Assignment 11: Critters
Critters

• A simulation world with animal objects with behavior:
  • fight       animal fighting
  • getColor    color to display
  • getMove     movement
  • toString    letter to display
  • eat         eat food?

• You must implement:
  • Ant
  • Bird
  • Vulture
  • Hippo
  • Longhorn (Wins Overall and Creative)
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.
Scoring

• Score for each species:
  • For all animals of that species
  • Number of animals alive
  • Number of fights won
  • Pieces of food eaten
Food

• Simulator places food randomly around world
• Eating food increases score for species, but ...
• Critters sleep after eating
  • simulator (CritterMain) handles this
• A Critter that gets in a fight while sleeping always loses
  • simulator handles this
Mating

- Two Critters of same species next to each other mate and produce a baby Critter
- Simulator handles this
- Critters not asked if they want to mate
- Critters vulnerable while mating (heart graphic indicates mating)
  - automatically lose fight
- The Simulator handles all of this
  - You don't write any code to deal with mating
public abstract class Critter {
    public boolean eat() {
        return false;
    }
    public Attack fight(String opponent) {
        return Attack.FORFEIT;
    }
    public Color getColor() {
        return Color.BLACK;
    }
    public Direction getMove() {
        return Direction.CENTER;
    }
    public String toString() {
        return "?";
    }
}
 Enums

- Critter class has two nested Enums for Direction of movement and how to fight

```java
// constants for directions
public static enum Direction {
    NORTH, SOUTH, EAST, WEST, CENTER
};

// constants for fighting
public static enum Attack {
    ROAR, POUNCE, SCRATCH, FORFEIT
};
```
Nested Enums

- To access a Direction or Attack a class external to Critter would use the following syntax:
  - Critter.Direction.NORTH
  - Critter.ATTACK.POUNCE
- Classes that are descendants of Critter (like the ones you implement) do not have to use the Critter.
  - it is implicit
- Direction.SOUTH, Attack.ROAR
A Critter class

class name extends Critter {
    ...
}

- extends Critter tells the simulator your class is a Critter

- override methods from Critter based on Critter spec
- Critter has a number of methods not required by the 4 simple Critter classes (Ant, Bird, Vulture, Hippo)
- ... but you should use them to create an interesting and successful Longhorn
class A {
    int a;
    public A(int a) {
        this.a = a;
    }

    public int a() { return a; }
}

class B extends A {
    public B(int a) { super(a); }
}

public class Cs312 {
    public static void main(String[] args) {
        A a = new A(3);
        B b = new B(3);
        System.out.print((a.a() == b.a()) + " " + (a == b));
    }
}
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><strong>Always</strong> Attack.ROAR</td>
</tr>
<tr>
<td>getColor</td>
<td><strong>Always</strong> Color.GRAY</td>
</tr>
<tr>
<td>getMove</td>
<td><strong>Always</strong> Direction.CENTER</td>
</tr>
<tr>
<td>toString</td>
<td>&quot;S&quot;</td>
</tr>
<tr>
<td>eat</td>
<td><strong>Always</strong> false</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 fights;  // total times Critter has fought
public int getMove() {
    if (fights % 2 == 0) {
        return Direction.WEST;
    } else {
        return Direction.EAST;
    }
}
```
Testing critters

- Use the MiniMain to create String based on actions and print those out

- 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><code>public Snake(boolean northSnake)</code></td>
</tr>
<tr>
<td>fight</td>
<td>alternates between SCRATCH and POUNCE</td>
</tr>
<tr>
<td>getColor</td>
<td>Yellow</td>
</tr>
<tr>
<td>getMove</td>
<td>north bound snakes: 5 steps north, pause 5 ticks, 5 steps north, pause 5 ticks, ...</td>
</tr>
<tr>
<td></td>
<td>otherwise:</td>
</tr>
<tr>
<td></td>
<td>5 steps west, pause 5 ticks, 5 steps west, pause 5 ticks, ...</td>
</tr>
<tr>
<td>eat</td>
<td>always eats</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?