```
/
**********************************************************************
****

 Class:  Aging Lions
Author:  Jacob Rust
  Date:  November 28, 2018

**********************************************************************
*****/

import java.awt.Color;

public class agingLion extends Lion implements Predator
{
    private double visualRange = 30.0;
    private int age;

    /**
    *   Constructor creates a Lion with Position 0,0.  Animal
    *   has no cage in which to live.
    */
    public agingLion()
    {
        super();
        age = 0;
    }

    /**
    *   Constructor creates a Lion in a random empty spot in
    *   the given cage.
    *   @param cage  the cage in which lion will be created.
    */
    public agingLion(Cage cage)
    {
        super(cage, Color.yellow);
        age = 0;
    }

    /**
    *   Constructor creates a Lion in a random empty spot in
```

```
*   the given cage with the specified Color.
*   @param cage  the cage in which lion will be created.
*   @param color  the color of the lion
*/
public agingLion(Cage cage, Color color)
{
    super(cage, color);
    age = 15;
}

/**
*   Constructor creates a Lion in the given Position
*   the given cage with the specified Color.
*   @param cage  the cage in which lion will be created.
*   @param color  the color of the lion
*   @param pos  the position of the lion
*/
public agingLion(Cage cage, Color color, Position pos)
{
    super(cage, color, pos);
    age = 15;
}

/**
*   Method causes the Lion to act.  This may include
*   any number of behaviors (moving, eating, etc.).
*/
public void act()
{
    age++;
    int xPrey, yPrey, myX, myY;

    // if Lion is full, it just lays around
    if(age == 0)
    {
        myColor = Color.yellow;
        return;
    }
    if(age == 25)
    {
        myColor = Color.green;
```

```
    return;
}
if(age == 50)
{
    myColor = Color.red;
    return;
}
if(age == 75)
{
    myColor = Color.blue;
    return;
}
if(age == 100)
{
    myColor = Color.black;
    return;
}

Animal closestPrey = findClosestPrey();

if(isSomethingICanEat(closestPrey)==true)
{
    xPrey = closestPrey.getPosition().getX();
    yPrey = closestPrey.getPosition().getY();
    myX = myPos.getX();
    myY = myPos.getY();
    Position newPos, oldPos = new Position(myX, myY);

    // Compare x and y coordinates and move toward
    // the Prey (by adding or subtracting one to each)
    if(xPrey>myX)
        myX++;
    else if (xPrey<myX)
        myX--;
    if(yPrey>myY)
        myY++;
    else if (yPrey<myY)
        myY--;

    newPos = new Position(myX, myY);
```

```
            // check to see if Lion just caught Prey
            if(newPos.equals(closestPrey.getPosition()))
            {
                closestPrey.kill();
                myCage.removeAnimal(closestPrey);
                myPos = newPos;
                myCage.moveAnimal(oldPos, this);
            }
            // check to see if newPos is empty
            else if (myCage.isEmptyAt(newPos))
            {
                myPos = newPos;
                myCage.moveAnimal(oldPos, this);
            }
            // newPos was already filled, move as generic Animal
            else
                super.act();

        }
        else // no Prey was seen, move as generic Animal
        {
            super.act();
        }
    }


    /**
    *   Method returns the closest Prey to the Lion provided that Prey
is
    *   also within the Lion's visual range.  If no Prey is seen it
will return
    *   a generic Animal.
    *   @return closest Prey the Lion can see
    */
    public Animal findClosestPrey()
    {
        Animal closestPrey = new Animal(myCage);
        double distanceToClosest = visualRange+.01;
        // Distance set to just longer than a Lion can see

        for(int y=0; y<myCage.getMax_Y(); y++)
```

```
        {
            for(int x=0; x<myCage.getMax_X(); x++)
            {
                if(isSomethingICanEat(myCage.animalAt(x,y)) == true)
                {
                    if(myPos.distanceTo(new Position(x,y)) <
distanceToClosest)
                    {
                        closestPrey = myCage.animalAt(x,y);
                        distanceToClosest = myPos.distanceTo(new
Position(x,y));
                    }
                }
            }
        }

        return closestPrey;
    }

    /**
    *   Method returns true if obj is a type the animal can eat,
    *   returns false otherwise
    *   @param  obj object to be evaluated
    *   @return true if obj can be eaten, false otherwise
    */
    public boolean isSomethingICanEat(Animal obj)
    {
        if(obj instanceof Prey)
        {
            return true;
        }
        return false;
    }

    /**
    *   Method sets the Lions's visual range to the given value.
    *   @param range  sets the Lion's visual range to 'range'
    */
    public void setVisualRange(double range)
    {
        visualRange = range;
```

```
    }

    /**
    *   Returns String form of Animal, which is its position
    *   and its type.
    *   @return String form of Animal
    */
    public String toString()
    {
        return (myPos.toString() + " is a Lion.  ");
    }

    /**
    *   Method returns the String form of the Animal's
    *   species, in this case "HungryLion"
    *   @return the String "HungryLion"
    */
    public String getSpecies()
    {
        return "Aging Lion";
    }

}
```