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.
Question 1: Climbing Club

Part (a)  addClimb (append)  2 points

Intent: Create new ClimbInfo using data from parameters and append to climbList

+1 Creates new ClimbInfo object using parametric data correctly
+1 Appends the created object to climbList (no bounds error and no destruction of existing data) (point not awarded if inserted more than once)

Part (b)  addClimb (alphabetical)  6 points

Intent: Create new ClimbInfo object using data from parameters and insert into climbList, maintaining alphabetical order

+1 Creates new ClimbInfo object(s), using parametric data correctly
+1 Compares peakName value with value retrieved from object in list (must use getName)
+1 Inserts object into list based on a comparison (other than equality) with object in list (point not awarded if inserted more than once)
+1 Compares parametric data with all appropriate entries in climbList (no bounds error)
+1 Inserts new ClimbInfo object into climbList (no destruction of existing data)
+1 Inserts new ClimbInfo object into climbList once and only once in maintaining alphabetical order (no destruction of existing data)

Part (c)  analysis  1 point

Intent: Analyze behavioral differences between append and alphabetical versions of addClimb

+1 (i) NO (ii) YES Both must be answered correctly

Question-Specific Penalties

-1 (2) Attempts to return a value from addClimb
Question 1: Climbing Club

Part (a):

public void addClimb(String peakName, int climbTime) {
    this.climbList.add(new ClimbInfo(peakName, climbTime));
}

Part (b):

public void addClimb(String peakName, int climbTime) {
    for (int i = 0; i < this.climbList.size(); i++) {
        if (peakName.compareTo(this.climbList.get(i).getName()) <= 0) {
            this.climbList.add(i, new ClimbInfo(peakName, climbTime));
            return;
        }
    }
    this.climbList.add(new ClimbInfo(peakName, climbTime));
}

Part (c):

NO
YES
Question 2: RetroBug (GridWorld)

<table>
<thead>
<tr>
<th>Class: RetroBug</th>
<th>9 points</th>
</tr>
</thead>
</table>

**Intent:** Define extension to Bug class that implements a restore method to revert to previous location and direction

+1 Provides properly formed class header for RetroBug that extends Bug class

+1 Overrides at least one Bug method, other than constructor, and maintains all Bug behaviors

+2 Saves state at beginning of act
   +1 Remembers location or direction in RetroBug instance variable at beginning of act method and nowhere else
   (point awarded only if instance variable is explicitly declared)
   +1 Remembers both location and direction in RetroBug instance variables

+5 Implements restore
   +½ Provides correct method header: public void restore()
   +½ Guards against any effect if called before first invocation of act
   +1 Always restores remembered direction
   +1 Moves to remembered location
   +1 Moves if remembered location is empty (must check for empty location)
   +1 Moves if remembered location is occupied only by a flower (must check for flower at location)

**Question-Specific Penalties**

-1 (r) Use of "RetroBug." instead of "this."

-1 (v) Confused use of location and direction
   (e.g., saved location used as direction and vice versa)

-1 (z) Attempts to return a value from restore

-0 Missing public qualifier on class header
Question 2: RetroBug (GridWorld)

```java
public class RetroBug extends Bug {
    Location savedLocation;
    int savedDirection;

    public void act() {
        savedLocation = getLocation();
        savedDirection = getDirection();
        super.act();
    }

    public void restore() {
        if (savedLocation == null) return;
        setDirection(savedDirection);
        if (getGrid().get(savedLocation) == null
             || getGrid().get(savedLocation) instanceof Flower) {
            moveTo(savedLocation);
        }
    }
}
```
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
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.
Question 4: GrayImage

**Part (a)  countWhitePixels  4 points**

**Intent:** Return the number of white pixels in the image

+1 Accesses all entries in `pixelValues` (no bounds errors)
+1 Compares an entry of array with `WHITE` or with 255 in context of iteration
+1 Initializes and increments a counter
+1 Returns correct count of number of white pixels in `pixelValues`

**Part (b)  processImage  5 points**

**Intent:** Process elements of `pixelValues` and apply specified formula

+1 Accesses all necessary elements in at least one row or one column of `pixelValues`
+1 Accesses all necessary elements of `pixelValues` (no bounds errors)
+1 Decrements element at index `[a][b]` by the original value found in element at index `[a+2][b+2]`
+1 Modifies all and only appropriate elements of `pixelValues` (changes must not affect last two rows and columns)
+1 Assigns `BLACK` or 0 to elements of `pixelValues` that would otherwise have a value less than `BLACK` (negative value)

**Question-Specific Penalties**

-1 (z) Attempts to return a value from `processImage`
Question 4: GrayImage

Part (a):

```java
public int countWhitePixels() {
    int whitePixelCount = 0;
    for (int[] row : this.pixelValues) {
        for (int pv : row) {
            if (pv == this.WHITE) {
                whitePixelCount++;
            }
        }
    }
    return whitePixelCount;
}
```

Part (a): Alternative solution

```java
public int countWhitePixels() {
    int whitePixelCount = 0;
    for (int row = 0; row < pixelValues.length; row++) {
        for (int col = 0; col < pixelValues[0].length; col++) {
            if (pixelValues[row][col] == WHITE) {
                whitePixelCount++;
            }
        }
    }
    return whitePixelCount;
}
```

Part (b):

```java
public void processImage() {
    for (int row = 0; row < this.pixelValues.length-2; row++) {
        for (int col = 0; col < this.pixelValues[0].length-2; col++) {
            this.pixelValues[row][col] -= this.pixelValues[row+2][col+2];
            if (this.pixelValues[row][col] < BLACK) {
                this.pixelValues[row][col] = BLACK;
            }
        }
    }
}
```

Part (b): Alternative solution

```java
public void processImage() {
    for (int row = 0; row < this.pixelValues.length; row++) {
        for (int col = 0; col < this.pixelValues[0].length; col++) {
            if (row + 2 < pixelValues.length &&
                col + 2 < pixelValues[row].length) {
                this.pixelValues[row][col] -= this.pixelValues[row+2][col+2];
                if (this.pixelValues[row][col] < BLACK) {
                    this.pixelValues[row][col] = BLACK;
                }
            }
        }
    }
}
```

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.