



AP[®] Computer Science A 2004 Sample Student Responses

The materials included in these files are intended for noncommercial use by AP teachers for course and exam preparation; permission for any other use must be sought from the Advanced Placement Program[®]. Teachers may reproduce them, in whole or in part, in limited quantities, for face-to-face teaching purposes but may not mass distribute the materials, electronically or otherwise. This permission does not apply to any third-party copyrights contained herein. These materials and any copies made of them may not be resold, and the copyright notices must be retained as they appear here.

The College Board is a not-for-profit membership association whose mission is to connect students to college success and opportunity. Founded in 1900, the association is composed of more than 4,500 schools, colleges, universities, and other educational organizations. Each year, the College Board serves over three million students and their parents, 23,000 high schools, and 3,500 colleges through major programs and services in college admissions, guidance, assessment, financial aid, enrollment, and teaching and learning. Among its best-known programs are the SAT[®], the PSAT/NMSQT[®], and the Advanced Placement Program[®] (AP[®]). The College Board is committed to the principles of excellence and equity, and that commitment is embodied in all of its programs, services, activities, and concerns.

For further information, visit www.collegeboard.com

Copyright © 2004 College Entrance Examination Board. All rights reserved. College Board, Advanced Placement Program, AP, AP Central, AP Vertical Teams, APCD, Pacesetter, Pre-AP, SAT, Student Search Service, and the acorn logo are registered trademarks of the College Entrance Examination Board. PSAT/NMSQT is a registered trademark of the College Entrance Examination Board and National Merit Scholarship Corporation. Educational Testing Service and ETS are registered trademarks of Educational Testing Service. Other products and services may be trademarks of their respective owners.

For the College Board's online home for AP professionals, visit AP Central at apcentral.collegeboard.com.

- (a) Write the `PondStocker` method `numUnder`. Method `numUnder` returns the smallest number of fish that must be added to make the density of fish in the environment greater than `minDensity`. If the density of fish in the environment is already greater than `minDensity`, then `numUnder` returns zero. Recall that the `Environment` methods `numRows` and `numCols` return the number of rows and the number of columns, respectively, in an environment.

Complete method `numUnder` below.

```
// postcondition: returns the minimum number of fish that need to be
//                added to make the population density greater than
//                minDensity
private int numUnder()
{
    int numSpaces = theEnv.numRows() * theEnv.numCols();
    int filledSpaces = theEnv.numObjects();
    int needed = 0;

    while(((double)needed + filledSpaces) / ((double)numSpaces) <= minDensity)
    {
        needed++;
    }
    return needed;
}
```

Part (b) begins on page 14.

GO ON TO THE NEXT PAGE.

- (b) Write the `PondStocker` method `randomLocation`. Method `randomLocation` returns a random location within the bounds of the environment.

In writing `randomLocation`, you may use any of the accessible methods of the classes in the case study. Solutions that reimplement functionality provided by these methods, rather than invoking these methods, will not receive full credit.

Complete method `randomLocation` below.

```
// postcondition: returns a random location within the bounds of theEnv  
private Location randomLocation()
```

```
{
```

```
    Random randNumGen = Random Number Generator.getInstance();  
    return new Location(randNumGen.nextInt(theEnv.numRows())  
                        randNumGen.nextInt(theEnv.numCols()));  
}
```

GO ON TO THE NEXT PAGE.

- (c) Write the `PondStocker` method `addFish`. Method `addFish` adds `numToAdd` Fish to the environment at random locations that are not already occupied. You may use the two-parameter `Fish` constructor, so that the fish added have a random direction and color.

In writing `addFish`, you may call `randomLocation`. Assume that `randomLocation` works as specified, regardless of what you wrote in part (b). You may also use any of the accessible methods of the classes in the case study. Solutions that reimplement functionality provided by these methods, rather than invoking these methods, will not receive full credit.

Complete method `addFish` below.

```
// precondition:  0 <= numToAdd <= number of empty locations in theEnv
// postcondition: the number of fish in theEnv has been increased
//                by numToAdd; the fish added are placed at
//                random empty locations in theEnv
public void addFish(int numToAdd)
```

```
{
```

```
  for (int x = 0; x < numToAdd; x++)
```

```
  {
```

```
    Location nextLocation = randomLocation();
```

```
    while (!theEnv.is Empty(nextLocation))
```

```
    {
```

```
      nextLocation = randomLocation();
```

```
    }
```

```
    Fish current = new Fish(theEnv, nextLocation);
```

```
  }
```

```
}
```

GO ON TO THE NEXT PAGE.

A3 B.

- (a) Write the PondStocker method numUnder. Method numUnder returns the smallest number of fish that must be added to make the density of fish in the environment greater than minDensity. If the density of fish in the environment is already greater than minDensity, then numUnder returns zero. Recall that the Environment methods numRows and numCols return the number of rows and the number of columns, respectively, in an environment.

Complete method numUnder below.

```
// postcondition: returns the minimum number of fish that need to be
//                added to make the population density greater than
//                minDensity
private int numUnder() {
    int base = the Env. numRows() * the Env. numCols();
    Locatable [ ] the Fishes = the Env. allObjects();
    int numFish = the Fishes.length;
    double currentDensity = (Double) numFish / base;
    while (currentDensity < minDensity) {
        numFish++;
        currentDensity = (Double) numFish / base;
    }
    int numUnder = numFish - the Fishes.length;
    return numUnder;
}
```

Part (b) begins on page 14.

GO ON TO THE NEXT PAGE.

B₂

- (b) Write the PondStocker method randomLocation. Method randomLocation returns a random location within the bounds of the environment.

In writing randomLocation, you may use any of the accessible methods of the classes in the case study. Solutions that reimplement functionality provided by these methods, rather than invoking these methods, will not receive full credit.

Complete method randomLocation below.

```
// postcondition: returns a random location within the bounds of the Env
private Location randomLocation() {
```

```
    Random rand = new Random();
    int randRow = rand.nextInt (theEnv.numRows);
    int randCol = rand.nextInt (theEnv.numCols);
    Location randomLocation = new Location (randRow, randCol);
    IF (isValid (randomLocation) == true)
        return randomLocation; }
```

GO ON TO THE NEXT PAGE.

- (c) Write the PondStocker method addFish. Method addFish adds numToAdd Fish to the environment at random locations that are not already occupied. You may use the two-parameter Fish constructor, so that the fish added have a random direction and color.

In writing addFish, you may call randomLocation. Assume that randomLocation works as specified, regardless of what you wrote in part (b). You may also use any of the accessible methods of the classes in the case study. Solutions that reimplement functionality provided by these methods, rather than invoking these methods, will not receive full credit.

Complete method addFish below.

```
// precondition: 0 <= numToAdd <= number of empty locations in theEnv
// postcondition: the number of fish in theEnv has been increased
//                by numToAdd; the fish added are placed at
//                random empty locations in theEnv
public void addFish(int numToAdd){
```

```
    for (int i=0; i <= numToAdd; i++){
```

```
        Location randomLocation = new Location();
```

```
        if (is Empty (randomLocation) == true){
```

```
            theEnv.add (new Fish (theEnv, randomLocation, theEnv.randomDirection);
                randomColor(); } } }
```

GO ON TO THE NEXT PAGE.

- (a) Write the PondStocker method numUnder. Method numUnder returns the smallest number of fish that must be added to make the density of fish in the environment greater than minDensity. If the density of fish in the environment is already greater than minDensity, then numUnder returns zero. Recall that the Environment methods numRows and numCols return the number of rows and the number of columns, respectively, in an environment.

Complete method numUnder below.

```
// postcondition: returns the minimum number of fish that need to be
//                added to make the population density greater than
//                minDensity
private int numUnder()
```

```
private int numUnder()
{
    int moreFish = 0;
    int cols = Environment.numCols;
    int rows = Environment.numRows;
    int size = cols * rows;
    int neededDens = minDensity * size;
    if (Environment.allObjects() <= minDensity)
    {
        moreFish = neededDens - Environment.allObjects();
    }
    return moreFish;
}
```

Part (b) begins on page 14.

GO ON TO THE NEXT PAGE.

(b) Write the PondStocker method randomLocation. Method randomLocation returns a random location within the bounds of the environment. C₂

In writing randomLocation, you may use any of the accessible methods of the classes in the case study. Solutions that reimplement functionality provided by these methods, rather than invoking these methods, will not receive full credit.

Complete method randomLocation below.

```
// postcondition: returns a random location within the bounds of theEnv  
private Location randomLocation()
```

```
private Location randomLocation()
```

```
{
```

```
    int cols = Environment.numCols();
```

```
    int rows = Environment.numRows();
```

```
    x = Random.nextInt(cols + 1);
```

```
    y = Random.nextInt(rows + 1);
```

```
    return new Location(x, y);
```

```
}
```

GO ON TO THE NEXT PAGE.

C3

- (c) Write the PondStocker method addFish. Method addFish adds numToAdd Fish to the environment at random locations that are not already occupied. You may use the two-parameter Fish constructor, so that the fish added have a random direction and color.

In writing addFish, you may call randomLocation. Assume that randomLocation works as specified, regardless of what you wrote in part (b). You may also use any of the accessible methods of the classes in the case study. Solutions that reimplement functionality provided by these methods, rather than invoking these methods, will not receive full credit.

Complete method addFish below.

```
// precondition: 0 <= numToAdd <= number of empty locations in theEnv
// postcondition: the number of fish in theEnv has been increased
//                by numToAdd; the fish added are placed at
//                random empty locations in theEnv
public void addFish(int numToAdd)
```

```
public void addFish(int numToAdd)
{
  for(x=0; x < numToAdd; x++)
  {
    (Environment.add(Fish), randomLocation);
  }
}
```

GO ON TO THE NEXT PAGE.