AP® COMPUTER SCIENCE A
2012 SCORING GUIDELINES

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.