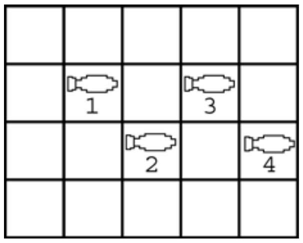
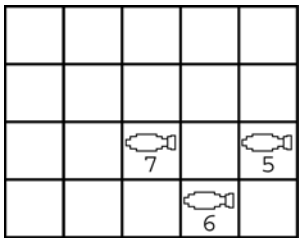
3. This question involves reasoning about the code from the Marine Biology Simulation case study. A copy of the code is provided as part of this exam.

   Consider defining a new type of fish called `ZigZagFish`, which moves in a zigzag pattern. The first time a `ZigZagFish` moves, it will move to the location forward and to the right if that cell is empty. This is illustrated in the figure below as the move from position 1 to position 2. In each subsequent move, the `ZigZagFish` will attempt to move to the forward diagonal location on the side opposite its previous move (the second move will be to the left forward diagonal, the third move will be to the right forward diagonal, and so on). When the `ZigZagFish` has successfully moved, its direction does not change. If the `ZigZagFish` is unable to move, it stays in the same location but reverses its direction. After reversing its direction, the next time the `ZigZagFish` moves, it will attempt to move in the same diagonal direction as it tried before reversing. The diagrams below show the path followed by a single `ZigZagFish` object as a result of multiple moves.



   Now consider what happens when the `ZigZagFish` attempts to move forward diagonally to the left from position 4. This move is blocked, and consequently the `ZigZagFish` stays in the same location but reverses its direction. This is illustrated as position 5 in the diagram below. From position 5, the `ZigZagFish` moves forward diagonally to the left, and from position 6, it moves forward diagonally to the right to position 7.



GO ON TO THE NEXT PAGE.

The `ZigZagFish` class is defined by extending the `Fish` class and overriding the `move` and `nextLocation` methods. Because a `ZigZagFish` alternates its pattern of movement, a private instance variable `willZigRight` keeps track of the direction of the next movement.

The partial declaration for class `ZigZagFish` is shown below.

```
public class ZigZagFish extends Fish
{
  private boolean willZigRight;
    // true indicates the next move should be forward to the right
    // false indicates the next move should be forward to the left


  public ZigZagFish(Environment env, Location loc)
  {
    super(env, loc);
    willZigRight = true; // direction of the next move
  }


  // returns the forward diagonal cell to the left or right of this fish
  // (depending on willZigRight) if that cell is empty;
  // otherwise, returns this fish's current location
  // postcondition: the state of this ZigZagFish is unchanged
  protected Location nextLocation()
  {  /* to be implemented in part (a) */  }


  // moves this ZigZagFish diagonally (as specified in nextLocation) if
  // possible; otherwise, reverses direction without moving;
  // after a diagonal move, willZigRight is updated
  protected void move()
  {  /* to be implemented in part (b) */  }


  // other constructors, generateChild, and other methods not shown
}
```

GO ON TO THE NEXT PAGE.

(a) Override the `nextLocation` method for the `ZigZagFish` class. The `nextLocation` method returns the cell diagonally forward to the right, the cell diagonally forward to the left, or the cell that the `ZigZagFish` currently occupies, according to the description given at the beginning of this question.

In writing `nextLocation`, you may use any of the accessible methods of the classes in the case study.

Complete method `nextLocation` below.

```
// returns the forward diagonal cell to the left or right of this fish
// (depending on willZigRight) if that cell is empty;
// otherwise, returns this fish's current location
// postcondition: the state of this ZigZagFish is unchanged
protected Location nextLocation()
```

(b) Override the `move` method for the `ZigZagFish` class. This method should change the location and direction of the fish as needed, according to the rules of movement described at the beginning of the question. In addition, the state of the fish must be updated.

In writing `move`, you may call `nextLocation`. Assume that `nextLocation` works as specified, regardless of what you wrote in part (a). You may also use any of the accessible methods of the classes in the case study.

Complete method `move` below.

```
// moves this ZigZagFish diagonally (as specified in nextLocation) if
// possible; otherwise, reverses direction without moving;
// after a diagonal move, willZigRight is updated
protected void move()
```

GO ON TO THE NEXT PAGE.