中山大學
SUN YAT-SEN UNIVERSITY

# 实训报告

| | | | | | | |
|---|---|---|---|---|---|---|
| 作业名称 | Part 4: Interacting Objects | | | | | |
| 院系 | 软件学院 | | 班级 | 教务 2 班 | 姓名 | 李明宽 |
| 学号 | 11331173 | | 完成日期 | 2013 年 07 月 05 日 | | |
| **QQ** | 736459905 | | **E-mail** | limkuan@mail2.sysu.edu.cn | | |

## The Critter Class

Critters are actors that share a common pattern of behavior, but the details may vary for each type of critter. When a critter acts, it first gets a list of actors to process. It processes those actors and then generates the set of locations to which it may move, selects one, and moves to that location.

Different types of critters may select move locations in different ways, may have different ways of selecting among them, and may vary the actions they take when they make the move. For example, one type of critter might get all the neighboring actors and process each one of them in some way (change their color, make them move, and so on). Another type of critter may get only the actors to its front, front-right, and front-left and randomly select one of them to eat. A simple critter may get all the empty neighboring locations, select one at random, and move there. A more complex critter may only move to the location in front or behind and make a turn if neither of these locations is empty.

### Do You Know?

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

1. What methods are implemented in Critter?
   Answer: The class methods implemented six methods, they are act() which let the critter act, getActors() which get all actor is adjacent to the critter, processActors(ArrayList<Actor>) which remove all actor except critter or rock around the critter, getMoveLocations() which get all available cell that adjacent to critter, selectMoveLocation(ArrayList<Location>) which random select a location from the location list, makeMove(Location) which move the critter to the location .

1. What are the five basic actions common to all critters when they act?
   Answer: The five basic actions of critter is get actors around itself, remove the actor except rock and critter from the grid, get the adjacent location can move to, random pick a location, and move to the location.

2. Should subclasses of Critter override the getActors method? Explain.
   Answer: It depends on whether the subclasses act like critter.
   If it also needs to process the actor around itself except rock and critter, the method doesn't need to be overridden. But if the subclasses don't act like critter, they process other actors, in this way, the getActors method need to be overridden. In other words, the getActors method could be overridden.

3. Describe the way that a critter could process actors.
   Answer: It can eat them all, or change the color of them, change them direction or other attributes, or critter can let them evolve, such as a bug to a BoxBug, a flower to a rock etc.

4. What three methods must be invoked to make critter move? Explain each of these methods.
   Answer: First, critter must know which cell adjacent it is available, which means the method getMoveLocations(), and the critter must choose a location to move to, which means the method selectMoveLocation(ArrayList<Location>), finally, the critter need to move to the cell just chose, which means the method makeMove(Location). This three methods must be invoked to make a critter move.

5. Why is there no Critter constructor?
   Answer: It use the constructor from its super class Actor. Because we also a blue critter which is facing to north like its father Actor. So it doesn't need to implement its own constructed function.

## Default Critter Behavior

Before moving, critters process other actors in some way. They can examine them, move them, or even eat them.

There are two steps involved:

1. Determination of which actors should be processed
2. Determination of how they should be processed

## Extending the Critter Class

The ChameleonCritter class defines a new type of critter that gets the same neighboring actors as a Critter. However, unlike a Critter, a ChameleonCritter doesn't process actors by eating them. Instead, when a ChameleonCritter processes actors, it randomly selects one and changes its own color to the color of the selected actor.

### Do You Know?

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?

   Answer: Because the makeMove method has been overridden.

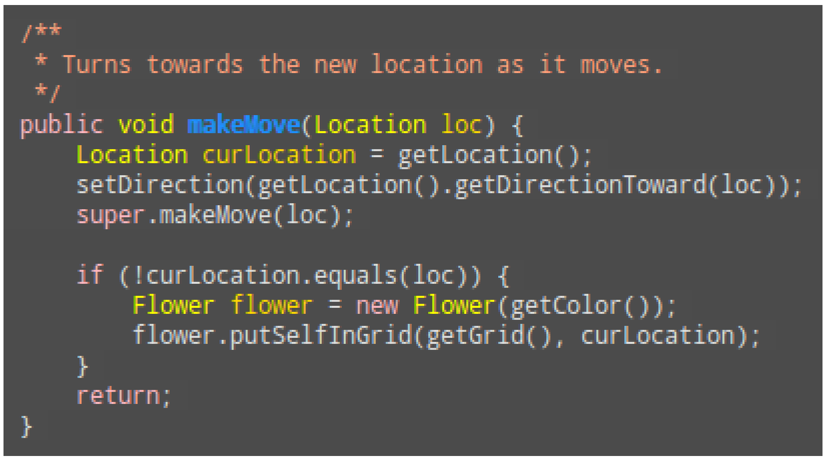
   While the ChameleonCritter is move, it call the act method which if from its superclass Critter, and it get the actor that wants to process, and processes them, while it move, the act method is calling the function makeMove. ChameleonCritter override the function makeMove, so it can be different to the Critter.

2. Why does the makeMove method of ChameleonCritter call super.makeMove?

   Answer: After change the direction, it needs to move to the location which it just chose. In this situation, it acts like Critter, so it can call the function makeMove from its superclass to move to the new location.

3. How would you make the ChameleonCritter drop flower in its old location when it moves?

   Answer: We can drop flower in its old location in this way:



```java
/**
 * Turns towards the new location as it moves.
 */
public void makeMove(Location loc) {
    Location curLocation = getLocation();
    setDirection(getLocation().getDirectionToward(loc));
    super.makeMove(loc);

    if (!curLocation.equals(loc)) {
        Flower flower = new Flower(getColor());
        flower.putSelfInGrid(getGrid(), curLocation);
    }
    return;
}
```

   Must be attention that if the old location is equal loc, in other words, the critter can't move to the cell adjacent it and just stay in old location, we can't drop a flower into it.

4. Why doesn't ChameleonCritter override the getActors method?

   Answer: ChameleonCritter and Critter get the actors that adjacent themself, they are the same and get the same actor list. So we don't need to override the getActors method again, just use the method implement in Critter class.

5. Which class contains the getLocation method?

   Answer: The Actor class contains the implement of getLocation method. And the subclass of Actor can use the getLocation method directly.

6. How can a Critter access its own grid?

   Answer: It can get its own grid by invoking the method getGrid().

## Another Critter

A CrabCritter is a critter that eats whatever is found in the locations immediately in front, to the right-front, or to the left-front of it. It will not eat a rock or another critter (this restriction is inherited from the Critter class). A CrabCritter can move only to the right or to the left. If both locations are empty, it randomly selects one. If a CrabCritter cannot move, then it turns 90 degrees, randomly to the left or right.

### Do You Know?

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?

   Answer: Because the method of processing actors is the same.

   The CrabCritter and Critter 'eat' the actor they pick, means that remove the actor from grid. The difference between CrabCritter and Critter is which actors they pick, Critter pick all actors except critter and rock around it, and CrabCritter pick actors in front, to the right-front and to the left-front.

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

   Answer: The CrabCritter find actor adjacent it only in three directions, in front of the CrabCritter, to the right-front and to the left-front. After it picks the actors, it will eat them if the actor isn't rock or critter.

   It only processes three directions around itself. In other directions, any actor can't be processed. So it is not always eat all neighboring actors, because if the neighboring actors are rock or critter, or are in other direction towards the CrabCritter, the CrabCritter doesn't eat them.

3. Why is the getLocationsInDirections method used in CrabCritter?
   Answer: We can get a location list by the array of directions. In CrabCritter, it uses to get the location which is facing CrabCritter, which return the front, to the right-front and to the left-front locations, preparing to pick the actor from these locations and to process them. And it uses to get the location in CrabCritter's left and right, to determine whether the CrabCritter can move.

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?
   Answer: The Actor which locates in (4, 3), (4, 4) and (4, 5) is returned by getActors method.

5. What are the similarities and differences between the movements of a CrabCritter and a Critter?
   Answer: The similarities and differences between the movements of a CrabCritter and a Critter are below:
   Similarities: They are all random move to a location around them which is available.
   Differences: The Critter can move to locations in every direction, and when it can't move, it never turns. But the CrabCritter can only move to the location which is the left-side and right-side to the CrabCritter, and it can't move, it will turn 90 degrees and test again.

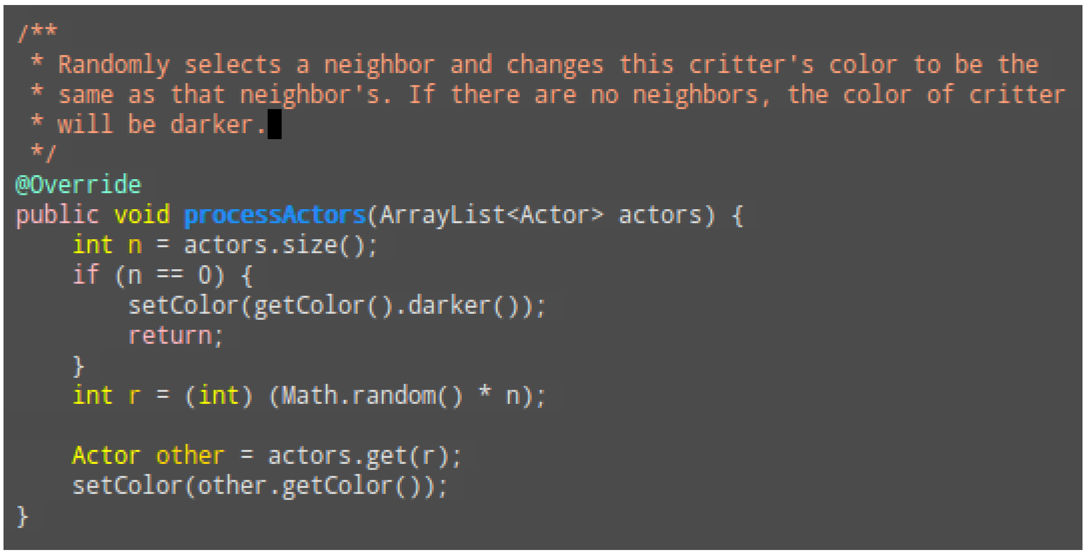
6. How does a CrabCritter determine when it turns instead of moving?
   Answer: When it can move, it will turn. In other words, when the getMoveLocations method returns a null ArrayList, it means that the CrabCritter can't move to the right side or the left side, so the CrabCritter will turns for 90 degrees.

7. Why don't the CrabCritter objects eat each other?
   Answer: Because CrabCritter don't eat rock and other critter, and the CrabCritter is the subclass of the class Critter, is also instance of Critter, so CrabCritter don't eat each other.

## 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).

   Answer：This exercise is easy and just set the color darker when the list of actors is empty. The core code is below:



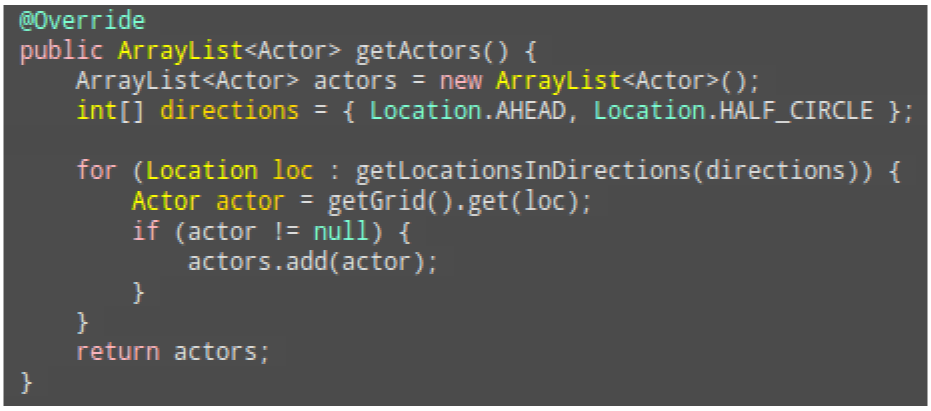
```
/**
 * Randomly selects a neighbor and changes this critter's color to be the
 * same as that neighbor's. If there are no neighbors, the color of critter
 * will be darker.
 */
@Override
public void processActors(ArrayList<Actor> actors) {
    int n = actors.size();
    if (n == 0) {
        setColor(getColor().darker());
        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.

   Answer: Use the method getLocationsInDirections, and override the function getActors, just return the actor which is in front or behind.

   The core code is below:

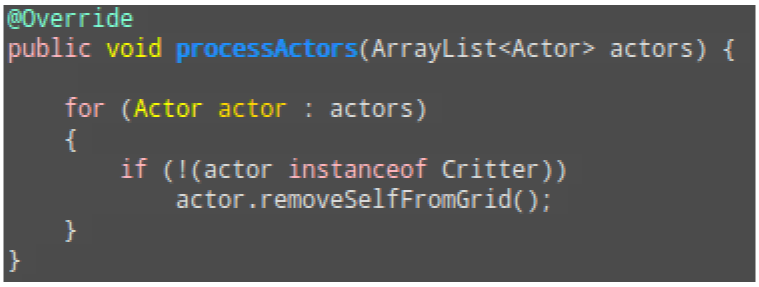


```
@Override
public ArrayList<Actor> getActors() {
    ArrayList<Actor> actors = new ArrayList<Actor>();
    int[] directions = { Location.AHEAD, Location.HALF_CIRCLE };

    for (Location loc : getLocationsInDirections(directions)) {
        Actor actor = getGrid().get(loc);
        if (actor != null) {
            actors.add(actor);
        }
    }
    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.

   Answer: Just override the function processActors, if the actor is not a critter, the RockHound will eat them.

   The core code is below:



```java
@Override
public void processActors(ArrayList<Actor> actors) {

    for (Actor actor : actors)
    {
        if (!(actor instanceof Critter))
            actor.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 is 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.

   Answer: The class BlusterCritter has its own constructor, and we need to override getActors method and processActors method. In getActors, we need to scan 24 locations which are two steps of the critter's current location, so I have a getTwoAdjacentLocations method which can scan the locations and if it is valid then return. In the method processActors, we count all critters around the BlusterCritter, and if determine darker the critter or brighter. In color method, I use hsb to manage color. The b is means brightness, its range is (0, 1), 0 means the darkest color, and 1 means the brightest color. We can control b to change the brightness of BlusterCritter.

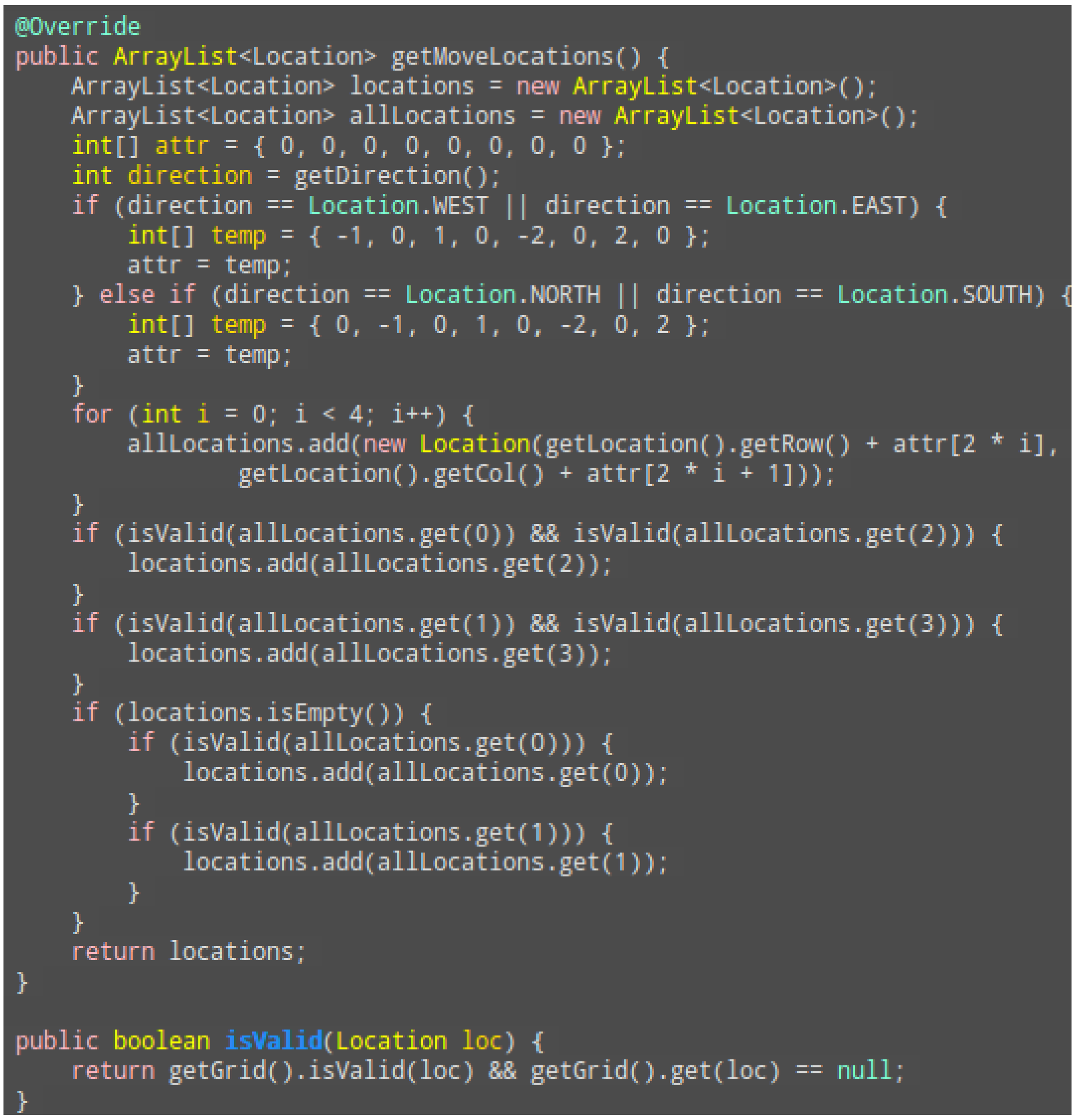
   (不使用 RGB 是因为如果直接是加一个常数，一个分量会加到 255 然后不能继续增加，这样会导致颜色不能恢复？)

   The core code is below:

```java
public class BlusterCritter extends Critter {

    private int courage;

    BlusterCritter(int cou) {
        super();
        courage = cou;
    }

    @Override
    public ArrayList<Actor> getActors() {

        Grid<Actor> grid = getGrid();
        ArrayList<Actor> actors = new ArrayList<Actor>();
        for (Location loc:getTwoAdjacentLocations(getLocation())) {
            Actor actor = grid.get(loc);
            if (actor != null) {
                actors.add(actor);
            }
        }
        return actors;
    }
    @Override
    public void processActors(ArrayList<Actor> actors) {

        int courageNum = 0;
        for (Actor actor : actors) {
            if (actor instanceof Critter) {
                courageNum++;
            }
        }
        if (courageNum < courage) {
            brighter();
        } else {
            darker();
        }
    }

    /**
     * make the color of critter darker
     */
    private void darker() {
        float color[] = Color.RGBtoHSB(getColor().getRed(),
                getColor().getGreen(), getColor().getBlue(), null);
        color[2] *= 0.95;
        Color newColor = Color.getHSBColor(color[0], color[1],
color[2]);
        setColor(newColor);
    }

    /**
     * make the color of critter brighter
     */
    private void brighter() {
        float color[] = Color.RGBtoHSB(getColor().getRed(),
                getColor().getGreen(), getColor().getBlue(), null);
        double temp = 1.0 - color[2];
        temp *= 0.95;
        color[2] = (float) (1 - temp);
        Color newColor = Color.getHSBColor(color[0], color[1],
color[2]);
        setColor(newColor);
    }

    /**
     * get the location list that is two step around critter
     */
    private ArrayList<Location> getTwoAdjacentLocations(Location
location) {
        Grid<Actor> grid = getGrid();
        ArrayList<Location> locs = new ArrayList<Location>();
        for (int i = location.getRow() - 2; i <= location.getRow()
+ 2; i++)
            for (int j = location.getCol() - 2; j <= location.getCol()
+ 2; j++) {
                if (!location.equals(new Location(i, j))
                        && grid.isValid(new Location(i, j))) {
                    locs.add(new Location(i, j));
                }
            }
        return locs;
    }
}
```

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.

   Answer: we need to override the method getMoveLocations, if left and right side two cells far is available, it can jump two cells, and it can random select a cell, or if it has only one side has two cells far is available, then it move to that cell, otherwise, it is act like CrabCritter. The core code is below:

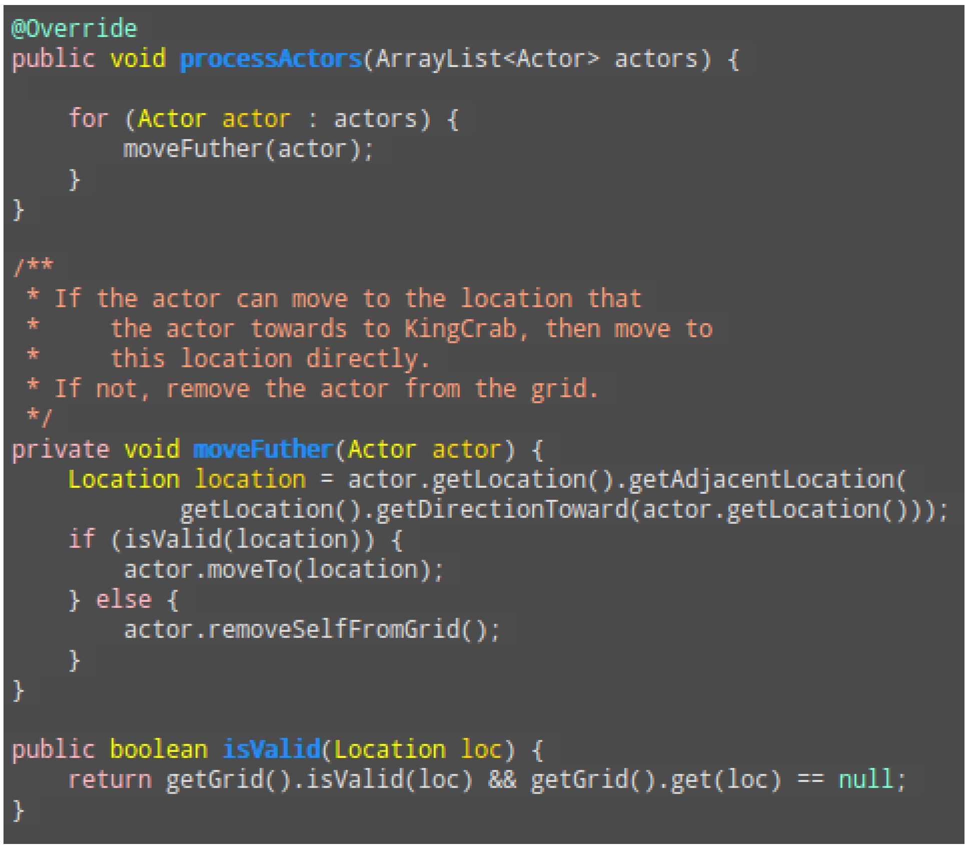


```java
@Override
public ArrayList<Location> getMoveLocations() {
    ArrayList<Location> locations = new ArrayList<Location>();
    ArrayList<Location> allLocations = new ArrayList<Location>();
    int[] attr = { 0, 0, 0, 0, 0, 0, 0, 0 };
    int direction = getDirection();
    if (direction == Location.WEST || direction == Location.EAST) {
        int[] temp = { -1, 0, 1, 0, -2, 0, 2, 0 };
        attr = temp;
    } else if (direction == Location.NORTH || direction == Location.SOUTH) {
        int[] temp = { 0, -1, 0, 1, 0, -2, 0, 2 };
        attr = temp;
    }
    for (int i = 0; i < 4; i++) {
        allLocations.add(new Location(getLocation().getRow() + attr[2 * i],
                getLocation().getCol() + attr[2 * i + 1]));
    }
    if (isValid(allLocations.get(0)) && isValid(allLocations.get(2))) {
        locations.add(allLocations.get(2));
    }
    if (isValid(allLocations.get(1)) && isValid(allLocations.get(3))) {
        locations.add(allLocations.get(3));
    }
    if (locations.isEmpty()) {
        if (isValid(allLocations.get(0))) {
            locations.add(allLocations.get(0));
        }
        if (isValid(allLocations.get(1))) {
            locations.add(allLocations.get(1));
        }
    }
    return locations;
}

public boolean isValid(Location loc) {
    return getGrid().isValid(loc) && getGrid().get(loc) == null;
}
```

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.

   Answer: The location which is in front, on left-front, or on right-front of the crab need to be empty. Just force moves the Actor to the direction which is toward to KingCrab, if it can move, remove it from the grid.

   The core code is below:



```
@Override
public void processActors(ArrayList<Actor> actors) {

    for (Actor actor : actors) {
        moveFuther(actor);
    }
}

/**
 * If the actor can move to the location that
 *     the actor towards to KingCrab, then move to
 *     this location directly.
 * If not, remove the actor from the grid.
 */
private void moveFuther(Actor actor) {
    Location location = actor.getLocation().getAdjacentLocation(
            getLocation().getDirectionToward(actor.getLocation()));
    if (isValid(location)) {
        actor.moveTo(location);
    } else {
        actor.removeSelfFromGrid();
    }
}

public boolean isValid(Location loc) {
    return getGrid().isValid(loc) && getGrid().get(loc) == null;
}
```