3. 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.
}
```

Part (a) begins on page 14.

(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.

GO ON TO 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)
```

Part (b) begins on page 16.

(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()
```

Question 3: Horse Barn

Part (a)	`findHorseSpace`	4 points

Intent: *Return index of space containing horse with specified name*

+1 Accesses all entries in `spaces` (*no bounds errors*)

+1 Checks for `null` reference in array and avoids dereferencing it (*in context of loop*)

+1 Checks for name equality between array element and parameter
(must use `String` *equality check*)

+1 Returns correct index, if present; -1 point if not

Part (b)	`consolidate`	5 points

Intent: *Repopulate* `spaces` *such that the order of all non-*`null` *entries is preserved and all* `null` *entries are found contiguously at the largest indices*

+1 Accesses all entries in `spaces` (*no bounds errors*)

+1 Identifies and provides different treatment of `null` and non-`null` elements in array

+1 Assigns element in array to a smaller index
(*must have identified source as non-*`null` *or destination as* `null`)

+1 On exit: The number, integrity, and order of all identified non-`null` elements in `spaces` is preserved, and the number of `null` elements is preserved

+1 On exit: All non-`null` elements in `spaces` are in contiguous locations, beginning at index 0 *(no destruction of data*)

Question-Specific Penalties

-1 (z) Attempts to return a value from `consolidate`

-2 (v) Consistently uses incorrect array name instead of `spaces`

AP® COMPUTER SCIENCE A
2012 CANONICAL SOLUTIONS

Question 3: Horse Barn

Part (a):

```
public int findHorseSpace(String name) {
   for (int i = 0; i < this.spaces.length; i++) {
      if (this.spaces[i]!=null && name.equals(this.spaces[i].getName())) {
         return i;
      }
   }
   return -1;
}
```

Part (b):

```
public void consolidate() {
   for (int i = 0; i < this.spaces.length-1; i++) {
      if (this.spaces[i] == null) {
         for (int j = i+1; j < this.spaces.length; j++) {
            if (this.spaces[j] != null) {
               this.spaces[i] = this.spaces[j];
               this.spaces[j] = null;
               j = this.spaces.length;
            }
         }
      }
   }
}
```

Part (b): Alternative solution (auxiliary with array)

```
public void consolidate() {
   Horse[] newSpaces = new Horse[this.spaces.length];
   int nextSpot = 0;
   for (Horse nextHorse : this.spaces) {
      if (nextHorse != null) {
         newSpaces[nextSpot] = nextHorse;
         nextSpot++;
      }
   }
   this.spaces = newSpaces;
}
```

Part (b): Alternative solution (auxiliary with `ArrayList`)

```
public void consolidate() {
   List<Horse> horseList = new ArrayList<Horse>();
   for (Horse h : this.spaces) {
      if (h != null) horseList.add(h);
   }
   for (int i = 0; i < this.spaces.length; i++) {
      this.spaces[i] = null;
   }
   for (int i = 0; i < horseList.size(); i++) {
      this.spaces[i] = horseList.get(i);
   }
}
```

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.