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

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

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


import java.awt.*;


public class SmartLion extends Lion implements Predator
{


private double visualRange = 50.0;

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

    /**
    *   Constructor creates a Lion in a random empty spot in
    *   the given cage.
    *   @param cage  the cage in which lion will be created.
    */
    public SmartLion(Cage cage)
    {
        super(cage, Color.black);
    }
```

```
    /**
    *   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 SmartLion(Cage cage, Color color)
    {
        super(cage, color);
    }

    /**
    *   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 SmartLion(Cage cage, Color color, Position pos)
    {
        super(cage, color, pos);
    }

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

        Animal closestPrey = findClosestPrey();


        //finds prey within a 45 x 45 area
        if(closestPrey.getPosition().getX() < 45 &&
closestPrey.getPosition().getY() < 45)
        {
            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 //moves away from other lions
        {

            Animal closestPredator = findClosestLion();
            //if the closest predator is a type of lion, it will move
            //it will stay a certain distance away from the other lions
        if(closestPredator instanceof SmartLion || closestPredator
instanceof Lion || closestPredator instanceof HungryLion ||
closestPredator instanceof agingLion)
        {
            int predatorX = closestPredator.getPosition().getX();
            int predatorY = closestPredator.getPosition().getY();
            int myX1 = myPos.getX();
            int myY1 = myPos.getY();
            Position newPos, oldPos = new Position(myX1, myY1);

            if(predatorX > myX1 && myX1 > 0)
                myX1--;
            else if (predatorX < myX1 && myX1 < myCage.getMax_X()-1)
                myX1++;
            if(predatorY > myY1 && myY1 > 0)
                myY1--;
            else if(predatorY < myY1 && myY1 < myCage.getMax_Y()-1)
                myY1++;
            newPos = new Position(myX1, myY1);

            // SmartLion could not move away, so it moves as a
            // generic Prey, which means randomly
            if(newPos.equals(oldPos))
                super.act();
            // SmartLion moves to new position which is empty
            else if (myCage.isEmptyAt(newPos))
            {
                myPos = newPos;
                myCage.moveAnimal(oldPos, this);
            }
        }
    }
}
```

```
    /**
    *   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 the closest predator that is not a dingo
    //Method returns the closest predator not in the same location as
the current animal
    public Animal findClosestLion()
```

```
    {
        Animal closestPredator = new Animal(myCage);
        double distanceToClosest = visualRange+.01;


        for(int y=0; y<myCage.getMax_Y(); y++)
        {
            for(int x=0; x<myCage.getMax_X(); x++)
            {

                //finds a predator that is not a dingo


                if(myCage.animalAt(x,y) instanceof Predator & !
(myCage.animalAt(x, y) instanceof Dingos) & x !=(myPos.getX()) & y !
=(myPos.getY()))
                {
                    if(myPos.distanceTo(new Position(x,y)) <
distanceToClosest)
                    {
                        closestPredator = myCage.animalAt(x,y);
                        distanceToClosest = myPos.distanceTo(new
Position(x,y));
                    }
                }
            }
        }
        return closestPredator;
    }

    /**
    *   Method returns true if obj is a type the animal can eat,
    *   returns false otherwise

    *   @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 "Lion"
*   @return the String "Lion"
*/
public String getSpecies()
{
    return "Lion";
}
```

```
}
```