Assignment 10: Critters

reading: HW10 assignment spec
Critters

- A simulation world with animal objects with behavior:
  - fight: animal fighting
  - getColor: color to display
  - getMove: movement
  - toString: letter to display

- You must implement:
  - Lion
  - Tiger
  - Bear
  - Wolf (Creative)
A Critter class

class name implements Critter {
    ...
}

- `implements Critter` tells the simulator your class is a critter
  - an example of `interfaces`

- Must implement all of the methods specified by the interface

- `interface` is like a "to do list"
public interface Critter {

    // methods to be implemented
    public int fight(String opponent);
    public Color getColor();
    public int getMove(CritterInfo info);
    public String toString();

    // constants for directions
    public static final int NORTH = -2;
    public static final int SOUTH = 4;
    public static final int EAST = 3;
    public static final int WEST = 19;
    public static final int CENTER = 11;
}
Interface

• Declares methods that classes of that type must implement
• Classes that implement the interface must have the method, but how the method works can be very different
• A way of ensuring certain behaviors / methods of classes
How the simulator works

- When you press "Go", the simulator enters a loop:
  - move each animal once (getMove), in random order
  - if the animal has moved onto an occupied square, fight!

- Key concept: The simulator is in control, NOT your animal.
  - Example: getMove can return only one move at a time. 
    getMove can't use loops to return a sequence of moves.
    - It wouldn't be fair to let one animal make many moves in one turn!

- Your animal must keep state (as fields, instance variables) so that it can make a single move, and know what moves to make later.
Critter exercise: Stone

- Write a critter class Stone (the dumbest of all critters):

<table>
<thead>
<tr>
<th>Method</th>
<th>Behavior</th>
</tr>
</thead>
<tbody>
<tr>
<td>constructor</td>
<td>public Stone()</td>
</tr>
<tr>
<td>fight</td>
<td>Always ROCK</td>
</tr>
<tr>
<td>getColor</td>
<td>Always Color.GRAY</td>
</tr>
<tr>
<td>getMove</td>
<td>CENTER</td>
</tr>
<tr>
<td>toString</td>
<td>&quot;S&quot;</td>
</tr>
</tbody>
</table>
Ideas for state

• You must not only have the right state, but update that state properly when relevant actions occur.

• Counting is helpful:
  • How many total moves has this animal made?
  • How many times has it fought?

• Remembering recent actions in fields is helpful:
  • Which direction did the animal move last?
    • How many times has it moved that way?
Keeping state

How can a critter move west until it fights?

```java
public Direction getMove() {
    while (animal has not fought) {
        return Direction.EAST;
    }
    while (animal has not fought a second time) {
        return Direction.EAST;
    }
}
```

```java
private int moves; // total moves made by this Critter

public int getMove() {
    moves++;
    if (moves % 4 == 1 || moves % 4 == 2) {
        return WEST;
    } else {
        return EAST;
    }
}
```
Testing critters

- Focus on one specific critter of one specific type
  - Only spawn 1 of each animal, for debugging

- Make sure your fields update properly
  - Use `println` statements to see field values

- Look at the behavior one step at a time
  - Use "Tick" rather than "Go"
Critter exercise: Snake

<table>
<thead>
<tr>
<th>Method</th>
<th>Behavior</th>
</tr>
</thead>
<tbody>
<tr>
<td>constructor</td>
<td>public Snake()</td>
</tr>
<tr>
<td>fight</td>
<td>SCISSORS</td>
</tr>
<tr>
<td>getColor</td>
<td>black</td>
</tr>
<tr>
<td>getMove</td>
<td>1 E, 1 S; 2 W, 1 S; 3 E, 1 S; 4 W, 1 S; 5 E, ...</td>
</tr>
<tr>
<td>toString</td>
<td>&quot;K&quot;</td>
</tr>
</tbody>
</table>
Determining necessary fields

- Information required to decide what move to make?
  - Direction to go in
  - Length of current cycle
  - Number of moves made in current cycle

- Remembering things you've done in the past:
  - an int counter?
  - a boolean flag?