2. 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.

   A `PounceFish` is a type of fish that looks for prey and then "pounces" on it. A `PounceFish` can see only a limited distance in its forward direction. If the `PounceFish` sees another fish, it rushes forward and eats the nearest one that it sees, ending up in the location where its prey was originally located. If the `PounceFish` does not see another fish, it acts as a `Fish`.

   The `PounceFish` class is shown below.

```
public class PounceFish extends Fish
{
  private int range;   // the distance that a PounceFish can see; range > 0


  /** Looks ahead range locations in current direction
   *  @return the nearest fish in that direction within range (if any);
   *          null if no such fish is found
   */
  private Fish findFish()
  {  /* to be implemented in part (a) */  }


  /** Acts for one step in the simulation
   */
  public void act()
  {  /* to be implemented in part (b) */  }


  // There may be fields, constructors, and methods that are not shown.
}
```

The following diagrams show an example environment containing a `PounceFish` (represented by `P`) and other fish (represented by `F1`, `F2`, etc.). The direction of the `PounceFish` is indicated by the character `">"` showing that, in this example, the `PounceFish` is facing east. If the `PounceFish` can see 2 or more locations ahead in its forward direction, it will see fish `F3` as shown in the first diagram and will move to that location to eat it, causing `F3` to die as shown in the second diagram.

Environment before the `PounceFish` acts

North

		0	1	2	3	4	5	
	0			F1				
West	1	F2	P>		F3	F4		**East**
	2		F5					
	3							

South

Environment after the `PounceFish` acts

North

		0	1	2	3	4	5	
	0			F1				
West	1	F2			P>	F4		**East**
	2		F5					
	3							

South

If the `PounceFish` in the first diagram above could see only 1 location ahead, it would not see any prey and therefore would act as an ordinary fish.

AP® COMPUTER SCIENCE A
2007 SCORING GUIDELINES

Question 2: Pounce Fish (MBS)

Part A: findFish **5 points**

+2 access & check neighbor
- **+1/2** determine current location
- **+1/2** determine current direction
- **+1/2** correctly access any neighbor
- **+1/2** determine if neighbor location is empty

+1 1/2 loop in forward direction
- **+1/2** loop with respect to range
- **+1** access up to `range` consecutive forward locations (as needed)

+ 1 1/2 return value
- **+1/2** return null if reach invalid location in loop
- **+1/2** return object at first non-empty location in loop
- **+1/2** return null if no non-empty location in loop

Special Usage:

-1 missing or incorrect environment access

Part B: act **4 points**

+1/2 call `findFish()`

+1/2 test if `findFish` returned null

+2 not null case
- **+1** `prey.die()` or `environment().remove(prey)`
- **+1** change location to prey's location

+1 null case
- **+1/2** attempt to act (`move()` or `super.move()` OK)
- **+1/2** `super.act()`

AP® COMPUTER SCIENCE A
2007 CANONICAL SOLUTIONS

Question 2: Pounce Fish (MBS)

PART A:

```
private Fish findFish()
{
    Environment env = environment();
    Location loc = location();
    Direction dir = direction();

    for (int i = 0; i < range; i++) {
        loc = env.getNeighbor(loc, dir);
        if (!env.isEmpty(loc)) {
            return (Fish)env.objectAt(loc);
        }
    }
    return null;
}
```

PART B:

```
public void act()
{
    if (! isInEnv() )
        return;

    Fish prey = findFish();
    if (prey != null) {
        prey.die();             // OR environment().remove(prey);
        changeLocation(prey.location());
    }
    else {
        super.act();
    }
}
```