Important Notes

- **This assignment is to be done on your own.** If you need help, see the instructor or TA.
- Please start the assignment as soon as possible and get your questions answered early.
- Read through this specification completely before you start.
- Some aspects of this specification are subject to change, in response to issues detected by students or the course staff.

1 Description

The goal of this assignment is to write a Java program to simulate an ecosystem that consists of bears and fish, living in a river. The simulation proceeds in cycles, at each of which all animals age, and a subset of them move, mate, or die, according to certain rules. Next, we explain how the river and its inhabitants must be modeled. The river ecosystem will be implemented via the River class, whose first few lines look something like this:

```
public class River {
  ...
  public Animal[] river;
  ...
}
```

The instance variable `river` represents a river as a one-dimensional array, whose cells are references to objects belonging to the abstract class Animal, which represent life forms (bears or fish)[1]. A cell might also be `null`, meaning that it contains neither a bear nor a fish.

`Animal` is an abstract class. Each `Animal` object has a `gender` field and an `age` field:

```
public abstract class Animal {
  ...
  protected enum Gender {
      FEMALE, MALE
    }

  protected Gender gender;
  protected int age;
  ...
}
```

The `Animal` class has two concrete subclasses:

[1]Observe that river is declared to be public instead of private. While proper object-oriented design would dictate the latter, we choose the former for convenience in testing your programs. We will follow this approach for many, if not most, of the methods and instance variables that we require you to implement. Of course, in the job world, you will be expected to practice encapsulation, so implementation details such as river should be kept private.

- The `Fish` class models fish. A fish lives at most up to the end of its fifth year (six simulation cycles), unless it is killed earlier.
- The `Bear` class models bears. A bear lives until the end of its 9th year (10 simulation cycles), unless it is killed earlier. A bear has a *strength* that varies with age, as follows.

Age	0	1	2	3	4	5	6	7	8	9
Strength	1	2	3	4	5	4	3	2	1	0

The initial configuration of the river can be generated at random. After that, the simulation proceeds in cycles; the number of cycles is also part of the input. In each cycle, all the animals in the river age by one year, and those that exceed their allotted time span die (i.e., disappear). After that, the cells of the river are considered from left to right, starting at cell 0. If the cell contains a bear or a fish, this animal will randomly choose to either stay where it is or to attempt to move into an adjacent array cell to its left or right.

1.1 Rules for a Move

1. If the cell to which an animal wants to move is empty, simply move that animal to the new cell.
2. If a bear and a fish collide, then the fish dies, regardless of the animals' genders.
3. If two animals of the same species and gender are about to collide in the same cell, then:
   (a) If the animals are fish, they stay where they are, and nothing else happens.
   (b) If the animals are bears of the same gender and of the same strength, they stay where they are, and nothing else happens.
   (c) If the animals are bears of the same gender and of different strengths, then the animal of lesser strength dies.
4. If two animals of the same type but different gender are about to collide in the same cell, they stay where they are, but they create a new instance of that type of animal, which is placed in a random empty (i.e., previously `null`) cell in the array. If the array is full, no new animal is created.

1.2 Notes

- During each cycle, before animals are processed, they all age by one year.
- Although multiple animals may move in one cycle, they are processed **one at a time**, from left to right, so that simultaneous moves do not occur. For instance, suppose we have two adjacent female bears of ages 1 and 2, represented by the strings "BF1" and "BF2", as shown below.

. . . . . BF1 BF2 . . . . . .

The simulation should consider BF1 before BF2. If BF1 moves right, it will be killed by BF2 before BF2 is even considered for a move. Similarly, if BF1 stays put, but BF2 moves to the left, then BF2 will kill BF1. Note that, if we had allowed simultaneous moves (but, remember, we do not), BF1 and BF2 could have jumped over each other, avoiding death.

- If an animal, say BF2 at river[2], moves to the right (at river[3]) and is not killed, then since it has been processed, it should not be processed at the current position again. That is, after this cycle, you should see that BF2 stays at river[3], rather than being processed with another move. As another example, if the length of the river is 9, and an animal FM2 at river[0] moves successfully to its left (river[8]), then river[8] should not be processed again in the same cycle.
- To model the birth of a new animal, use object instantiation, via the new operator. Note that there may be two new animals born by the same parents in one cycle. For example, assuming that river[2] is BF2 and river[3] is BM4, when BF2 is processed, it decides to move to the right, and when BM4 is processed, it decides to move to the left. Each time, although both BF2 and BM4 stay where they are, a new bear will be born.
- When an `Animal` object is instantiated as a result of rule 4, its age is zero and its gender is chosen at random.
- A move of an `Animal` object is modeled by updating pointer references. E.g., to move a Fish object from cell 8 to cell 9, we make cell 9 point to this object, and set cell 8 to `null`.
- The death of an animal is modeled by removing the reference to it; either by making it `null`, or making it point to some other object.
- The river should be treated as being circular; that is, its last and first cells should be treated as being adjacent. For instance, suppose array `river` is of length 12, that `river[11]` references a `Bear` object, and that `river[0]` references a `Fish` object. Then, if the bear decides to move to the right, it will kill the fish, leaving the bear in cell 0, and leaving cell 11 referencing `null`.

2 Tasks

Your job is to implement the `Animal` class, along with its `Bear` and `Fish` subclasses, and the `River` class. The main method must be in the class `RiverSimulator`, which repeatedly simulates evolutions of river ecosystems. In each iteration, RiverSimulator does the following.

1. It asks for the length of the river. Then, for each cell, it puts a `Bear`, a `Fish`, or `null` uniformly at random (i.e., with equal probability). The gender and age of each animal are also chosen uniformly at random, taking care that the age of an animal is within the appropriate range (0 to 9 for a bear, 0 to 4 for a fish)[2]
2. It asks the user for the number of cycles to simulate. When zero or a negative number of cycles is entered by the user, your code does nothing but waits for a positive input.
3. It executes the required number of cycles, printing the river after each step.

The methods associated with the various classes are presented in Section 3.

2.1 Example

We represent a river by a string that looks like this:

— FF2 BM3 — BF1 BM8 FM0 — — BF4 (1)

[2]Generate uniformly-distributed random integers using `nextInt()`, from `java.util.Random`.

The above represents a `River` object whose underlying array has length 10. Each element is represented by a three-character string, which is either '—', which stands for `null`, or it has the form $\alpha\beta\gamma$, where

- α is either F or B, depending on whether the cell references a `Fish` or a `Bear`,
- β is either F or M, depending on whether the object referenced is male or female, and
- γ is a single digit, between 0 and 4 for `Fish` objects and between 0 and 9 for `Bear` objects, which gives the age of the object.

There is precisely one space separating each three-letter string.
Note. We require the `toString()` method of the `River` class to produce precisely this representation on a `River` object.

```
Welcome to CS206 River Ecosystem Simulator!

River Ecosystem Simulator
Please choose: 1 (random river) 2 (exit)
9
Invalid selection. Please try again.

River Ecosystem Simulator
Please choose: 1 (random river) 2 (exit)
-1
Invalid selection. Please try again.

River Ecosystem Simulator
Please choose: 1 (random river) 2 (exit)
1
Creating a random river...
Enter the river length (an integer bigger than 0):
10
Enter the number of cycles (an integer bigger than 0):
9
Initial river:
BM1 --- --- FM1 FM4 FF2 FM2 FM0 FF3 FM1
After cycle 1
FM2 BM2 --- --- FM2 FF3 FM3 FM1 FF4 ---
After cycle 2
BM3 --- FM0 --- FM3 FF4 FM4 FM2 FF5 ---
After cycle 3
BM4 --- FM1 --- --- FM4 --- FM3 FM1 ---
After cycle 4
--- FM2 --- --- --- FM5 FM4 --- --- BM5
After cycle 5
BM6 FM3 --- --- --- --- --- FM5 --- ---
After cycle 6
--- BM7 --- --- --- --- --- --- --- ---
After cycle 7
--- --- BM8 --- --- --- --- --- --- ---
After cycle 8
--- BM9 --- --- --- --- --- --- --- ---
After cycle 9
--- --- --- --- --- --- --- --- --- ---


River Ecosystem Simulator
Please choose: 1 (random river) 2 (exit)
1
```

```
Creating a random river...
Enter the river length (an integer bigger than 0):
12
Enter the number of cycles (an integer bigger than 0):
10
Initial river:
BF5 --- FF1 BF7 FF4 FM2 BM2 --- FF0 --- FM2 ---
After cycle 1
BF6 FF2 BF8 --- --- BM3 --- --- FF1 --- --- FM3
After cycle 2
--- BF7 --- BF9 BM4 --- --- --- FF2 --- --- FM4
After cycle 3
BF8 --- --- --- --- BM5 --- --- FF3 --- --- ---
After cycle 4
--- --- --- --- --- --- BM6 --- --- FF4 --- BF9
After cycle 5
--- --- --- --- --- --- BM7 --- FF5 --- --- ---
After cycle 6
--- --- --- --- --- --- --- BM8 --- --- --- ---
After cycle 7
--- --- --- --- --- --- --- --- BM9 --- --- ---
After cycle 8
--- --- --- --- --- --- --- --- --- --- --- ---
After cycle 9
--- --- --- --- --- --- --- --- --- --- --- ---
After cycle 10
--- --- --- --- --- --- --- --- --- --- --- ---

River Ecosystem Simulator
Please choose: 1 (random river) 2 (exit)
1
Creating a random river...
Enter the river length (an integer bigger than 0):
15
Enter the number of cycles (an integer bigger than 0):
9
Initial river:
FM3 FM0 BF4 BF8 FF3 --- --- BM3 BM4 BF2 FF1 FF3 BF0 FM2 FM0
After cycle 1
FM4 FM1 --- BF5 --- FF4 BM4 --- BM5 BF3 FF2 BF1 FM3 FM1 ---
After cycle 2
FM2 --- --- --- BF6 --- BM5 BM6 BF4 --- --- BF2 FM4 FM2 ---
After cycle 3
FM3 --- --- --- BF7 BM6 BF0 BM7 --- BF5 --- BF3 --- --- FM3
After cycle 4
FM4 --- --- BF8 BM7 BF1 BM8 --- BF6 --- BF4 --- --- FM4 ---
After cycle 5
--- BM0 BM0 BF9 BM8 BF2 BM9 BM0 BF7 --- BF5 BM0 --- BM0 ---
After cycle 6
--- BM1 BM1 --- BM9 BF3 BM1 --- --- BF8 BF6 BM1 BM1 --- ---
After cycle 7
--- BM2 BF0 BM2 --- BF4 --- BM2 BF0 --- BF7 BM2 --- BM2 ---
After cycle 8
BM3 --- BF1 BM3 BF5 --- BM3 BF1 --- BM0 BF8 --- BM3 BM3 ---
After cycle 9
--- BF2 BM0 BM4 --- BF6 BM4 BF2 --- BM1 --- BF9 BM4 BM4 BM4

River Ecosystem Simulator
Please choose: 1 (random river) 2 (exit)
```

```
2
Goodbye !
```

Your code should print out the same text messages to engage user interactions.

3 Methods

We now list the methods that you are required to implement for each class. Of course, you may also define additional helper methods of your own.

3.1 The `Animal` Class

`public Animal()`
Create an animal of a random age and gender.

`public Animal(int age, Gender gender)`
Create an animal of the specified age and gender.

`public int getAge()`
Get the age of the animal.

`abstract boolean maxAge();`
Returns `true` if the current age of the animal has reached the limit for the species; otherwise, it returns `false`.

`abstract boolean incrAge();`
If the current age of the animal is less than the maximum for the species, increments the age of the animal by one and returns `true`.
Otherwise, it leaves the age as is and returns `false`.

`public String toString()`
This method overrides the toString() method and it returns, e.g., “BF7” for a 7-year old female bear.

3.2 The `Fish` Class

This extends the `Animal` class, providing appropriate implementations of the constructors, `maxAge()`, and `incrAge()`.

3.3 The `Bear` Class

This extends the `Animal` class, providing appropriate implementations of the constructors, `maxAge()`, and `incrAge()`. Additionally, it offers one new method:

`getStrength()`
Get the current strength of the bear.

3.4 The `River` Class

`public River(int length)`
Generates a random river ecosystem of the given length.

`public int getLength()`
Returns the length of the river.

`public int numEmpty()`
Returns the number of empty (null) cells in the river.

`public boolean addRandom(Animal animal)`
If the river has no empty (`null`) cells, then do nothing and return `false` .
Otherwise, add an animal of age 0 of randomly chosen gender and of the same type as animal to a cell chosen uniformly at random from among the currently empty cells and return `true` .

`public void updateCell(int i)`
Process the object at cell i, following the rules given in the Description.
If it is null, do nothing.
If it has reached the end of its lifespan, it dies.
Otherwise, decides which direction, if any, the animal should move, and what other actions to take (including the creation of a child), following the rules given in the Description.

`public void updateRiver()`
Perform one cycle of the simulation, going through the cells of the river, updating ages, moving animals, creating animals, and killing animals, as explained in the Description.

`public String toString()`
Produce a string representation of the river.

4 Rules for Making Random Choices

- When choosing what to put in a given cell of a random river ecosystem, generate a random integer k between 0 and 2. If $k = 0$, put a `null` ; if $k = 1$, then put a `Bear` ; if $k = 2$, then put a `Fish` .
- To choose the *gender* of an animal at random, generate a random integer k between 0 and 1. If $k = 0$, then the gender is MALE; if $k = 1$, then the gender is FEMALE.
- To choose the *age* of an animal of a given species at random, generate a random integer k between 0 and the maximum age for that species. Set the age of the animal to k.
- To choose between *no move*, a *left move*, or a *right move*, generate a random integer k between 0 and 2. If $k = 0$, don't move; if $k = 1$, move left; if $k = 2$, move right.

What to turn in:

Inside your project folder, create a pure text file called README.txt. You can install Sublime Text software for writing this file. In this file,

- write down any problems that you encountered and how you solved these problems;
- list any functionalities that you were not able to successfully implement;
- your experience of working on this problem, what did you learn?

Compress your project folder. Turn in a zip file named `Firstname_Lastname_P1.zip` containing all your source code (and the README file as well) for the project. Include the Javadoc tag @author in each class source file, along with the other expected Javadoc tags.

Grading:

- Program compiles: required for a grade
- Poorly organized code: up to −2 points
- Poor comments: up to −2 points
  Make sure that your projects are commented properly. This includes file headers, method headers, and appropriate inline comments.
- README file: 1 point
- Completeness of the classes' unit-testing: 2 points
- Correctness: 7 points
  - All methods and variables in a class are correctly defined with appropriate modifiers.
  - The application logic is clean and clear.
  - Use `Scanner` class correctly.
  - Inheritance is correctly implemented.
  - The output format of the application is exactly as it shows in the example run.
  - The implementation should satisfy all the requirements described in the project.