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.

(a) Write the `PounceFish` method `findFish`. If any fish are located within `range` locations in the direction that the `PounceFish` is currently facing, the method returns the nearest of these. Otherwise, the method returns `null`.

Complete method `findFish` below.

```
/** 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()
```

(b) Override the `act` method for the `PounceFish` class. A `PounceFish` attempts to find a fish that it can eat. If it finds such a fish, the `PounceFish` eats it (causing it to die) and moves to its location. If the `PounceFish` does not find a fish that it can eat, it acts as an ordinary fish.

In writing `act`, assume that `findFish` works as specified, regardless of what you wrote in part (a).

Complete method `act` below.

```
/** Acts for one step in the simulation
 */
public void act()
{
  if ( ! isInEnv() )
    return;

  // Write your code below.
```

3. Consider a system for processing student test scores. The following class will be used as part of this system and contains a student's name and the student's answers for a multiple-choice test. The answers are represented as strings of length one with an omitted answer being represented by a string containing a single question mark (`"?"`). These answers are stored in an `ArrayList` in which the position of the answer corresponds to the question number on the test (question numbers start at 0). A student's score on the test is computed by comparing the student's answers with the corresponding answers in the answer key for the test. One point is awarded for each correct answer and ¼ of a point is deducted for each incorrect answer. Omitted answers (indicated by `"?"`) do not change the student's score.

```
public class StudentAnswerSheet
{
  private ArrayList<String> answers;   // the list of the student's answers


  /** @param key the list of correct answers, represented as strings of length one
   *          Precondition: key.size() is equal to the number of answers in this answer sheet
   *  @return this student's test score
   */
  public double getScore(ArrayList<String> key)
  {  /* to be implemented in part (a) */  }


  /** @return the name of the student
   */
  public String getName()
  {  /* implementation not shown */  }

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