Apply the question-specific rubric first. To maintain scoring intent, a single error is generally accounted for only once per question thereby mitigating multiple penalties for the same error. The error categorization below is for cases not adequately covered by the question-specific rubric. Note that points can only be deducted if the error occurs in a part that has earned credit via the question-specific rubric. Any particular error is penalized only once in a question, even if it occurs on different parts of that question.

**Nonpenalized Errors**
- spelling/case discrepancies if no ambiguity
- local variable not declared if others are declared in some part
- use keyword as identifier
- [] vs. () vs. <>
- = instead of == (and vice versa)
- length/size confusion for array, String, and ArrayList, with or without()
- private qualifier on local variable
- extraneous code with no side effect; e.g., precondition check
- common mathematical symbols for operators (x • ÷ ≤ ≥ <> ≠)
- missing {} where indentation clearly conveys intent and {} used elsewhere
- default constructor called without parens; e.g., new Fish;
- missing () on parameterless method call
- missing () around if/while conditions
- missing ; when majority are present
- missing public on class or constructor header
- extraneous [] when referencing entire array
- extraneous size in array declaration, e.g., int[size] nums = new int[size];

**Minor Errors (1/2 point)**
- confused identifier (e.g., len for length or left() for getLeft())
- local variables used but none declared
- modifying new in constructor call
- modifying a constant (final)
- use equals or compareTo method on primitives, e.g., int x; …x.equals(val)
- array/collection access confusion ([]) get
- assignment dyslexia, e.g., x + 3 = y; for y = x + 3;
- super(method()) instead of super.method()
- formal parameter syntax (with type) in method call, e.g., a = method(int x)
- missing public from method header when required
- "false"/"true" or 0/1 for boolean values
- "null" for null

**Major Errors (1 point)**
- extraneous code that causes side effect; e.g., information written to output
- interface or class name instead of variable identifier; e.g., Bug.move() instead of aBug.move()
- method(obj) instead of obj.method()
- attempt to use private data or method when not accessible
- destruction of persistent data (e.g., changing value referenced by parameter)
- use class name in place of super in constructor or method call
- void method (or constructor) returns a value

### Applying Minor Errors (½ point):
A minor error that occurs exactly once when the same concept is correct two or more times is regarded as an oversight and not penalized. A minor error must be penalized if it is the only instance, one of two, or occurs two or more times.

* Spelling and case discrepancies for identifiers fall under the “nonpenalized” category only if the correction can be unambiguously inferred from context; for example, “ArrayList” instead of “ArrayList”. As a counter example, note that if a student declares “Bug bug;” then uses “Bug.move()” instead of “bug.move()”, the context does not allow for the reader to assume the object instead of the class.
Question 3: Trail

**Part (a)** isLevelTrailSegment 5 points

**Intent:** Return true if maximum difference ≤ 10 (segment is level); false otherwise

+3 Determination of information needed to test level-trail condition
+1/2 Creates and maintains local state for determination of maximum (or minimum); alternate solution: tests difference in elevations
+1/2 Accesses the value of any element of this.markers
+1 All and only appropriate elements of this.markers participate in determination of information needed to test level-trail condition; no out-of-bounds access potential
+1 Compares element to state in context of updating maximum (or minimum); alternate solution: tests difference in elevations
+1 Correctly determines information needed to test level-trail condition for the elements examined; must address two or more pairs of elements
+1 Returns true if determined maximum difference is ≤ 10, false otherwise

**Part (b)** isDifficult 4 points

**Intent:** Return true if trail is difficult (based on number of changes of given magnitude); false otherwise

+3 Determine number of changes, greater than or equal to 30, between consecutive values in this.markers
+1/2 Creates, initializes and accumulates a count of number of changes
+1/2 Accesses the value of any element of this.markers in context of iteration
+1/2 Accesses the value of all elements of this.markers, no out-of-bounds access potential
+1/2 Computes difference of all and only consecutive values in this.markers
+1 Updates accumulated count if and only if absolute value of difference is >= 30
+1 Returns true if accumulated count is >= 3; false otherwise
Question 3: Trail

Part (a):

```java
public boolean isLevelTrailSegment(int start, int end) {
    int min = this.markers[start];
    int max = this.markers[start];
    for (int i = start + 1; i <= end; i++) {
        if (min > this.markers[i]) {
            min = this.markers[i];
        }
        if (max < this.markers[i]) {
            max = this.markers[i];
        }
    }
    return ((max - min) <= 10);
}

// Alternative solution (compares differences; uses early return):
public boolean isLevelTrailSegment(int start, int end) {
    for (int i = start; i < end; i++) {
        for (int j = start + 1; j <= end; j++) {
            if (Math.abs(this.markers[i] - this.markers[j]) > 10) {
                return false;
            }
        }
    }
    return true;
}
```

Part (b):

```java
public boolean isDifficult() {
    int numChanges = 0;
    for (int i = 0; i < this.markers.length - 1; i++) {
        if (Math.abs(this.markers[i] - this.markers[i + 1]) >= 30) {
            numChanges++;
        }
    }
    return (numChanges >= 3);
}
```
Complete method isLevelTrailSegment below.

/**
 * Determines if a trail segment is level. A trail segment is defined by a starting marker,
 * an ending marker, and all markers between those two markers.
 * A trail segment is level if it has a difference between the maximum elevation
 * and minimum elevation that is less than or equal to 10 meters.
 * @param start the index of the starting marker
 * @param end the index of the ending marker
 * Precondition: 0 <= start < end <= markers.length - 1
 * @return true if the difference between the maximum and minimum
 * elevation on this segment of the trail is less than or equal to 10 meters;
 * false otherwise.
 */
public boolean isLevelTrailSegment(int start, int end)
{
    int max = Integer.MIN_VALUE;
    int min = Integer.MAX_VALUE;
    for (int i = start; i <= end; i++)
    {
        if (markers[i] > max)
            max = markers[i];
        if (markers[i] < min)
            min = markers[i];
    }
    return (max - min) <= 10;
}
Complete method isDifficult below.

```java
/**
 * Determines if this trail is difficult. A trail is rated by counting the number of changes in
 * elevation that are at least 30 meters (up or down) between two consecutive markers. A trail
 * with 3 or more such changes is rated difficult.
 * @return true if the trail is rated difficult; false otherwise.
 */
public boolean isDifficult()

    int count = 0;
    for (int i = 0; i < markers.length - 1; i++)
        int val = Math.abs(markers[i] - markers[i + 1]);
        if (val >= 30)
            count ++;
    return count >= 3;
```
Complete method isLevelTrailSegment below.

/**
 * Determines if a trail segment is level. A trail segment is defined by a starting marker, 
 * an ending marker, and all markers between those two markers.
 * A trail segment is level if it has a difference between the maximum elevation 
 * and minimum elevation that is less than or equal to 10 meters.
 * @param start the index of the starting marker 
 * @param end the index of the ending marker 
 * @return true if the difference between the maximum and minimum 
 * elevation on this segment of the trail is less than or equal to 10 meters; 
 * false otherwise. 
 */

public boolean isLevelTrailSegment(int start, int end) {
    int min = start, max = start;
    for (int i = start; i <= end; i++) {
        if (markers[i] > max)
            max = markers[i];
        if (markers[i] < min)
            min = markers[i];
    }
    return ((max - min) <= 10);
}
/** Determines if this trail is difficult. A trail is rated by counting the number of changes in
* elevation that are at least 30 meters (up or down) between two consecutive markers. A trail
* with 3 or more such changes is rated difficult.
* @return true if the trail is rated difficult; false otherwise.
*/
public boolean isDifficult() {
    int changes = 0;
    for (int i = 0; i < markers.length - 1; i++) {
        if (Math.abs(markers[i] - markers[i + 1]) > 30)
            changes++;
    }
    return (changes >= 3);
Complete method `isLevelTrailSegment` below.

```java
/**
 * Determines if a trail segment is level. A trail segment is defined by a starting marker,
 * an ending marker, and all markers between those two markers.
 * A trail segment is level if it has a difference between the maximum elevation
 * and minimum elevation that is less than or equal to 10 meters.
 * @param start the index of the starting marker
 * @param end the index of the ending marker
 * @return true if the difference between the maximum and minimum
 * elevation on this segment of the trail is less than or equal to 10 meters;
 * false otherwise.
 */
public boolean isLevelTrailSegment(int start, int end) {
    int elev = 0;
    boolean isTrue = true;
    while (isTrue == true) {
        while (start <= end) {
            elev = markers[start] - markers[start + 1];
            if (elev < 10) {
                isTrue = true;
            } else {
                isTrue = false;
            }
            start += 1;
        }
    }
    return isTrue;
}
```

Part (b) begins on page 14.
Complete method isDifficult below.

/**
 * Determines if this trail is difficult. A trail is rated by counting the number of changes in
 * elevation that are at least 30 meters (up or down) between two consecutive markers. A trail
 * with 3 or more such changes is rated difficult.
 * @return true if the trail is rated difficult; false otherwise.
 */
public boolean isDifficult()
{
    int elev = 0;
    int count = 0;
    for (int x = 0; x < markers.length; x++)
    {
        elev = markers[x] - markers[x+1];
        if (elev >= 30 || elev <= -30)
            count ++;
        if (count >= 3)
            return true;
        else
            return false;
    }
}
Overview

This question focused on array traversal, finding an extreme value (minimum and/or maximum) within a specified segment of an array, and counting occurrences of array elements satisfying a given condition. Students were given a class that contained an instance data array that represented elevations and were asked to write two unrelated methods. Part (a), isLevelTrailSegment, required students to determine whether there was an overall difference of at most 10 in a segment of the array defined by the two parameters passed to the method. The wording of the question led students to a solution of first finding the maximum and minimum values and then checking if the difference of these values was at most 10. That approach was not required, and some students used alternatives that did not involve such computations. The second method, isDifficult, required students to examine the differences of pairs of consecutive array element values and determine if at least three of those differences had an absolute value greater than 30.

Sample: 3A
Score: 9

In part (a) the student creates and maintains variables for determining the highest and lowest elevations. The loop correctly examines all elements in the array markers between start and end. There are correct comparisons to determine the highest and lowest elevations. The student returns a correct value based on the difference between the highest and lowest elevations that were considered. Part (a) earned all 5 points.

In part (b) the student correctly creates, initializes and accumulates a count of the number of large elevation changes. The loop accesses all elements of the array markers and computes the differences of all consecutive elevations. There is a correct comparison of the absolute value of the computed difference to 30. The student returns a correct value based on the number of large elevation changes. Part (b) earned all 4 points.

Sample: 3B
Score: 7

In part (a) the student creates and maintains variables for determining the highest and lowest elevations. The loop correctly examines all elements in the array markers between start and end. There are correct comparisons to determine the highest and lowest elevations; however, these calculated values are incorrect because min and max are initialized to the parameter start. The student returns a correct value based on the difference between the highest and lowest elevations that were considered. Part (a) earned 4 points.

In part (b) the student correctly creates, initializes and accumulates a count of the number of large elevation changes. The loop accesses all elements of the array markers and computes the differences of all consecutive elevations. The comparison of the absolute value of the computed difference incorrectly considers only values above 30. The student returns a correct value based on the number of large elevation changes. Part (b) earned 3 points.
Sample: 3C
Score: 3

In part (a) the student does not create or maintain any variables for determining the highest and lowest elevations. There are accesses to elements in the array markers, but there is an out-of-bounds error. There is no comparison to a minimum or maximum state variable. The test for being level only considers consecutive elevation changes. The student incorrectly returns the result of only the last level test. Part (a) earned ½ point.

In part (b) the student correctly creates, initializes and accumulates a count of the number of large elevation changes. The loop accesses elements of the array markers but has an out-of-bounds access as well. The student correctly computes the differences of all consecutive elevations but incorrectly compares the computed difference to -30. The student returns a correct value based on the number of large elevation changes. Part (b) earned 2½ points.