AP® COMPUTER SCIENCE A
2012 GENERAL SCORING GUIDELINES

Apply the question-specific rubric first, which always takes precedence. Penalty points can only be deducted in a part of the question that has earned credit via the question-specific rubric. No part of a question — (a), (b), or (c) — may have a negative point total. A given penalty can be assessed only once for a question, even if it occurs multiple times or in different parts of that question.

1-Point Penalty

(w) Extraneous code that causes a side effect or prevents earning points in the rubric (*e.g., information written to output*)

(x) Local variables used but none declared

(y) Destruction of persistent data *(e.g., changing value referenced by parameter)*

(z) `Void` method or constructor that returns a value

No Penalty

- Extraneous code that causes no side effect
- Extraneous code that is unreachable and would not have earned points in rubric
- Spelling/case discrepancies where there is no ambiguity*
- Local variable not declared, provided that other variables are declared in some part
- `private` qualifier on local variable
- Missing `public` qualifier on class or constructor header
- Keyword used as an identifier
- Common mathematical symbols used for operators (x • ÷ ≤ ≥ < > ≠)
- `[]` vs. `()` vs. `<>`
- `=` instead of `==` (and vice versa)
- Array/collection element access confusion (`[]` vs. `get` for r-values)
- Array/collection element modification confusion (`[]` vs. `set` for l-values)
- `length/size` confusion for array, `String`, and `ArrayList`, with or without `()`
- Extraneous `[]` when referencing entire array
- `[i,j]` instead of `[i][j]`
- Extraneous size in array declaration, (*e.g.,* `int[size] nums = new int[size];`)
- Missing `;` provided that line breaks and indentation clearly convey intent
- Missing `{ }` where indentation clearly conveys intent and `{ }` are used elsewhere
- Missing `( )` on parameter-less method or constructor invocations
- Missing `( )` around `if/while` conditions
- Use of local variable outside declared scope (must be within same method body)
- Failure to cast object retrieved from nongeneric collection

* *Spelling and case discrepancies for identifiers fall under the "No Penalty" category only if the correction can be* **unambiguously** *inferred from context; for example, "*`ArayList`*" instead of "*`ArrayList`*". As a counterexample, note that if the code declares "*`Bug bug;`*" and then uses "*`Bug.move()`*" instead of "*`bug.move()`*", the context does* **not** *allow for the reader to assume the object instead of the class.*

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.

3Aa

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)
{ ~~int index = -1;~~

    for(int i = 0; i < spaces.length ; i++)
        if (name.equals(spaces[i].getName()))
            return i;

    return (-1);
}
```

Part (b) begins on page 18.

3A6

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()
{ Horses[] temp = new Horses [spaces.length]
   int count = 0;
   for (int i = 0 ; i< spaces.length ; i++ ){
        if( spaces[i] != null ) {
            temp [count]= spaces[i];
            count ++;
        }
   }
    spaces = temp;
} // close method
```

3B

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)
```

```
public int findHorseSpace(String name) {
    for (int k=0; k < spaces.length; k++) {
        if (name.equals(spaces[k]))
            return k;
    }
    return -1;
}
```

Part (b) begins on page 18.

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()
```

```
public void consolidate(){
  for (k=0; k<spaces.length; k++){
    if( spaces[k]==null && spaces[k+1]!=null){
      Horses temp = spaces[k];
      spaces[k] = spaces[k+1];
      spaces[k+1] = null;
      k=0;
    }else k++;
  }
}
```

3C

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)
```

```
for(int i=0; i<spaces.length(); i++){
  if(spaces[i].getName().equals(name){
    return spaces[i];
  }
  else{
    return -1;
  }
}
```

Part (b) begins on page 18.

3C6

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()
```

```
for(int i=0; i<spaces.length(); i++){
    if(spaces[i].equals(null){
        spaces[i] = spaces[i+1];
    }
}
```

AP® COMPUTER SCIENCE A
2012 SCORING COMMENTARY

Question 3

Overview

This question entailed understanding array operations, array traversals, loops, and interfaces. Students were provided with an interface, `Horse`, and the framework for the `HorseBarn` class. In part (a) students were required to implement the method `findHorseSpace` that searches an array instance variable and returns the index of an element of the array that refers to an object whose name matches a parameter value. This could be done by traversing the array, skipping over `null` elements, and comparing the parameter with the value returned by the `getName` method of each object referenced by the elements. If no matching object was found, the method needed to return -1. In part (b) students were required to move the elements of the array such that the order of non-`null` elements was maintained, and those elements were clustered beginning at index 0 with no interspersed `null` elements. There were many ways to achieve this result, including two common approaches of either shifting array elements in place or using a temporary auxiliary collection.

Sample: 3A
Score: 8

In part (a) the student uses a loop to examine the elements in `spaces`. During each iteration of the loop, the student uses the `equals()` method to compare the name of the horse currently being examined with the name of the horse to find. The student does not check if the current array element is `null` before invoking `getName()` to obtain the name of the horse. If the current element is `null`, an exception will be thrown at runtime. This was a common error made by students and resulted in the loss of 1 point. Assuming that the comparison does not throw an exception, the student correctly returns the index of the space containing the horse, if the horse is in the barn. If the specified horse is not in the barn, the `return` statement after the loop correctly returns -1. Part (a) earned 3 points.

In part (b) the student creates a new array that is the same size as the array `spaces` and holds references to instances of `Horse` objects. The student uses a loop to iterate through `spaces` and copy, in order, the non-`null` elements to the new array. A local variable `count` is used to keep track of where the next reference should be copied to. Because `count` is initialized to 0 and is incremented by 1 each time a reference is copied to the new array, the references are copied to the new array starting at index 0 with no spaces between any two references. Note that because array elements are initialized to `null`, there is no need to make any further modifications to the new array. The last statement in the method changes the instance variable `spaces` so that it refers to the new array. Part (b) earned all 5 points.

Sample: 3B
Score: 5

In part (a) the student uses a loop to examine the elements in `spaces`. During each iteration of the loop, the student attempts to use the `equals()` method to compare the name of the horse currently being examined with the name of the horse to find. The student does not use the method `getName()` to obtain a reference to the name of the horse and so lost 1 point. Additionally, the student does not check if the current array element is `null` and so lost 1 point. Assuming that the comparison is done correctly, the student returns the index of the space containing the horse. If the specified horse is not in the barn, the `return` statement after the loop correctly returns -1. Part (a) earned 2 points.

Question 3 (continued)

In part (b) the student attempts to consolidate the barn by looking for situations in `spaces` where a `null` is followed immediately by a non-`null` element. Whenever this situation is detected, the elements are swapped. If this is done until there are no situations where `null` elements are followed by non-`null` elements, the barn will be consolidated. The loop in the solution iterates over the elements of the array checking for situations where a `null` element is followed by a non-`null` element. Because the loop runs through `spaces.length`, and element `spaces[k+1]` is examined in the `if` statement, an out of bounds error occurs, and 1 point was lost. Ignoring the out of bounds condition, the `if` statement executes the swap correctly. Note that the loop control variable is reset whenever a swap is made, ensuring that `spaces` will be examined until it is consolidated. The `else` clause of the `if` statement introduces a second error in the solution, as it increments the loop control variable whenever a swap is not made. This will cause elements in the array to be skipped, and consequently, it is possible that on exit some elements in `spaces` may not be contiguous. This resulted in the loss of 1 point. Part (b) earned 3 points.

Sample: 3C
Score: 2

Part (a) is an example of an early return, as the method will return during the first iteration of the loop. If the horse in the first space has the specified name, the method returns the correct index; otherwise, it will return -1. As a result of the early return, 2 points were lost: the loop does not access all of the entries in `spaces` (it only accesses the first one), and it will not always return the correct index. The solution lost an additional point because it does not check for a `null` reference in `spaces[i]` before attempting to obtain the name of the horse. Part (a) earned 1 point.

In part (b) the use of `i++` in the assignment statement resulted in the loss of several points. One point was lost because whenever `i++` is evaluated, the loop control variable is incremented and, as a result, the loop will skip over some of the entries in `spaces`. Additionally, the assignment statement does not make any changes to `spaces`, because `i` is incremented after the expression `spaces[i++]` is evaluated. This resulted in the loss of 1 point, because the solution does not assign an element in the array to a smaller index. Note that because `spaces` is not changed, the number, integrity, and order of all identified non-`null` elements are preserved, and the number of `nulls` is preserved. On the other hand, because `spaces` is not changed, it is not the case that all non-`null` elements in `spaces` will be in contiguous locations starting at index 0, which resulted in the loss of 1 point. The use of `equals()` to determine whether or not the contents of `spaces[i]` is `null` will result in an exception if `spaces[i]` is `null`. The student does not correctly differentiate between `null` and non-`null` elements in the array, and so 1 point was lost. Part (b) earned 1 point.