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.
(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 \( \text{minDensity} \). If the density of fish in the environment is already greater than \( \text{minDensity} \), 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.

```java
// postcondition: returns the minimum number of fish that need to be
// added to make the population density greater than
// \( \text{minDensity} \)
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.
(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.

```java
// postcondition: returns a random location within the bounds of the Env
private Location randomLocation()
{
    Random randNumGen = Random Num Generator.getInstance();
    return new Location(randNumGen.nextInt(Env.numRows()),
                         randNumGen.nextInt(Env.numCols()));
}
```
(c) Write the `PondStock` 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.

```java
// 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.isEmpty(nextLocation))
        {
            nextLocation = randomLocation();
        }
        Fish current = new Fish(theEnv, nextLocation);
    }
}
```

GO ON TO THE NEXT PAGE.
(a) Write the PondStock 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.

```java
// 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 = theEnv.numRows() * theEnv.numCols();
    List<愕> theFishes = theEnv.allObjects();
    int numFish = theFishes.length;
    double currentDensity = (double) numFish / base;
    while (currentDensity < minDensity) {
        numFish++;
        currentDensity = (double) numFish / base;
    }
    int numUnder = numFish - theFishes.length;
    return numUnder;
}
```

Part (b) begins on page 14.
(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 rand = new Random();  
    int randRow = rand.nextInt (theEnv.numRows);  
    int randCol = rand.nextInt (theEnv.numCols);  
    Location randomLocation = new Location (randRow, randCol);  
    if (is Valid (randomLocation) == true)  
        return randomLocation;  
    else  
        return randomLocation;  
}
(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.

```java
// 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 (!randomLocation.isEmpty()) {
            theEnv.add(new Fish(theEnv, randomLocation, theEnv.randomDirection),
                        randomColor());
        }
    }
}
```
(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.

```java
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().size() <= minDensity) {
        moreFish = neededDens - Environment.allObjects().size();
    }
    return moreFish;
}
```

Part (b) begins on page 14.
(b) Write the PondStock 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.

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

    int cols = Environment.numCols();
    int rows = Environment.numRows();
    x = Random.nextInt(cols + 1);
    y = Random.nextInt(rows + 1);
    return x, y;
```

GO ON TO THE NEXT PAGE.
(c) Write the PondStock 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.

```java
// 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++)
    {
        (Environment, add(Fish), randomLocation());
    }
}
```

Go on to the next page.