Computer Science 228
Spring 2013
Project 1: Bears and Fish
Part 1 (50 pints): Due at 11:59 pm, Friday, February 1
Part 2 (150) points: Due at 11:59 pm, Friday, February 8

Important Notes

- ***This assignment is to be done on your own.*** If you need help, see one of the instructors or the TAs. Please make sure you understand the "Academic Dishonesty Policy" section of the syllabus.
- 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. ***Check Blackboard regularly for updates and clarifications.***

First posted: January 24 **Last updated:** 5 pm, January 26 (see p. 9)

1 Project 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[1]:

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

[1] More details of this and other classes will be provided in the templates that will accompany this project.

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).[2] 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:

- The `Fish` class models fish. A fish lives at most up to the end of its fourth year (five 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

As explained later, the initial configuration of the river can be generated at random or it can be provided by the user. 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. The rules for a move are as follows.

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.

[2]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.

(c) If the animals are 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.

Notes.

- 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.
- To model the birth of a new animal, use object instantiation, via the `new` operator.
- 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 the user how initial river ecosystem will be generated. There are two options.

   Random: In this case, 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).[3]

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

File: In this case, it asks for a file name and then reads the input from a file. The format of the input will be described below.

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.

Task for Part 1: Write and submit JUnit tests for all the classes required for this project.

Task for Part 2: Provide working code for all the classes and method required for this project.

Example

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

```
--- FF2 BM3 --- BF1 BM8 FM0 --- --- BF4
```
(1)

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
- δ 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.

```
River Ecosystem Simulator
keys: 1 (random river) 2 (file input) 3 (exit)

Trial 1: 1
Random river
Enter river length: 8
Enter the number of cycles: 1

Initial river:

BF3 FF4 --- --- BM2 BF9 --- FM3

After cycle 1:
```

```
FM4 BF4 --- BF0 BM3 --- --- ---

Trial 2: 1
Random river
Enter river length: 11
Enter the number of cycles: 3

Initial river:

--- FF1 FM3 --- --- --- BF3 --- --- BF2 ---

After cycle 1:

--- FF2 FM4 --- FM0 --- --- BF4 --- BF3 ---

After cycle 2:

FF3 --- --- --- FM1 --- --- --- BF5 BF4 ---

After cycle 3:

--- --- --- --- FN2 --- --- --- --- BF6 FF4

 . . .

Trial 4: 3
```

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

Note. We require the `toString()` method of the `River` class to produce precisely this representation on a `River` object.

3 Methods

We now list the methods that you are required to implement for each class.

Important

Since a considerable part of the testing is automated, you ***must*** follow the specifications given here ***exactly***. This means, among other things, that the names and types of classes, instance variables, and methods must be kept ***as is***. Also, for testing, all classes must be declared `public`.

Of course, you may also define additional helper methods of your own.

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`.

Animals should also have `toString()` method, which returns, e.g., '`BF7`' for a 7-year old female bear.

The `Fish` Class

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

The `Bear` Class

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

`public int getStrength()`
: Get the current strength of the bear.

The `River` Class

Note that some of the methods below allow you to set the seed of the random number generator; this can useful for unit testing, as it makes it easier to replicate behaviors.

`public River(String inputFileName) throws FileNotFoundException`
: Construct a river from a file. The format is as described in page 4.

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

`public River(int length, long seed)`
: Generates a random river ecosystem of the given length, where the seed of the random number generator is as given.

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

`public void setSeed(long seed)`
: Sets the seed of the random number generator used in updating 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 Project Description. If it is `null`, do nothing. If it is an animal and it has reached the end of its lifespan, the animal 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 Project 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 Project Description.

`public void write(String outputFileName) throws FileNotFoundException`
Write the river to an output file.

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

4 Grading

Your grade for Part 1 will mainly be determined by running your JUnit test on some correct and incorrect versions of the various classes. The correct versions should pass your tests, and the incorrect versions should fail appropriately.

A significant portion of your grade for Part 2 will be based on running our unit tests on your classes. Documentation and style will count for roughly 15% of the total points.

5 Submission

All submissions must be done through Blackboard Learn. Please follow the guidelines posted under "Documents & Links".

Part 1. Submit a file containing all your JUnit test classes.

Part 2. Turn in a zip file named `Firstname_Lastname_HW1.zip` containing all your source code. The package name for the project must be `edu.iastate.cs228.hw1`. Include the Javadoc tag `@author` in each class source file. ***Do not turn in class files.***

Late Penalties

Asssignments may be submitted up to 24 hours after the the deadline with a 25% penalty (not counting weekends or holidays). No credit will be given for assignments submitted after that time.

Updates (as of January 26)

- The sample output format has been corrected and modified slightly.
- Templates have been provided for all classes.
- The `getAge()` method is now to be implemented within the `Animal` class. This is because the same method is used for `Bear` and `Fish`.
- For testing purposes, the `getAge()`, `maxAge()`, and `incrAge()` methods are now `public`.
- Along with the templates mentioned earlier, we have provided a `RandomSingleton` class, which you ***must*** use to generate random numbers. This new class will enable the TAs to set the seed for the random number generator as needed for testing. ***Do not*** use any other class to generate random numbers.
- Here are some 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.