CSE112 Object Oriented Programming
Summer 2018, Project#2
Due: 13.08.2018, 23:59

IŞIK UNIVERSITY
COMPUTER SCIENCE AND ENGINEERING

SURVIVOR SIMULATION SYSTEM

You are supposed to design and implement a "survivor simulation system" with Java programming language. You are required to deliver Java codes only. UML is not required. Please note that implementing the project functionally is not sufficient for getting the full points. You are required to solve the problem in an object-oriented way. Remember reference arrays-of-objects, access modifiers (encapsulation), **inheritance**, **method overriding**, **abstract types**, **interfaces** and so on. You should not copy-paste any kind of algorithms or logic in your system. The system should be maintainable and extendable. Our main objective is to write minimum lines-of-code as possible. You are free to use many classes and interfaces as you like.

Introduction

There exists an island called the **Habitat** where 7 species of animals (birds, dinosaurs, reptiles, mammals) in 4 classes live. Animals have some abilities such as walking, flying and swimming. Some animals inherited their abilities from their classes (mammals can walk) and some animals have some specific abilities directly from their species (bats can fly). Following tables display list of animals living in the Habitat with their specific attributes and their classes (classification of animals).

Animal ID	Animal Name	Class (type of animal)	Specific Abilities	Weight (kg)
1	Bat	Mammal	CanWalk, CanFly	2
2	Brachiosaurus	Dinosaur		56000
3	Crocodile1	Reptile		400
4	Crocodile2	Reptile		350
5	Lion	Mammal	CanWalk	450
6	Pigeon	Bird		3
7	Rat	Mammal	CanWalk, CanSwim	1
8	Tyrannosaurus	Dinosaur		15000

Class	Inherent Abilities	Meaning
Mammals	--	
Birds	CanSwim	All birds can swim.
Reptiles	CanSwim, CanWalk	All reptiles can swim and walk.
Dinosaurs	CanWalk	All dinosaurs can walk.

The Problem

The simulation system should run with given initial conditions of the Habitat to **find out which animal will survive** after n runs. You can think every n as an iteration which represents some unit of time passing. For each iteration, two animals will come face-to-face. If two animals come face-to-face, that means the first animal is **attacking** to the second one. In most scenarios, when attacking happens, first animal (attacking) might kill the second animal (defending). The complete list of attacking rules are defined below:

Attacking/Defending Rules

1. As a general animal attacking rule, if the first animal (attacking) is heavier (use weight attributes) than the second animal (defending), second animal should die. Otherwise, both animals should survive (successful defense).

2. Default killing action should end with transferring the half of the defending (now dead) animals' weight to the attacking animal.
3. Only not-dead animals should come face-to-face. Already dead animals should not affect the habitat. Dead animals cannot attack or defense.
4. Reptiles can kill every non-swimming mammals by choking regardless of animals' weights.
5. If two dinosaurs come face-to-face, both of them should die. Attacker and defender die at the same time.
6. Dinosaurs can't catch/kill any flying animals.
7. Since Brachiosaurus is very bulky, it can only attack to walking animals.

Some additional requirements for the system:

8. Every animal must have a unique integer ID value. IDs cannot be zero or null. See the tables for animal IDs.
9. Every animal can only be in a one class (type of animal) at a time. For example: T-rex is a dinosaur, and cannot be a bird, mammal or a reptile.
10. Habitat could include multiple instances of one animal (species) at a time. There can be two crocodiles (crocodile1, crocodile2) in the habitat.
11. In this habitat, every bird's attacking rules must be the same as with the default attacking rules. For future, there will be no **birds** to define new attacking rule rejecting the default one. As a programmer, you should guarantee that.

Main Habitat Setup

1. Add 8 animals from the table into the Habitat.
2. Setup the following attacks in order and run the Habitat (12 iterations).
   1. lion -> trex (lion attacks trex)
   2. crococile1 -> crocodile2
   3. crocodile1 -> lion
   4. trex -> lion
   5. brachiosaurus -> rat
   6. rat -> pigeon
   7. trex -> bat
   8. lion -> trex
   9. crocodile1 -> pigeon
   10. trex -> crocodile
   11. bat -> rat
   12. brachiosaurus -> trex

Implementation Hints

Below are some technical hints which can be useful to know when implementing this problem.

1. When extending a class, if the super class has a constructor with arguments, you must pass those arguments to the super type from derived types constructor. Ex:

```
//constructor of a derived type
public Dinosaur(int ID, float w) {
        super(ID,w);   //passing arguments to the super class// calling super's ctor.

        //derived type's own constructor logic here.
}
```

2. Inheritance helps you to avoid code copy-pasting.
3. In object-oriented programming, we represent abilities with interfaces. Classes can **implement** multiple interfaces at the same time. When **extending** a class (or an abstract class), you can only have a single super/base/parent class. Notice that **implementing and extending are different** in that sense.
4. Polymorphism can help you using multiples bodies of algorithms under the single contract. For example, in this Habitat problem, you can represent the attacking concept with the following contract below:
```
public void Attack(Animal defending)
```
5. When overriding methods, you can still use your super's body if you like. Ex:
```
@Override
public void MethodToOverride(int x){
    if (x > 0){
        //do that algorithm instead of super's.
    }
    else{        //use super's implementation
        super.MethodToOverride(x);
    }
}
```
6. You can override your objects' toString() methods for easier debugging and printing.
7. Markup interfaces: Sometimes interfaces can have no methods/contracts. We call them markup interfaces.
```
public interface CanDoSomeAbility {
    //no special methods here for this ability.
}
```
8. You can check whether some object has implemented that ability or not by the following code:
```
if (animal instanceof CanDo){   //means that object has implemented that ability/interface.
    //do this
}
```

Expected Output

Below are the expected outputs from the simulation system. Please report what happened in every attack and report the final survivor to the screen:

Iteration: 1
Att: Lion[5] (450.0 kg)
Def: Trex[8] (15000.0 kg)

```
No one died.
Nr of living animals: 8 (You can list those animals if you like.)

Other iteration reports here……………….

……………

Iteration: 10
Att: Trex[8] (15225.0 kg)
Def: Crocodile[3] (801.5 kg)
Crocodile[3] (801.5 kg) died.
Attacker Trex[8] (15625.75 kg) new weight is 15625.75
Nr of living animals: 3

……. iterations here….

SURVIVOR IS: Some {Animal} with {X} weight.
```

Happy coding.