

**AP[®] COMPUTER SCIENCE A
2010 SCORING GUIDELINES**

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 1/2 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: -½ parameter dyslexia in new `Location` constructor invocation

**AP[®] COMPUTER SCIENCE A
2010 CANONICAL SOLUTIONS**

Question 4: GridChecker (GridWorld)

Part (a):

```
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.

**AP[®] COMPUTER SCIENCE A
2010 CANONICAL SOLUTIONS**

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):

```
public 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;
}
```

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.