```java
//Class: Fast Gazelle
//Name: Jacob Rust
//Date: 11/12/18



import java.awt.*;


import java.util.*;


public class FastGazelle extends Gazelle
{
    private static double visualRange = 30.0;

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

    /**
    *   Constructor creates a SmartGazelle in a random empty spot in
    *   the given cage.
    *   @param cage  the cage in which Gazelle will be created.
    */
    public FastGazelle(Cage cage)
    {
        super(cage, Color.green);
    }


    /**
    *   Constructor creates a SmartGazelle in a random empty spot in
    *   the given cage with the specified Color
    *   @param cage  the cage in which Gazelle will be created.
    *   @param color  the color of the Gazelle
```

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

/**
*   Constructor creates a SmartGazelle in the given Position
*   the given cage with the specified Color.
*   @param cage  the cage in which Gazelle will be created.
*   @param color  the color of the Gazelle
*   @param pos  the position of the Gazelle
*/
public FastGazelle(Cage cage, Color color, Position pos)
{
    super(cage, color, pos);
}

/**
*   Method sets the Gazelle's visual range to the given value.
*   @param range  sets the Gazelle'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 Smart Gazelle.  ");
}

/**
*   Method overwrites the Act method in Animal.  Gazelle will
*   attempt to move away from a lion if it sees the lion.
*/
```

```
    public void act()
    {
        Animal closestPredator = findClosestPredator();

        // In this case it sees a predator and tries to run away
        if(closestPredator instanceof Predator)
        {
            int predatorX = closestPredator.getPosition().getX();
            int predatorY = closestPredator.getPosition().getY();
            int myX = myPos.getX();
            int myY = myPos.getY();
            Position newPos, oldPos = new Position(myX, myY);

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

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

```
        if(closestPredator instanceof Predator)
        {
            int predatorX = closestPredator.getPosition().getX();
            int predatorY = closestPredator.getPosition().getY();
            int myX = myPos.getX();
            int myY = myPos.getY();
            Position newPos, oldPos = new Position(myX, myY);

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

            // SmartGazelle could not move away, so it moves as a
            // generic Prey, which means randomly
            if(newPos.equals(oldPos))
                super.act();
            // SmartGazelle moves to new position which is empty
            else if (myCage.isEmptyAt(newPos))
            {
                myPos = newPos;
                myCage.moveAnimal(oldPos, this);
            }
            // SmartGazelle could not move to a new location because
            // it was not empty, so it moves as a generic Prey
(randomly)
            else
            {
                super.act();
            }
        }
        if(closestPredator instanceof Predator)
        {
            int predatorX = closestPredator.getPosition().getX();
            int predatorY = closestPredator.getPosition().getY();
```

```
            int myX = myPos.getX();
            int myY = myPos.getY();
            Position newPos, oldPos = new Position(myX, myY);

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

            // SmartGazelle could not move away, so it moves as a
            // generic Prey, which means randomly
            if(newPos.equals(oldPos))
                super.act();
            // SmartGazelle moves to new position which is empty
            else if (myCage.isEmptyAt(newPos))
            {
                myPos = newPos;
                myCage.moveAnimal(oldPos, this);
            }
            // SmartGazelle could not move to a new location because
            // it was not empty, so it moves as a generic Prey
(randomly)
            else
            {
                super.act();
            }
        }if(closestPredator instanceof Predator)
        {
            int predatorX = closestPredator.getPosition().getX();
            int predatorY = closestPredator.getPosition().getY();
            int myX = myPos.getX();
            int myY = myPos.getY();
            Position newPos, oldPos = new Position(myX, myY);

            if(predatorX > myX && myX > 0)
                myX--;
```

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

            // SmartGazelle could not move away, so it moves as a
            // generic Prey, which means randomly
            if(newPos.equals(oldPos))
                super.act();
            // SmartGazelle moves to new position which is empty
            else if (myCage.isEmptyAt(newPos))
            {
                myPos = newPos;
                myCage.moveAnimal(oldPos, this);
            }
            // SmartGazelle could not move to a new location because
            // it was not empty, so it moves as a generic Prey
(randomly)
            else
            {
                super.act();
            }
        }if(closestPredator instanceof Predator)
        {
            int predatorX = closestPredator.getPosition().getX();
            int predatorY = closestPredator.getPosition().getY();
            int myX = myPos.getX();
            int myY = myPos.getY();
            Position newPos, oldPos = new Position(myX, myY);

            if(predatorX > myX && myX > 0)
                myX--;
            else if (predatorX < myX && myX < myCage.getMax_X()-1)
                myX++;
            if(predatorY > myY && myY > 0)
                myY--;
            else if(predatorY < myY && myY < myCage.getMax_Y()-1)
                myY++;
```

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

            // SmartGazelle could not move away, so it moves as a
            // generic Prey, which means randomly
            if(newPos.equals(oldPos))
                super.act();
            // SmartGazelle moves to new position which is empty
            else if (myCage.isEmptyAt(newPos))
            {
                myPos = newPos;
                myCage.moveAnimal(oldPos, this);
            }
            // SmartGazelle could not move to a new location because
            // it was not empty, so it moves as a generic Prey
(randomly)
            else
            {
                super.act();
            }
        }if(closestPredator instanceof Predator)
        {
            int predatorX = closestPredator.getPosition().getX();
            int predatorY = closestPredator.getPosition().getY();
            int myX = myPos.getX();
            int myY = myPos.getY();
            Position newPos, oldPos = new Position(myX, myY);

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

            // SmartGazelle could not move away, so it moves as a
            // generic Prey, which means randomly
            if(newPos.equals(oldPos))
                super.act();
```

```
            // SmartGazelle moves to new position which is empty
            else if (myCage.isEmptyAt(newPos))
            {
                myPos = newPos;
                myCage.moveAnimal(oldPos, this);
            }
            // SmartGazelle could not move to a new location because
            // it was not empty, so it moves as a generic Prey
(randomly)
            else
            {
                super.act();
            }
        }
        // SmartGazelle could not see a predator, so it acts as a
generic
        // Prey would act, meaning random movement
        else
            super.act();
    }

    /**
    *   Method returns the closest Predator to the Gazelle provided
that
    *   Predator is also within the Gazelle's visual range, if no
Predators
    *   are seen the method returns a generic Animal.
    *   @return closest Predator the Gazelle can see
    */
    public Animal findClosestPredator()
    {
        Animal closestPredator = new Animal(myCage);
        double distanceToClosest = visualRange+.01;
        // Distance set to just longer than a Gazelle can see

        for(int y=0; y<myCage.getMax_Y(); y++)
        {
            for(int x=0; x<myCage.getMax_X(); x++)
            {
                if(myCage.animalAt(x,y) instanceof Predator)
                {
```

```
                    if(myPos.distanceTo(new Position(x,y)) <
distanceToClosest)
                    {
                        closestPredator = myCage.animalAt(x,y);
                        distanceToClosest = myPos.distanceTo(new
Position(x,y));
                    }
                }
            }
        }
        return closestPredator;
    }

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



}
```