1. Consider a software system that models a horse barn. Classes that represent horses implement the following interface.

```
public interface Horse
{
  /** @return the horse's name */
  String getName();

  /** @return the horse's weight */
  int getWeight();

  // There may be methods that are not shown.
}
```

A horse barn consists of N numbered spaces. Each space can hold at most one horse. The spaces are indexed starting from 0; the index of the last space is $N - 1$. No two horses in the barn have the same name.

The declaration of the `HorseBarn` class is shown below. You will write two unrelated methods of the `HorseBarn` class.

```
public class HorseBarn
{
  /** The spaces in the barn. Each array element holds a reference to the horse
   *  that is currently occupying the space. A null value indicates an empty space.
   */
  private Horse[] spaces;


  /** Returns the index of the space that contains the horse with the specified name.
   *  Precondition: No two horses in the barn have the same name.
   *  @param name the name of the horse to find
   *  @return the index of the space containing the horse with the specified name;
   *          -1 if no horse with the specified name is in the barn.
   */
  public int findHorseSpace(String name)
  {  /* to be implemented in part (a) */  }


  /** Consolidates the barn by moving horses so that the horses are in adjacent spaces,
   *  starting at index 0, with no empty space between any two horses.
   *  Postcondition: The order of the horses is the same as before the consolidation.
   */
  public void consolidate()
  {  /* to be implemented in part (b) */  }

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

(a) Write the `HorseBarn` method `findHorseSpace`. This method returns the index of the space in which the horse with the specified name is located. If there is no horse with the specified name in the barn, the method returns -1.

For example, assume a `HorseBarn` object called `sweetHome` has horses in the following spaces.

0	1	2	3	4	5	6
"Trigger" 1340	null	"Silver" 1210	"Lady" 1575	null	"Patches" 1350	"Duke" 1410

The following table shows the results of several calls to the `findHorseSpace` method.

Method Call	Value Returned	Reason
`sweetHome.findHorseSpace("Trigger")`	0	A horse named Trigger is in space 0.
`sweetHome.findHorseSpace("Silver")`	2	A horse named Silver is in space 2.
`sweetHome.findHorseSpace("Coco")`	-1	A horse named Coco is not in the barn.

Information repeated from the beginning of the question

```
public interface Horse

String getName()
int getWeight()

public class HorseBarn

private Horse[] spaces
public int findHorseSpace(String name)
public void consolidate()
```

WRITE YOUR SOLUTION ON THE NEXT PAGE.

Complete method `findHorseSpace` below.

```
/** Returns the index of the space that contains the horse with the specified name.
 *  Precondition: No two horses in the barn have the same name.
 *  @param name the name of the horse to find
 *  @return the index of the space containing the horse with the specified name;
 *          -1 if no horse with the specified name is in the barn.
 */
public int findHorseSpace(String name)
```

(b) Write the `HorseBarn` method `consolidate`. This method consolidates the barn by moving horses so that the horses are in adjacent spaces, starting at index 0, with no empty spaces between any two horses. After the barn is consolidated, the horses are in the same order as they were before the consolidation.

For example, assume a barn has horses in the following spaces.

0	1	2	3	4	5	6
"Trigger" 1340	null	"Silver" 1210	null	null	"Patches" 1350	"Duke" 1410

The following table shows the arrangement of the horses after `consolidate` is called.

0	1	2	3	4	5	6
"Trigger" 1340	"Silver" 1210	"Patches" 1350	"Duke" 1410	null	null	null

Information repeated from the beginning of the question

```
public interface Horse

String getName()
int getWeight()

public class HorseBarn

private Horse[] spaces
public int findHorseSpace(String name)
public void consolidate()
```

WRITE YOUR SOLUTION ON THE NEXT PAGE.

Complete method `consolidate` below.

```
/** Consolidates the barn by moving horses so that the horses are in adjacent spaces,
 *  starting at index 0, with no empty space between any two horses.
 *  Postcondition: The order of the horses is the same as before the consolidation.
 */
public void consolidate()
```

2. A multiplayer game called Token Pass has the following rules.

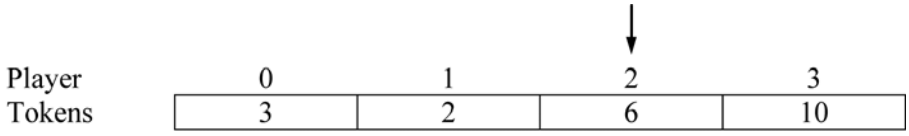
Each player begins with a random number of tokens (at least 1, but no more than 10) that are placed on a linear game board. There is one position on the game board for each player. After the game board has been filled, a player is randomly chosen to begin the game. Each position on the board is numbered, starting with 0.

The following rules apply for a player's turn.

- The tokens are collected and removed from the game board at that player's position.
- The collected tokens are distributed one at a time, to each player, beginning with the next player in order of increasing position.
- If there are still tokens to distribute after the player at the highest position gets a token, the next token will be distributed to the player at position 0.
- The distribution of tokens continues until there are no more tokens to distribute.

The Token Pass game board is represented by an array of integers. The indexes of the array represent the player positions on the game board, and the corresponding values in the array represent the number of tokens that each player has. The following example illustrates one player's turn.

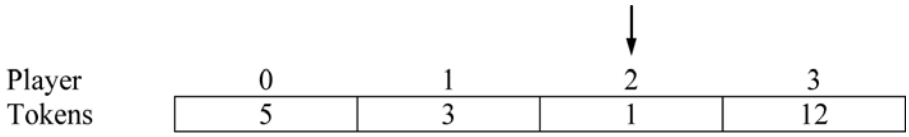
Example
The following represents a game with 4 players. The player at position 2 was chosen to go first.



The tokens at position 2 are collected and distributed as follows.

1st token - to position 3 (The highest position is reached, so the next token goes to position 0.)
2nd token - to position 0
3rd token - to position 1
4th token - to position 2
5th token - to position 3 (The highest position is reached, so the next token goes to position 0.)
6th token - to position 0

After player 2's turn, the values in the array will be as follows.



The Token Pass game is represented by the `TokenPass` class.

```
public class TokenPass
{
  private int[] board;
  private int currentPlayer;


  /** Creates the board array to be of size playerCount and fills it with
   *  random integer values from 1 to 10, inclusive. Initializes currentPlayer to a
   *  random integer value in the range between 0 and playerCount-1, inclusive.
   *  @param playerCount the number of players
   */
  public TokenPass(int playerCount)
  {  /* to be implemented in part (a) */  }


  /** Distributes the tokens from the current player's position one at a time to each player in
   *  the game. Distribution begins with the next position and continues until all the tokens
   *  have been distributed. If there are still tokens to distribute when the player at the
   *  highest position is reached, the next token will be distributed to the player at position 0.
   *  Precondition: the current player has at least one token.
   *  Postcondition: the current player has not changed.
   */
  public void distributeCurrentPlayerTokens()
  {  /* to be implemented in part (b) */  }

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

(a) Write the constructor for the `TokenPass` class. The parameter `playerCount` represents the number of players in the game. The constructor should create the `board` array to contain `playerCount` elements and fill the array with random numbers between 1 and 10, inclusive. The constructor should also initialize the instance variable `currentPlayer` to a random number between 0 and `playerCount-1`, inclusive.

Complete the `TokenPass` constructor below.

```
/** Creates the board array to be of size playerCount and fills it with
 *  random integer values from 1 to 10, inclusive. Initializes currentPlayer to a
 *  random integer value in the range between 0 and playerCount-1, inclusive.
 *  @param playerCount the number of players
 */
public TokenPass(int playerCount)
```

Part (b) begins on page 11.

(b) Write the `distributeCurrentPlayerTokens` method.

The tokens are collected and removed from the game board at the current player's position. These tokens are distributed, one at a time, to each player, beginning with the next higher position, until there are no more tokens to distribute.

Class information repeated from the beginning of the question

```
public class TokenPass

private int[] board
private int currentPlayer
public TokenPass(int playerCount)
public void distributeCurrentPlayerTokens()
```

Complete method `distributeCurrentPlayerTokens` below.

```
/** Distributes the tokens from the current player's position one at a time to each player in
 *  the game. Distribution begins with the next position and continues until all the tokens
 *  have been distributed. If there are still tokens to distribute when the player at the
 *  highest position is reached, the next token will be distributed to the player at position 0.
 *  Precondition: the current player has at least one token.
 *  Postcondition: the current player has not changed.
 */
public void distributeCurrentPlayerTokens()
```