Set 7

The source code for the Critter class is in the critters directory

1. What methods are implemented in Critter?

Ans： act getActors processActors getMovelocations selectMoveLocation

2. What are the five basic actions common to all critters when they act?

Ans: the five basic actions are below

```
ArrayList<Actor> actors = getActors();
processActors(actors);
ArrayList<Location> moveLocs = getMoveLocations();
Location loc = selectMoveLocation(moveLocs);
makeMove(loc);
```

3. Should subclasses of Critter override the getActors method? Explain.

Ans: yes, because maybe the subclasses require actors in concrete direction

4. Describe the way that a critter could process actors.

Ans: if the critter is not Rock or Critter , it will eat them(disappear);

5. What three methods must be invoked to make a critter move? Explain each of these methods.

Ans: first you should get the movable position : getMoveLocation

then select one location for the movable location ; selectMoveLocation

final move : makeMove

6. Why is there no Critter constructor?

Ans: because no necessary requirement

Set 8

The source code for the ChameleonCritter class is in the critters directory

1. Why does act cause a ChameleonCritter to act differently from a Critter even though ChameleonCritter does not override act?
Ans: because the five steps is the same, you should change the some of the five steps. Otherwise, you can inherit the Actor class

2. Why does the makeMove method of ChameleonCritter call super.makeMove?
Ans： this can reuse the parent code and reduce the press of work

3. How would you make the ChameleonCritter drop flowers in its old location when it moves?
Ans: test the flower, process the flower to drop flowers

4. Why doesn't ChameleonCritter override the getActors method?
Ans: originally, the ChameleonCritter should gets the all Actors from neighbor

5. Which class contains the getLocation method?
Ans: Actor

6. How can a Critter access its own grid?
Ans: getGrid()

Set 9

The source code for the CrabCritter class is reproduced at the end of this part of GridWorld.

1. Why doesn't CrabCritter override the processActors method?

Ans: it only need the default method

2. Describe the process a CrabCritter uses to find and eat other actors. Does it always eat all neighboring actors? Explain.

Ans: it select three location of its direction , AHEAD,HALF_LEFT, HALF_RIGHT

,eat the actors except the Rock and Critter

3. Why is the getLocationsInDirections method used in CrabCritter?

Ans: ovrewriter the getMoveLocation function, so you should add something

4. If a CrabCritter has location (3, 4) and faces south, what are the possible locations for actors that are returned by a call to the getActors method?

Ans: (4,4) , (4,3) , (4,5)

5. What are the similarities and differences between the movements of a CrabCritter and a Critter?

Ans: CrabCritter can turn angle,

Critter go straight

all move one pane per step

6. How does a CrabCritter determine when it turns instead of moving?

Ans： judge the left and right location, if all null , move, otherwise turn

7. Why don't the CrabCritter objects eat each other?

Ans: in Critter's processActors, there is a control and CrabCritter inherits the Critter class

Exercises

1. Modify the processActors method in ChameleonCritter so that if the list of actors to process is empty, the color of the ChameleonCritter will darken (like a flower).

```
public class ChameleonCritter extends Critter{
    private static final double DARKENING_FACTOR = 0.05;
    public void processActors(ArrayList<Actor> actors){
        int n = actors.size();
        if (n == 0) {
            Color c = getColor();
            int red = (int) (c.getRed() * (1 - DARKENING_FACTOR));
            int green = (int) (c.getGreen() * (1 - DARKENING_FACTOR));
            int blue = (int) (c.getBlue() * (1 - DARKENING_FACTOR));
            setColor(new Color(red, green, blue));
            return;
        }
        int r = (int) (Math.random() * n);
        Actor other = actors.get(r);
        setColor(other.getColor());
    }

}
```

In the following exercises, your first step should be to decide which of the five methods--~~getActors, processActors, getMoveLocations, selectMoveLocation, and makeMove~~-- should be changed to get the desired result.

2. Create a class called ChameleonKid that extends ChameleonCritter as modified in exercise 1. A ChameleonKid changes its color to the color of one of the actors immediately in front or behind. If there is no actor in either of these locations, then the ChameleonKid darkens like the modified ChameleonCritter.

Ans: change getActors

```
public class ChameleonKid extends ChameleonCritter
{
    public ArrayList<Actor> getActors(){
        ArrayList<Actor> actors = new ArrayList<Actor>();
        Location loc = getLocation();
        Location neighborLoc1 = loc.getAdjacentLocation(getDirection());
        Location neighborLoc2 = loc.getAdjacentLocation(getDirection()+180);
        Grid<Actor> grid =  getGrid();
        if (grid.isValid(neighborLoc1) && grid.get(neighborLoc1) != null) actors.add(grid.get(neighborLoc1));
        if (grid.isValid(neighborLoc2) && grid.get(neighborLoc2) != null ) actors.add(grid.get(neighborLoc2));
        return actors;
    }
}
```

3. Create a class called RockHound that extends Critter. A RockHound gets the actors to be processed in the same way as a Critter. It removes any rocks in that list from the grid. A RockHound moves like a Critter.

Ans: processActors

```
public class RockHound extends Critter
{
  public void processActors(ArrayList<Actor> actors)
  {
    for (Actor a : actors)
    {
      if (!(a instanceof Critter))
        a.removeSelfFromGrid();
    }
  }
}
```

4. Create a class BlusterCritter that extends Critter. A BlusterCritter looks at all of the neighbors within two steps of its current location. (For a BlusterCritter not near an edge, this includes 24 locations). It counts the number of critters in those locations. If there are fewer than c critters, the BlusterCritter's color gets brighter (color values increase). If there are c or more critters, the BlusterCritter's color darkens (color values decrease). Here, c is a value that indicates the courage of the critter. It should be set in the constructor.

Ans： getActors processActors

```
public class BlusterCritter extends Critter
{
   private int  criticalCount;
   private int  currentCount;
   public BlusterCritter(int c) {
      criticalCount = c;
      currentCount = 0;
      setColor(Color.RED);
   }

   public ArrayList<Actor> getActors()
   {
      // to-do list  get rect  , scan it
      Location currentLocation = getLocation();
      Location leftUpLocation = new Location(currentLocation.getRow()-2, currentLocation.getCol()-2);
      Location leftDownLocation = new Location(currentLocation.getRow()+2,currentLocation.getCol()-2);
      Location rightUpLocation = new Location(currentLocation.getRow() -2, currentLocation.getCol()+2);
      Location rightDownLocation = new Location(currentLocation.getRow()+2,currentLocation.getCol()+2);

      ArrayList<Actor> actors = new ArrayList<Actor>();
      for (int rowc = leftUpLocation.getRow(); rowc <= leftDownLocation.getRow(); rowc++) {
         for (int colc = leftUpLocation.getCol(); colc <= rightUpLocation.getCol(); colc++) {
            Location  oneLocation = new Location(rowc, colc);
            if (getGrid().isValid(oneLocation) && (currentLocation.getRow() != rowc|| currentLocation.getCol() !=
colc)) {
                   Actor neighbor = getGrid().get(oneLocation);
                    System.out.println(oneLocation);
                   if (neighbor != null && neighbor instanceof Critter) {
                       actors.add(neighbor);


                   }
                }


            }
        }
        return actors;


    }


    public void processActors(ArrayList<Actor> actors)
    {
        //System.out.println(criticalCount + "     " + actors.size());
        if (actors.size() < criticalCount) {
              // setColor(getColor().brighter());
                setColor(Color.RED);
              // setColor(new Color(10 * actors.size(),10 * actors.size(), 10 * actors.size() ));
        } else {
                setColor(Color.BLUE);
              //setColor(getColor().darker());
               setColor(new Color(1 * actors.size(),2 * actors.size(), 3 * actors.size() ));
        }
    }
}
```

5. Create a class QuickCrab that extends CrabCritter. A QuickCrab processes actors the same way a CrabCritter does. A QuickCrab moves to one of the two locations, randomly selected, that are two spaces to its right or left, if that location and the intervening location are both empty. Otherwise, a QuickCrab moves like a CrabCritter.

Ans： getMoveLocation

```
 /**
 * @return list of empty locations immediately to the right and to the left
 */
public ArrayList<Location> getMoveLocations()
{
    ArrayList<Location> locs = new ArrayList<Location>();
    Location cl = getLocation();
    Location l1 = cl.getAdjacentLocation(getDirection()+Location.LEFT);
    Location r1 = cl.getAdjacentLocation(getDirection()+Location.RIGHT);
    Location l2 = l1.getAdjacentLocation(getDirection()+Location.LEFT);
    Location r2 = r1.getAdjacentLocation(getDirection()+Location.RIGHT);
    int open1 = 0;
    int openl1 = 0;
    int openl2 = 0;
    if (getGrid().isValid(l1) && getGrid().get(l1) == null) {
        openl1 = 1;
        if (getGrid().isValid(l2)&& getGrid().get(l2) == null) {
            locs.add(l2);
            open1 = 1;
        }
    }
    if (getGrid().isValid(r1) && getGrid().get(r1) == null) {
        openl2 = 1;
        if (getGrid().isValid(r2)&& getGrid().get(r2) == null) {
            locs.add(r2);
            open1 = 1;
        }
    }
    if (open1 == 0&& openl1 == 1) locs.add(l1);
    if (open1 == 0&& openl2 == 1) locs.add(r1);
    return locs;
}
```

6. Create a class KingCrab that extends CrabCritter. A KingCrab gets the actors to be processed in the same way a CrabCritter does. A KingCrab causes each actor that it processes to move one location further away from the KingCrab. If the actor cannot move away, the KingCrab removes it from the grid. When the KingCrab has completed processing the actors, it moves like a CrabCritter.

Ans: processActors

```
public class KingCrab extends CrabCritter{
    public void processActors(ArrayList<Actor> actors) {
        int[] dirs =   { Location.AHEAD + getDirection(), Location.HALF_LEFT + getDirection(), Location.HALF_RIGHT + 
getDirection()};
        int count = 0;
       // System.out.println("总数" +  actors.size());
        for (Actor a : actors) {
            if (getGrid().get(getLocation().getAdjacentLocation(dirs[count])) == a) {
        //      System.out.println(count);
                break;
            }
            count++;
        }

        for (Actor a : actors) {
            if (getGrid().get(getLocation().getAdjacentLocation(dirs[count])) == a) {
          //    System.out.println(count);
                Location xx = getLocation().getAdjacentLocation(dirs[count]);
                Location newxx = xx.getAdjacentLocation(dirs[count]);
                if (getGrid().isValid(newxx)) {
                    a.moveTo(newxx);
                } else {
                    a.removeSelfFromGrid();
                }
                count++;
            }
        }
    }
}
```