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.

<table>
<thead>
<tr>
<th>Nonpenalized Errors</th>
<th>Minor Errors (1/2 point)</th>
<th>Major Errors (1 point)</th>
</tr>
</thead>
<tbody>
<tr>
<td>spelling/case discrepancies if no ambiguity*</td>
<td>confused identifier (e.g., <code>len</code> for <code>length</code> or <code>left()</code> for <code>getLeft()</code> )</td>
<td>extraneous code that causes side effect; e.g., <code>information written to output</code></td>
</tr>
<tr>
<td>local variable not declared if others are declared in some part</td>
<td>local variables used but none declared</td>
<td>interface or class name instead of variable identifier; e.g., <code>Bug.move()</code> instead of <code>aBug.move()</code></td>
</tr>
<tr>
<td>use keyword as identifier</td>
<td>modifying a constant (final)</td>
<td><code>aMethod(obj)</code> instead of <code>obj.aMethod()</code></td>
</tr>
<tr>
<td><code>[]</code> vs. <code>()</code> vs. <code>&lt;&gt;</code></td>
<td>use <code>equals</code> or <code>compareTo</code> method on primitives, e.g., <code>int x; ...x.equals(val)</code></td>
<td>attempt to use private data or method when not accessible</td>
</tr>
<tr>
<td><code>=</code> instead of <code>==</code> (and vice versa)</td>
<td>array/collection access confusion (<code>[]</code> get)</td>
<td>use class name in place of <code>super</code> in constructor or method call</td>
</tr>
<tr>
<td>length/size confusion for array, <code>String</code>, and <code>ArrayList</code>, with or without <code>()</code></td>
<td>assignment dyslexia, e.g., <code>x + 3 = y; for y = x + 3;</code></td>
<td><code>void</code> method (or constructor) returns a value</td>
</tr>
<tr>
<td>private qualifier on local variable</td>
<td><code>super(method())</code> instead of <code>super.method()</code></td>
<td></td>
</tr>
<tr>
<td>extraneous code with no side effect; e.g., <code>precondition check</code></td>
<td>formal parameter syntax (with type) in method call, e.g., <code>a = method(int x)</code></td>
<td></td>
</tr>
<tr>
<td>common mathematical symbols for operators (x • ÷ ≥ ≥ &lt; &gt; ≠)</td>
<td>missing <code>public</code> from method header when required</td>
<td></td>
</tr>
<tr>
<td>missing <code>{ }</code> where indentation clearly conveys intent and <code>{ }</code> used elsewhere</td>
<td>&quot;false&quot;/&quot;true&quot; or 0/1 for boolean values</td>
<td></td>
</tr>
<tr>
<td>default constructor called without parens; e.g., <code>new Fish;</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td>missing <code>( )</code> on parameterless method call</td>
<td>&quot;null&quot; for <code>null</code></td>
<td></td>
</tr>
<tr>
<td>missing <code>( )</code> around <code>if/while conditions</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td>missing <code>;</code> when majority are present</td>
<td></td>
<td></td>
</tr>
<tr>
<td>missing <code>public</code> on class or constructor header</td>
<td></td>
<td></td>
</tr>
<tr>
<td>extraneous <code>[]</code> when referencing entire array</td>
<td></td>
<td></td>
</tr>
<tr>
<td>extraneous size in array declaration, e.g., <code>int[size] nums = new int[size];</code></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

* 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 1: Master Order

<table>
<thead>
<tr>
<th>Part (a)</th>
<th>getTotalBoxes 3 points</th>
</tr>
</thead>
</table>

**Intent:** Compute and return the sum of the number of boxes of all cookie orders in this.orders

+1 Considers all CookieOrder objects in this.orders
+1/2 Accesses any element of this.orders
+1/2 Accesses all elements of this.orders with no out-of-bounds access potential

+1 1/2 Computes total number of boxes
  +1/2 Creates an accumulator (declare and initialize)
  +1/2 Invokes getNumBoxes on object of type CookieOrder
  +1/2 Correctly accumulates total number of boxes

+1/2 Returns computed total

<table>
<thead>
<tr>
<th>Part (b)</th>
<th>removeVariety 6 points</th>
</tr>
</thead>
</table>

**Intent:** Remove all CookieOrder objects from this.orders whose variety matches cookieVar; return total number of boxes removed

+4 Identifies and removes matching CookieOrder objects
+1/2 Accesses an element of this.orders
+1/2 Compares parameter cookieVar with getVariety() of a CookieOrder object (must use .equals or .compareTo)
+1 Compares parameter cookieVar with getVariety() of all CookieOrder objects in this.orders, no out-of-bounds access potential
+1/2 Removes an element from this.orders
+1/2 Removes only matching CookieOrder objects
+1 Removes all matching CookieOrder objects, no elements skipped

+1 1/2 Computes total number of boxes in removed CookieOrder objects
  +1/2 Creates an accumulator (declare and initialize)
  +1/2 Invokes getNumBoxes on object of type CookieOrder
  +1/2 Correctly accumulates total number of boxes (must be in context of loop and match with cookieVar)

+1/2 Returns computed total

**Usage:**
-1 consistently references incorrect name instead of orders, of potentially correct type
-1 1/2 consistently references incorrect name instead of orders, incorrect type (e.g., this, MasterOrder)
**Question 2: APLine**

**Intent:** Design complete APLine class including constructor, getSlope and isOnLine methods.

+1 Complete, correct header for APLine [class APLine]

*Note: Accept any visibility except private*

+1 1/2 State maintenance

+1/2 Declares at least one instance variable capable of maintaining numeric value
+1/2 Declares at least three instance variables capable of maintaining numeric values
+1/2 All state variables have private visibility

*Note: Accept any numeric type (primitive or object)*

*Note: Accept any distinct Java-valid variable names*

+1 1/2 APLine Constructor

*Method header*

+1/2 Correctly formed header (visibility not private; name APLine)
+1/2 Specifies exactly three numeric parameters

*Method body*

+1/2 Sets appropriate state variables based on parameters (no shadowing errors)

*Note: Interpret instance fields by usage not by name*

+2 1/2 getSlope

*Method header*

+1/2 Correct method header (visibility not private; type double or Double; name getSlope; parameterless)

*Method body*

+1/2 Computation uses correct formula for slope
+1 Computation uses double precision (no integer division)
+1/2 Returns computed value

+2 1/2 isOnLine

*Method header*

+1/2 Correct formed header (visibility not private; type boolean or Boolean, name isOnLine)
+1/2 Specifies exactly two numeric parameters

*Method body*

+1/2 Computation uses correct formula involving state and parameters

\( a \times x + b \times y + c \)

+1/2 Computation uses correct comparison test (equal to zero)
+1/2 Returns true if is on this APLine; false otherwise
Question 3: Trail

<table>
<thead>
<tr>
<th>Part (a)</th>
<th>isLevelTrailSegment</th>
<th>5 points</th>
</tr>
</thead>
</table>

**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

<table>
<thead>
<tr>
<th>Part (b)</th>
<th>isDifficult</th>
<th>4 points</th>
</tr>
</thead>
</table>

**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 4: GridChecker (GridWorld)

Part (a)  actorWithMostNeighbors  4 points

Intent: Identify and return actor in this.gr with most neighbors; return null if no actors in grid

+1  Consider all occupied locations or all actors in grid
+1/2  Iterates over all occupied locations in this.gr
+1/2  Performs action using actor or location from this.gr within iteration

+1 1/2  Determination of maximum number of neighbors
+1/2  Determines number of occupied neighboring locations of a location
+1  Correctly determines maximum number of neighbors

+1  Return actor
+1/2  Returns reference to Actor (not Location)
+1  Returns reference to a correct actor; null if no actors in this.gr

*Note: This may be done using getOccupiedAdjacentLocations, getNeighbors, or an iterative get of surrounding locations

Part (b)  getOccupiedWithinTwo  5 points

Intent: Return list of all occupied locations within 2 rows/columns of parameter, parameter excluded

+1/2  Creates and initializes local variable to hold collection of locations

+2  Consider surrounding locations
+1/2  Considers at least two locations 1 row and/or 1 column away from parameter
+1/2  Considers at least two locations 2 rows and/or 2 columns away from parameter
+1  Correctly identifies all and only valid locations within 2 rows and 2 columns of parameter

+1  Collect occupied locations†
+1/2  Adds any location object to collection
+1/2  Adds location to collection only if occupied

+1 1/2  Return list of locations
+1/2  Returns reference to a list of locations
+1/2  List contains all and only identified locations†
+1/2  Parameter loc excluded from returned list

†Note: Duplication of locations in returned list is not penalized

Usage: −1/2 parameter dyslexia in new Location constructor invocation
Question 1: Master Order

Part (a):

```java
public int getTotalBoxes() {
    int sum = 0;
    for (CookieOrder co : this.orders) {
        sum += co.getNumBoxes();
    }
    return sum;
}
```

Part (b):

```java
public int removeVariety(String cookieVar) {
    int numBoxesRemoved = 0;
    for (int i = this.orders.size() - 1; i >= 0; i--) {
        if (cookieVar.equals(this.orders.get(i).getVariety())) {
            numBoxesRemoved += this.orders.get(i).getNumBoxes();
            this.orders.remove(i);
        }
    }
    return numBoxesRemoved;
}
```

// Alternative solution (forward traversal direction):

```java
public int removeVariety(String cookieVar) {
    int numBoxesRemoved = 0;
    int i = 0;
    while (i < this.orders.size()) {
        if (cookieVar.equals(this.orders.get(i).getVariety())) {
            numBoxesRemoved += this.orders.get(i).getNumBoxes();
            this.orders.remove(i);
        } else {
            i++;
        }
    }
    return numBoxesRemoved;
}
```
public class APLine {
    /** State variables. Any numeric type; object or primitive. */
    private int a, b, c;

    /** Constructor with 3 int parameters. */
    public APLine(int a, int b, int c) {
        this.a = a;
        this.b = b;
        this.c = c;
    }

    /** Determine the slope of this APLine. */
    public double getSlope() {
        return (- (this.a / (double) this.b));
    }

    /** Determine if coordinates represent a point on this APLine. */
    public boolean isOnLine(int x, int y) {
        return (0 == (this.a * x) + (this.b * y) + this.c);
    }
}

// Alternative solution (state variables of type double):

public class APLine {
    private double a1, b1, c1;

    public APLine(int a, int b, int c) {
        this.a1 = a;
        this.b1 = b;
        this.c1 = c;
    }

    public double getSlope() {
        return -(this.a1 / this.b1);
    }

    public boolean isOnLine(int x, int y) {
        return (0 == (this.a1 * x) + (this.b1 * y) + this.c1);
    }
}
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);
}
```
AP® COMPUTER SCIENCE A
2010 CANONICAL SOLUTIONS

Question 4: GridChecker (GridWorld)

Part (a):

```java
public Actor actorWithMostNeighbors() {
    if (0 == this.gr.getOccupiedLocations().size()) {
        return null;
    }
    Location where = null;
    int most = -1;
    for (Location loc : this.gr.getOccupiedLocations()) {
        if (most < this.gr.getOccupiedAdjacentLocations(loc).size()) {
            most = this.gr.getOccupiedAdjacentLocations(loc).size();
            where = loc;
        }
    }
    return this.gr.get(where);
}

// Alternative solution (uses getNeighbors):

public Actor actorWithMostNeighbors() {
    if (0 == this.gr.getOccupiedLocations().size()) {
        return null;
    }
    Location where = this.gr.getOccupiedLocations().get(0);
    for (Location loc : this.gr.getOccupiedLocations()) {
        if (this.gr.getNeighbors(where).size() <
            this.gr.getNeighbors(loc).size()) {
            where = loc;
        }
    }
    return this.gr.get(where);
}
```

These canonical solutions serve an expository role, depicting general approaches to a 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.
Question 4: GridChecker (GridWorld) (continued)

Part (b):

public List<Location> getOccupiedWithinTwo(Location loc) {
    List<Location> occupied = new ArrayList<Location>();
    for (int row = loc.getRow() - 2; row <= loc.getRow() + 2; row++) {
        for (int col = loc.getCol() - 2; col <= loc.getCol() + 2; col++) {
            Location loc1 = new Location(row, col);
            if (gr.isValid(loc1) && this.gr.get(loc1) != null &&
                !loc1.equals(loc)) {
                occupied.add(loc1);
            }
        }
    }
    return occupied;
}

// Alternative solution (uses getOccupiedLocations):

global List<Location> getOccupiedWithinTwo(Location loc) {
    List<Location> occupied = new ArrayList<Location>();
    for (Location loc1 : this.gr.getOccupiedLocations()) {
        if ((Math.abs(loc.getRow() - loc1.getRow()) <= 2) &&
            (Math.abs(loc.getCol() - loc1.getCol()) <= 2) &&
            !loc1.equals(loc)) {
            occupied.add(loc1);
        }
    }
    return occupied;
}