0. Objective

Completing all tasks in this assignment will help you learn about:

- Inheritance
- Object Oriented Programming
- Simulation

First skim, and then carefully read the entire assignment before starting any tasks! You must use the identifiers (variable names, method names, etc.) exactly as we specify, including capitalization. Use good programming style and observe the style guidelines that are given in the previous grading guides.

1. Developing the Animal kingdom

For this assignment you will be developing an Animal kingdom. In our make-belief Animal kingdom, there are only two types of Animals: Predator and Prey. Both Predator and Prey are Animals but each has its own special features also.

The Animal class is already provided to you. Your job in this assignment will be to write the two sub-classes (Predator and Prey) and then to write a simple simulator that shows an interaction between these animals.

It will benefit you to study the parent class and to become familiar with its variables and methods. Notice that every Animal will have six attributes. The class constant BENEFIT_RATE is the general rate at which all Animals get stronger, faster, healthier, etc. Always multiply or divide by BENEFIT_RATE, never add or subtract.

*All Animals start off with the same fitness level of 7 (class constant initFITNESS).

**If an Animal’s fitness ever drops below 5 (class constant minFITNESS) or if it ever starves more than 2 (class constant maxSTARVATION) times in a row, then the Animal dies. As you write methods in the subclasses, always check the Animal’s fitness after its value has been decreased to determine whether the Animal will die.

***The constructor for Animal has a String and an int parameters. The String will be the name of the Animal while the integer indicates the type of the animal: 1 for Predators (class constant aPREDATOR), 2 for Prey (class constant aPREY).

2. Predator

Write this subclass exactly to specification. Predators don’t have any additional fields, but there are four methods and a constructor that you must write. Whenever possible, use the methods and class constants from the parent class.

Predator(String name) – this is the only Predator constructor. Assign to both strength and speed random values between 5 and 15.

public void hunt(Animal target) – a Predator will call the hunt method on another Animal when it is trying to feed. There are rules that you must follow when you write this method. Formulate your <if-else> statements carefully—think about it before you write five <if> statements in a row. A Predator will only feed or starve once in a call of hunt.

1) If Animal target is dead, then the Predator has nothing to hunt so it will starve (call the starve method)
2) If Animal target is another Predator, then the two Predators will fight each other (call the fight method, described below). If the Predator wins, then it eats, otherwise it starves. This Predator wins if its toughness is at least as high as the target.

3) If the target is a Prey and it is faster than the Predator, then the Predator will fail the hunt and it will starve.

4) If the target is a Prey and it is slower but stronger than the Predator, then the Predator will not be able to catch the Prey (it will starve), but the Predator will injure the Prey (see injure method described below).

5) If the target is a Prey and the Predator is faster and stronger than the Prey, it will have a successful hunt and be able to call method eat.

public void eat() – notice that the Predator’s eat method is overriding the parent’s eat method. The Predator’s eat method has the same effect as the parent’s eat method except that it also increases the strength of the Predator by the BENEFIT_RATE (new strength is old strength * BENEFIT_RATE)

private int fight(Predator enemy) – fight method called during a hunt (if the target is another Predator). Both Predators suffer a penalty to their fitness. If the enemy is tougher than the current Predator, then the current Predator gets injured (call the injure method) and the method returns a value indicating a loss (Animal class constant LOST). However, if the enemy is not as tough, the current Predator will injure the enemy and the method returns a value indicating a victory (Animal class constant WON). Note: Since the Predators’ fitness decrease, both may die after the fight. Method fight will return that the current (dead) Predator wins, which works for the rest of the program since the dead Predator’s fitness will be zero.

private void injure(Animal target) – The parameter target is an Animal (may be either Predator or Prey). The effects of getting injured are: 1) a penalty to the target’s strength, speed, and fitness by the BENEFIT_RATE. Remember to check if an Animal’s fitness is below minFITNESS after it gets injured.

3. Prey

The Prey class is very similar to the Predator class. Once you have implemented either of these classes, implementing the other shouldn’t be difficult. The Prey class has four parts that you need to complete. Whenever possible, use the methods and class constants from the parent class.

Prey(String name) – this is the only Prey constructor. Assign to the Prey’s strength field a random number (type double) between 5 and 10 and the speed field a random number (type double) between 7 and 12.

public void graze(Animal target) – a Prey will call method graze on Animal target in order to feed. This method doesn’t mean that the Prey will eat the target, rather, it means that the Prey will invade the target’s territory in search for food. Make sure your implementation follows the guidelines below, and again formulate the <if-else> statements carefully. The Prey eats or starves only once in a call of graze.

1) If Animal target is dead, then the Prey can graze all it wants so it gets to eat.
2) If Animal target is a Predator, then the Prey runs away and starves for this round.
3) If Animal target is a Prey, then the Prey fights the other Prey. If this Prey wins, then it eats. Otherwise, it starves.

public void eat() – notice that the Prey’s eat method overrides the parent’s eat method. The Prey’s eat method has the same effect as the parent’s eat method except that it also increases the speed of the Prey by the BENEFIT_RATE (new speed is old speed * BENEFIT_RATE)

private int fight(Prey enemy) – fight method is called during grazing (if the target is another Prey). Both Preys suffer a penalty to their fitness. If the target Prey is tougher than the current Prey, then
the current Prey loses the fight and the method returns a value indicating a loss (Animal class constant LOST). Otherwise, the target Prey is not as tough as the current Prey and the current Prey wins the fight and the method returns a value indicating a victory (Animal class constant WON).

Note: Since the Preys’ fitness decrease, both may die after the fight. Method fight will return that the current (dead) Prey wins, which works for the rest of the program since the dead Prey’s fitness will be zero.

4. Simulator

The final part to this assignment is to build a simple simulator. The Simulator class has a main method that has been started but not completed. Your goal is to populate an array of Animals with Predators and Preys. After you populate the “kingdom”, simulate interactions between the Animals in multiple cycles. In each cycle, each Animal either hunt or graze once. You need to pass in a target Animal as the argument for either method: select an Animal from the kingdom randomly and pass it in as the target. Do not re-define any of the given variables.

You also have to implement two methods in the Simulator class.

public static void printstate(Animal[] kingdom) – given the array kingdom, print a description of the Animals in the kingdom. Use this method to show the initial state of the kingdom, then call this method show the state after each cycle.

public static void printmasterofkingdom(Animal[] kingdom) – a method that determines the Animal with the highest toughness in the kingdom and then prints out that Animal’s name along with its toughness. If multiple Animals have the highest toughness value, choose any one of them as the master.

Aim to output something similar to the example below. In the example output, we generate a kingdom of 10 Animals with 5 Predators and 5 Preys and print the state of the kingdom. Notice that we name of each Animal based on its position in the kingdom array, without using any user input. We let each of the Animals in the kingdom interact for 2 cycles (in each cycle, all the alive Animals get to either hunt or graze, depending on what they are) and print the state of the kingdom after each cycle. The final line indicates shows the master of the kingdom.

Note that the output indicating the death of an Animal (e.g., “Predator x Has died.”) may appear more than once, since several methods check an Animal’s fitness and starvation_level (and report its death).

5. Example Output

> java Simulator

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Fitness</th>
<th>Strength</th>
<th>Speed</th>
<th>Toughness</th>
</tr>
</thead>
<tbody>
<tr>
<td>Predator</td>
<td>Predator0</td>
<td>7.00</td>
<td>9.91</td>
<td>9.54</td>
<td>68.07</td>
</tr>
<tr>
<td>Predator</td>
<td>Predator1</td>
<td>7.00</td>
<td>8.88</td>
<td>8.66</td>
<td>61.39</td>
</tr>
<tr>
<td>Predator</td>
<td>Predator2</td>
<td>7.00</td>
<td>6.38</td>
<td>8.67</td>
<td>52.69</td>
</tr>
<tr>
<td>Predator</td>
<td>Predator3</td>
<td>7.00</td>
<td>6.14</td>
<td>7.98</td>
<td>49.42</td>
</tr>
<tr>
<td>Predator</td>
<td>Predator4</td>
<td>7.00</td>
<td>8.86</td>
<td>7.31</td>
<td>56.58</td>
</tr>
<tr>
<td>Prey</td>
<td>Prey5</td>
<td>7.00</td>
<td>7.17</td>
<td>8.70</td>
<td>55.57</td>
</tr>
<tr>
<td>Prey</td>
<td>Prey6</td>
<td>7.00</td>
<td>7.56</td>
<td>7.57</td>
<td>52.95</td>
</tr>
<tr>
<td>Prey</td>
<td>Prey7</td>
<td>7.00</td>
<td>9.92</td>
<td>7.36</td>
<td>60.47</td>
</tr>
<tr>
<td>Prey</td>
<td>Prey8</td>
<td>7.00</td>
<td>9.42</td>
<td>11.54</td>
<td>73.36</td>
</tr>
<tr>
<td>Prey</td>
<td>Prey9</td>
<td>7.00</td>
<td>6.84</td>
<td>10.31</td>
<td>60.01</td>
</tr>
<tr>
<td>Prey5 has died.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Predator</td>
<td>Predator0</td>
<td>7.70</td>
<td>10.90</td>
<td>9.54</td>
<td>78.69</td>
</tr>
<tr>
<td>Predator</td>
<td>Predator1</td>
<td>5.79</td>
<td>8.88</td>
<td>7.87</td>
<td>48.46</td>
</tr>
<tr>
<td>Predator</td>
<td>Predator2</td>
<td>5.26</td>
<td>5.80</td>
<td>7.89</td>
<td>35.99</td>
</tr>
<tr>
<td>Predator</td>
<td>Predator3</td>
<td>7.00</td>
<td>6.75</td>
<td>7.98</td>
<td>51.56</td>
</tr>
<tr>
<td>Predator</td>
<td>Predator4</td>
<td>5.79</td>
<td>8.86</td>
<td>6.65</td>
<td>44.84</td>
</tr>
<tr>
<td>Prey5 is dead</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Prey</td>
<td>Prey6</td>
<td>5.79</td>
<td>7.56</td>
<td>7.57</td>
<td>43.76</td>
</tr>
<tr>
<td>Prey</td>
<td>Prey7</td>
<td>5.79</td>
<td>9.02</td>
<td>7.36</td>
<td>47.37</td>
</tr>
<tr>
<td>Prey</td>
<td>Prey8</td>
<td>6.36</td>
<td>9.42</td>
<td>12.69</td>
<td>70.36</td>
</tr>
<tr>
<td>Prey</td>
<td>Prey9</td>
<td>7.00</td>
<td>6.84</td>
<td>11.34</td>
<td>63.62</td>
</tr>
<tr>
<td>Prey7 has died.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Predator1 has died.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Predator1 has died.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Predator2 has died.
Predator4 has died.
Prey6 has died.
Predator  Predator0  7.70  11.99  9.54  82.88
Predator1 is dead
Predator2 is dead
Predator  Predator3  7.00  7.43  7.98  53.93
Predator4 is dead
Prey5 is dead
Prey6 is dead
Prey7 is dead
Prey  Prey8  6.36  9.42  13.96  74.40
Prey  Prey9  7.70  6.84  12.47  74.35
Predator0 is the master of the kingdom with 82.8843127517529 toughness

6. What to submit

Submit your files Predator.java, Prey.java, and Simulator.java on-line using CMS (Course Management System) before the project deadline. Make sure you are submitting the correct, up to date, .java files (not .class or .java~). We will not accept any files after the deadline for any reason (except for documented medical reasons). See the CMS link on the web page for instructions on using CMS. If necessary, turn off the DrJava feature that saves the .java~ files (see course webpage announcement on 2/17 for instructions).