0 Objective and instructions

In Part A of this project you will apply what you have learned about arrays in Java.

In project 5, you were given the Ocean class that dealt with the behavior of, and interactions with, the ocean where seaweed, tuna, and sharks live. In this project, you will once again simulate the action in the ocean but you will concentrate on the Ocean class that deals with a 2-d array of object references. In Part A, we assume there is only one kind of creatures in the ocean, which is represented by the given class Creature.

Although this project looks similar to the previous one, please read the following description carefully as there are new classes and new methods, and some classes and methods with the same names as P5 work differently. (We use the same names because they have the same purpose but the implementation details are different.)

The entire program is organized as follows: The given class P6A contains the starting point and is in charge of setting up the needed ocean and creature data. The given class Controller is in charge of initializing the ocean (putting creatures in the ocean), calling the ocean’s simulation method, and displaying the ocean’s status. The given class Creature specifies the properties and behaviors of a generic creature type. Class Ocean is to be completed by you. The ocean is where all the action takes place, so you will have to write the methods that deal with the Ocean itself (e.g., getWidth, display) as well as the methods where a Creature interacts with the Ocean (e.g., put, move, simulate).

As you read the project description, read the provided code carefully. You should understand all the given code. (If you don’t understand, please ask for help!) Do not modify the given classes P6A, Controller, and Creature. In class Ocean you must complete the specified methods and you may add more fields and methods as necessary.

For Part A, you will submit your completed file Ocean.java.

1 P6A

Class P6A contains the main method that is the starting point of the program. To run your program, compile all the files and type the following in the Interactions pane:

    java P6A

Read the provided code: it creates an instance of the Ocean class and an array of Creatures and passes them to a new Controller object. The Controller object then calls its doSimulation method to start the whole simulation.

Note that your completed project (Part A) should work even if we change the data values in class P6A (e.g., different ocean size, different number of creatures, and/or different starting positions).

2 Controller

The Controller class initializes the ocean by putting the Creatures into the Ocean. It also controls the system—simulate for numRounds number of rounds. Unlike P5, however, class Controller doesn’t contain lengthy code for the actual simulation. Instead, in each round Controller simply calls the simulate method of the Ocean object to make one simulation round of the entire ocean and then calls the display method of the Ocean object to display the result after the current round.

Class Controller has the following two public methods:

- **public Controller(Ocean ocean, Creature[] creatures)**

  This is the constructor, which puts all the Creatures in array creatures in the ocean according to the coordinates stored in the fields of the individual Creatures.
(Hint! Pay close attention to the method call to `put`, a method that you will write later in class Ocean. Notice that `put` needs to `get` the coordinates of the `Creature` which is stored in the fields of the `Creature`. This is different from method `put` in P5, which `sets` the values of the position fields in the creature. This is just a preview of the specification for method `put`. More later.)

- `public void doSimulation(int numRounds)`
  This method calls the `simulate` and `display` methods of the Ocean `numRounds` times. Where is the code that makes sure that a `Creature` doesn’t get to move more than once in a round? You will take care of it when you write the `simulate` method in class Ocean!

### 3 Ocean

This is the class that you need to complete in Part A. After writing a method, remember to compile and test the program! This helps you isolate errors. Careful, incremental development of your class will end up saving you time.

As you saw in P5, the ocean is a table of values corresponding to the type of creatures that live at specific coordinates. Therefore, class Ocean needs a field that references a 2-d array of non-primitive type. (Which type? Think about what lives in every cell of the array.) The Ocean has boundaries, so class Ocean also needs fields for the dimensions. Finally, during simulation the Ocean needs to know for each position whether the Creature living there is ready to move, so a 2-d array that stores the individual states would be useful.

The Ocean class has the following public methods:

- `public Ocean(int height, int width)`
  This constructor sets the size of the ocean and creates the array(s) for the ocean.

- `public int getHeight()`
- `public int getWidth()`
  Getter methods for the ocean.

- `public Creature getCreature(int x, int y)`
  This method returns the `Creature` reference at position $(x, y)$ of the ocean. If the coordinate is invalid or there is no creature at that position, this method should return `null`.

- `public void put(int x, int y, Creature creature)`
  This method puts `creature` at position $(x, y)$ of the ocean. Note that this method only affects the Ocean object itself and it does not change anything in the Creature object. This is different from P5.

  Here in P6 Part A, a `Creature` is responsible for “knowing” its own position, i.e., the `Creature`’s position fields are set to the correct values before a call to the `put` method. See method `simulate` in class Creature.

- `public void kill(int x, int y)`
  This method kills (or more appropriately, makes disappear) the `Creature` at position $(x, y)$ of the ocean. The method assumes that $(x, y)$ is valid and that there is a `Creature` there.

- `public void move(int x1, int y1, int x2, int y2)`
  This method moves the creature at position $(x1, y1)$ to position $(x2, y2)$ of the ocean. Like method `put`, this method only affects the Ocean object itself and does not change the values in the `Creature`. I.e., the `Creature` itself is responsible for setting its position fields to the correct values after a move. See method `simulate` in class `Creature`.

  Again, you may assume that both positions are valid, that there is a `Creature` at $(x1, y1)$, and that $(x2, y2)$ is empty.

*The three methods `put`, `kill`, and `move` should make use of the `toString` method in a `Creature` to display a message about what is going on.*
• public void simulate()
  This method does a round of simulation for the entire ocean. The order in which the Creatures get their turns is the same as that in P5.
  Make sure that a Creature gets exactly one turn in every round, and don’t let a newly created Creature have a turn in the same round in which it is created.

• public void display()
  This method displays the current ocean. For a position where there is no creature, this method displays a squiggle (i.e., the character ‘~’); otherwise, it calls the creature’s abbreviation method to get the correct character to display.

• public boolean isValid(int x, int y)
  This method checks whether coordinate (x, y) is valid, i.e. inside the ocean. Remember that valid x coordinates range from 0 to width – 1, and valid y coordinates range from 0 to height – 1.

You may not add, remove, or change the meaning of any public method described above. However, feel free to add your own private or protected methods, if needed.

4 Creature

The Creature class represents a generic creature that will live in an ocean. Class Creature has a field myOcean that references the Ocean in which a Creature lives and the coordinate fields x and y for storing its position in the ocean. Class Creature has the following public methods:

• public Creature(Ocean ocean)
  This constructor takes a reference to an Ocean, where the Creature will reside.

• public void setPosition(int x, int y)
  This method sets the coordinates fields so that the Creature itself knows where it is in the ocean. This method does not change anything in the Ocean object.

• public int getX()
  public int getY()
  Getter methods for the coordinate fields

• public String toString()
  This method returns a short String description of the Creature and its location. For example, if the Creature is currently at position (3, 2), this method returns a String “Creature (3, 2)”.

• public char abbreviation()
  This method returns the character ‘C’, representing Creature. This method is used by the display method of Ocean.

• public void simulate()
  This method simulates the Creature’s action in one round. Our generic Creature may do one of the following three things: (1) move to a random position (not necessarily an adjacent one) in the sea, (2) produce a Creature offspring in a random position (again, not necessarily an adjacent one), or (3) just stay still. Of course, it must move to or produce an offspring at a valid position.