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 × 5 + 3 = 18
8	3	2	2 × 5 = 10
9	3	2	2 × 5 + 3 = 13
10	3	3	3 × 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)
```


2. This question involves the `SignedText` class, which contains methods that are used to include a signature as part of a string of text. You will write the complete `SignedText` class, which contains a constructor and two methods.

The `SignedText` constructor takes two `String` parameters. The first parameter is a first name and the second parameter is a last name. The length of the second parameter is always greater than or equal to 1.

The `getSignature` method takes no parameters and returns a formatted signature string constructed from the first and last names according to the following rules.

- If the first name is an empty string, the returned signature string contains just the last name.
- If the first name is not an empty string, the returned signature string is the first letter of the first name, a dash (`"-"`), and the last name, in that order.

The `addSignature` method returns a possibly revised copy of its `String` parameter. The parameter will contain at most one occurrence of the object's signature, at either the beginning or the end of the parameter. The returned string is created from the parameter according to the following rules.

- If the object's signature does not occur in the `String` parameter of the method, the returned `String` is the value of the parameter with the signature added to the end.
- If the object's signature occurs at the end of the `String` parameter, the returned `String` is the unchanged value of the parameter.
- If the object's signature occurs at the beginning of the `String` parameter, the returned `String` is the value of the original parameter with the signature removed from the beginning and appended to the end of the parameter.

The following table contains a sample code execution sequence and the corresponding results. The code execution sequence appears in a class other than `SignedText`.

Statement	Method Call Return Value (blank if none)	Explanation
`SignedText st1 = new SignedText("", "Wong");`		The `SignedText` object `st1` has an empty first name and last name `"Wong"`.
`String temp = st1.getSignature();`	`"Wong"`	
`SignedText st2 = new SignedText("henri", "dubois");`		The `SignedText` object `st2` has first name `"henri"` and last name `"dubois"`.
`temp = st2.getSignature();`	`"h-dubois"`	
`SignedText st3 = new SignedText("GRACE", "LOPEZ");`		The `SignedText` object `st3` has first name `"GRACE"` and last name `"LOPEZ"`.
`temp = st3.getSignature();`	`"G-LOPEZ"`	
`SignedText st4 = new SignedText("", "FOX");`		The `SignedText` object `st4` has an empty first name and last name `"FOX"`.
`String text = "Dear";`		
`temp = st4.addSignature(text);`	`"DearFOX"`	The signature does not occur in the `addSignature` parameter, so the returned string is the value of the parameter with the signature appended.
`text = "Best wishesFOX";`		
`temp = st4.addSignature(text);`	`"Best wishesFOX"`	The signature occurs at the end of the `addSignature` parameter, so the returned string is the unchanged value of the parameter.

Statement	Method Call Return Value (blank if none)	Explanation
`text = "FOXThanks";`		
`temp = st4.addSignature(text);`	`"ThanksFOX"`	The signature occurs at the beginning of the `addSignature` parameter, so the returned string is the value of the original parameter with the signature removed from the beginning and appended to the end of the parameter.
`text = "G-LOPEZHello";`		
`temp = st3.addSignature(text);`	`"HelloG-LOPEZ"`	The signature occurs at the beginning of the `addSignature` parameter, so the returned string is the value of the original parameter with the signature removed from the beginning and appended to the end of the parameter.

Write the complete `SignedText` class. Your implementation must meet all specifications and conform to the examples in the table.

4. This question involves the process of taking a list of words, called `wordList`, and producing a formatted string of a specified length. The list `wordList` contains at least two words, consisting of letters only.

   When the formatted string is constructed, spaces are placed in the gaps between words so that as many spaces as possible are evenly distributed to each gap. The equal number of spaces inserted into each gap is referred to as the *basic gap width.* Any *leftover spaces* are inserted one at a time into the gaps from left to right until there are no more leftover spaces.

   The following three examples illustrate these concepts. In each example, the list of words is to be placed into a formatted string of length 20.

   Example 1: `wordList: ["AP", "COMP", "SCI", "ROCKS"]`

   Total number of letters in words: 14
   Number of gaps between words: 3
   Basic gap width: 2
   Leftover spaces: 0

   Formatted string:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
A	P			C	O	M	P			S	C	I			R	O	C	K	S

   Example 2: `wordList: ["GREEN", "EGGS", "AND", "HAM"]`

   Total number of letters in words: 15
   Number of gaps between words: 3
   Basic gap width: 1
   Leftover spaces: 2

   The leftover spaces are inserted one at a time between the words from left to right until there are no more leftover spaces. In this example, the first two gaps get an extra space.

   Formatted string:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
G	R	E	E	N			E	G	G	S			A	N	D		H	A	M

   Example 3: `wordList: ["BEACH", "BALL"]`

   Total number of letters in words: 9
   Number of gaps between words: 1
   Basic gap width: 11
   Leftover spaces: 0

   Formatted string:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
B	E	A	C	H												B	A	L	L

   You will implement three `static` methods in a class named `StringFormatter` that is not shown.

(a) Write the `StringFormatter` method `totalLetters`, which returns the total number of letters in the words in its parameter `wordList`. For example, if the variable `List<String> words` is `["A", "frog", "is"]`, then the call `StringFormatter.totalLetters(words)` returns 7. You may assume that all words in `wordList` consist of one or more letters.

Complete method `totalLetters` below.

```
/** Returns the total number of letters in wordList.
 *  Precondition: wordList contains at least two words, consisting of letters only.
 */
public static int totalLetters(List<String> wordList)
```

Part (b) begins on page 20.

(b) Write the `StringFormatter` method `basicGapWidth`, which returns the basic gap width as defined earlier.

Class information for this question

```
public class StringFormatter

public static int totalLetters(List<String> wordList)
public static int basicGapWidth(List<String> wordList,
                                int formattedLen)
public static int leftoverSpaces(List<String> wordList,
                                 int formattedLen)
public static String format(List<String> wordList, int formattedLen)
```

WRITE YOUR SOLUTION ON THE NEXT PAGE.

Assume that `totalLetters` works as specified regardless of what you wrote in part (a). You must use `totalLetters` appropriately to receive full credit.

Complete method `basicGapWidth` below.

```
/** Returns the basic gap width when wordList is used to produce
 *  a formatted string of formattedLen characters.
 *  Precondition: wordList contains at least two words, consisting of letters only.
 *                formattedLen is large enough for all the words and gaps.
 */
public static int basicGapWidth(List<String> wordList,
                                int formattedLen)
```

Part (c) begins on page 22.

(c) Write the `StringFormatter` method `format`, which returns the formatted string as defined earlier. The `StringFormatter` class also contains a method called `leftoverSpaces`, which has already been implemented. This method returns the number of leftover spaces as defined earlier and is shown below.

```
/** Returns the number of leftover spaces when wordList is used to produce
 *  a formatted string of formattedLen characters.
 *  Precondition: wordList contains at least two words, consisting of letters only.
 *                formattedLen is large enough for all the words and gaps.
 */
public static int leftoverSpaces(List<String> wordList,
                                 int formattedLen)
{  /* implementation not shown */  }
```

Class information for this question

```
public class StringFormatter

public static int totalLetters(List<String> wordList)
public static int basicGapWidth(List<String> wordList,
                                int formattedLen)
public static int leftoverSpaces(List<String> wordList,
                                 int formattedLen)
public static String format(List<String> wordList, int formattedLen)
```

WRITE YOUR SOLUTION ON THE NEXT PAGE.

Assume that `basicGapWidth` works as specified, regardless of what you wrote in part (b). You must use `basicGapWidth` and `leftoverSpaces` appropriately to receive full credit.

Complete method `format` below.

```
/** Returns a formatted string consisting of the words in wordList separated by spaces.
 *  Precondition: The wordList contains at least two words, consisting of letters only.
 *                formattedLen is large enough for all the words and gaps.
 *  Postcondition: All words in wordList appear in the formatted string.
 *    - The words appear in the same order as in wordList.
 *    - The number of spaces between words is determined by basicGapWidth and the
 *      distribution of leftoverSpaces from left to right, as described in the question.
 */
public static String format(List<String> wordList, int formattedLen)
```

STOP

END OF EXAM