earned by walking dogs from hour 7 through hour 10, inclusive.

Hour	Maximum Number of Dogs	Number of Dogs Walked	Amount Earned, in Dollars
7	3	3	$3 \times 5 + 3 = 18$
8	3	2	$2 \times 5 = 10$
9	3	2	$2 \times 5 + 3 = 13$
10	3	3	$3 \times 5 + 3 = 18$
Total			59

Complete method `dogWalkShift`. Assume that `walkDogs` works as specified, regardless of what you wrote in part (a). You must use `walkDogs` appropriately to earn full credit.

```
/**
 * Performs an entire dog-walking shift and returns the amount
 * earned, in dollars, as described in part (b)
 * Preconditions: 0 <= startHour <= endHour <= 23
 *               maxDogs > 0
 */
public int dogWalkShift(int startHour, int endHour)
```

1. This question involves simulation of the play and scoring of a single-player video game. In the game, a player attempts to complete three levels. A level in the game is represented by the `Level` class.

```
public class Level
{
    /** Returns true if the player reached the goal on this level and returns false otherwise */
    public boolean goalReached()
    { /* implementation not shown */ }

    /** Returns the number of points (a positive integer) recorded for this level */
    public int getPoints()
    { /* implementation not shown */ }

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

GO ON TO THE NEXT PAGE.

Play of the game is represented by the `Game` class. You will write two methods of the `Game` class.

```
public class Game
{
    private Level levelOne;
    private Level levelTwo;
    private Level levelThree;

    /** Postcondition: All instance variables have been initialized. */
    public Game()
    { /* implementation not shown */ }

    /** Returns true if this game is a bonus game and returns false otherwise */
    public boolean isBonus()
    { /* implementation not shown */ }

    /** Simulates the play of this Game (consisting of three levels) and updates all relevant
     * game data
     */
    public void play()
    { /* implementation not shown */ }

    /** Returns the score earned in the most recently played game, as described in part (a) */
    public int getScore()
    { /* to be implemented in part (a) */ }

    /** Simulates the play of num games and returns the highest score earned, as
     * described in part (b)
     * Precondition: num > 0
     */
    public int playManyTimes(int num)
    { /* to be implemented in part (b) */ }

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

GO ON TO THE NEXT PAGE.

- (a) Write the `getScore` method, which returns the score for the most recently played game. Each game consists of three levels. The score for the game is computed using the following helper methods.
- The `isBonus` method of the `Game` class returns `true` if this is a bonus game and returns `false` otherwise.
 - The `goalReached` method of the `Level` class returns `true` if the goal has been reached on a particular level and returns `false` otherwise.
 - The `getPoints` method of the `Level` class returns the number of points recorded on a particular level. Whether or not recorded points are earned (included in the game score) depends on the rules of the game, which follow.

The score for the game is computed according to the following rules.

- Level one points are earned only if the level one goal is reached. Level two points are earned only if both the level one and level two goals are reached. Level three points are earned only if the goals of all three levels are reached.
- The score for the game is the sum of the points earned for levels one, two, and three.
- If the game is a bonus game, the score for the game is tripled.

GO ON TO THE NEXT PAGE.

The following table shows some examples of game score calculations.

	Level One Results	Level Two Results	Level Three Results	isBonus Return Value	Score Calculation
goalReached Return Value: getPoints Return Value:	true 200	true 100	true 500	true	$(200 + 100 + 500) \times 3 = 2,400$ The recorded points for levels one, two, and three are earned because the goals were reached in all three levels. The earned points are multiplied by 3 because isBonus returns true.
goalReached Return Value: getPoints Return Value:	true 200	true 100	false 500	false	$200 + 100 = 300$ The recorded points for level one and level two are earned because the goal was reached in levels one and two. The recorded points for level three are not earned because the goal was not reached in level three.
goalReached Return Value: getPoints Return Value:	true 200	false 100	true 500	true	$200 \times 3 = 600$ The recorded points for only level one are earned because the goal was not reached in level two. The earned points are multiplied by 3 because isBonus returns true.
goalReached Return Value: getPoints Return Value:	false 200	true 100	true 500	false	0 Because the goal in level one was not reached, no points are earned for any level.

Complete the getScore method.

```
/** Returns the score earned in the most recently played game, as described in part (a) */
public int getScore()
```

Begin your response at the top of a new page in the Free Response booklet and fill in the appropriate circle indicating the question number.
If there are multiple parts to this question, write the part letter with your response.

GO ON TO THE NEXT PAGE.

- (b) Write the `playManyTimes` method, which simulates the play of `num` games and returns the highest game score earned. For example, if the four plays of the game that are simulated as a result of the method call `playManyTimes(4)` earn scores of 75, 50, 90, and 20, then the method should return 90.

Play of the game is simulated by calling the helper method `play`. Note that if `play` is called only one time followed by multiple consecutive calls to `getScore`, each call to `getScore` will return the score earned in the single simulated play of the game.

Complete the `playManyTimes` method. Assume that `getScore` works as intended, regardless of what you wrote in part (a). You must call `play` and `getScore` appropriately in order to receive full credit.

```
/** Simulates the play of num games and returns the highest score earned, as
 * described in part (b)
 * Precondition: num > 0
 */
public int playManyTimes(int num)
```

Begin your response at the top of a new page in the Free Response booklet and fill in the appropriate circle indicating the question number.
If there are multiple parts to this question, write the part letter with your response.

Class information for this question

```
public class Level
{
    public boolean goalReached()
    public int getPoints()
}

public class Game
{
    private Level levelOne
    private Level levelTwo
    private Level levelThree

    public Game()
    public boolean isBonus()
    public void play()
    public int getScore()
    public int playManyTimes(int num)
```

GO ON TO THE NEXT PAGE.

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.
}
```

- (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)
```

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)
```

1. This question involves reasoning about a simulation of a frog hopping in a straight line. The frog attempts to hop to a goal within a specified number of hops. The simulation is encapsulated in the following `FrogSimulation` class. You will write two of the methods in this class.

```
public class FrogSimulation
{
    /** Distance, in inches, from the starting position to the goal. */
    private int goalDistance;

    /** Maximum number of hops allowed to reach the goal. */
    private int maxHops;

    /** Constructs a FrogSimulation where dist is the distance, in inches, from the starting
     * position to the goal, and numHops is the maximum number of hops allowed to reach the goal.
     * Precondition: dist > 0; numHops > 0
     */
    public FrogSimulation(int dist, int numHops)
    {
        goalDistance = dist;
        maxHops = numHops;
    }

    /** Returns an integer representing the distance, in inches, to be moved when the frog hops.
     */
    private int hopDistance()
    {
        /* implementation not shown */
    }

    /** Simulates a frog attempting to reach the goal as described in part (a).
     * Returns true if the frog successfully reached or passed the goal during the simulation;
     * false otherwise.
     */
    public boolean simulate()
    {
        /* to be implemented in part (a) */
    }

    /** Runs num simulations and returns the proportion of simulations in which the frog
     * successfully reached or passed the goal.
     * Precondition: num > 0
     */
    public double runSimulations(int num)
    {
        /* to be implemented in part (b) */
    }
}
```

- (a) Write the `simulate` method, which simulates the frog attempting to hop in a straight line to a goal from the frog's starting position of 0 within a maximum number of hops. The method returns `true` if the frog successfully reached the goal within the maximum number of hops; otherwise, the method returns `false`.

The `FrogSimulation` class provides a method called `hopDistance` that returns an integer representing the distance (positive or negative) to be moved when the frog hops. A positive distance represents a move toward the goal. A negative distance represents a move away from the goal. The returned distance may vary from call to call. Each time the frog hops, its position is adjusted by the value returned by a call to the `hopDistance` method.

The frog hops until one of the following conditions becomes true:

- The frog has reached or passed the goal.
- The frog has reached a negative position.
- The frog has taken the maximum number of hops without reaching the goal.

The following example shows a declaration of a `FrogSimulation` object for which the goal distance is 24 inches and the maximum number of hops is 5. The table shows some possible outcomes of calling the `simulate` method.

```
FrogSimulation sim = new FrogSimulation(24, 5);
```

	Values returned by <code>hopDistance()</code>	Final position of frog	Return value of <code>sim.simulate()</code>
Example 1	5, 7, -2, 8, 6	24	<code>true</code>
Example 2	6, 7, 6, 6	25	<code>true</code>
Example 3	6, -6, 31	31	<code>true</code>
Example 4	4, 2, -8	-2	<code>false</code>
Example 5	5, 4, 2, 4, 3	18	<code>false</code>

Class information for this question

```
public class FrogSimulation
{
    private int goalDistance
    private int maxHops

    private int hopDistance()
    public boolean simulate()
    public double runSimulations(int num)
}
```

Complete method `simulate` below. You must use `hopDistance` appropriately to receive full credit.

```
/** Simulates a frog attempting to reach the goal as described in part (a).
 * Returns true if the frog successfully reached or passed the goal during the simulation.
```

- (b) Write the `runSimulations` method, which performs a given number of simulations and returns the proportion of simulations in which the frog successfully reached or passed the goal. For example, if the parameter passed to `runSimulations` is 400, and 100 of the 400 `simulate` method calls returned `true`, then the `runSimulations` method should return 0.25.

Complete method `runSimulations` below. Assume that `simulate` works as specified, regardless of what you wrote in part (a). You must use `simulate` appropriately to receive full credit.

```
/** Runs num simulations and returns the proportion of simulations in which the frog
 * successfully reached or passed the goal.
 * @param num the number of simulations to run
 * @return the proportion of simulations in which the frog successfully reached or passed the goal
```