13331235 TanXiao

1 Part 4 - Set7

1.1 What methods are implemented in Critter?

act, getActors, processActors, getMoveLocations, selectMoveLocation, makeMove

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

getActors, processActors, getMoveLocations, selectMoveLocation, makeMove

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

It shouldn't be overrided unless the subclasses need to get actors from other locations.

1.4 Describe the way that a critter could process actors.

Remove it from the grid, change their color, make them move, and so on.

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

1. getMoveLocations method. The critter must use this method to find which locations it can move.

2. selectMoveLocation method. The critter should select one of locations get by the getMoveLocations method as it's moving target.

3. makeMove method. The critter has to use this method to move to the location selected by the selectMoveLocation method.

1.6 Why is there no Critter constructor?

The Critter class has no parameter or another option different from the Actor class when it need be constructed.

The Critter class extends the Actor class and doesn't have a constructor, so that Java will write a default constructor which will call the Actor's constructor for it.

2 Part4 - Set8

2.1 Why does act cause a ChameleonCritter to act differently from a Critter even though ChameleonCritter does not override act?

Because that the ChameleonCritter class overrode processActors method and makeMove method. The act method call these two methods, so that the ChameleonCritter doesn't need to override act.

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

When it calls super.makeMove, it calls Critter's makeMove. Obviously we can find that after the ChameleonCritter called the setDirection method, it moves like a Critter.

2.3 How would you make the ChameleonCritter drop flowers in its old location when it moves?

I will use a Location variable to store it's old location like the Bug class.

```
Location loc = getLocation();
setDirection(getLocation().getDirectionToward(loc));
super.makeMove(loc);
Flower flower = new Flower(getColor());
flower.putSelfInGrid(gr, loc);
```

2.4 Why doesn't ChameleonCritter override the getActors method?

Because that it has the same behavior like a Critter in the getActors method. That is, it has the same list of actors to process like the Critter class.

2.5 Which class contains the getLocation method?

Actor

2.6 How can a Critter access its own grid?

Use getGrid method.

3 Part4 – Set9

3.1 Why doesn't CrabCritter override the processActors method?

Because that it has the same behavior like a Critter in the processActors method. That is, it has the same behavior to process actors like the Critter class.

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

1. Use getActors method to get actors which are in front of it and to it's front direction's half-left and half-right. Obviously that the CrabCritter only find and eat other actors which are on it's front(include left-front and right-front).

2. Use the processActors method to eat them.

3.3 Why is the getLocationsInDirections method used in CrabCritter?

Use this method to find an array of locations which the CrabCritter can move to.

3.4 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?

(4, 3), (4, 4), (4, 5)

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

Similarities: These two classes both eat other actors. They don't need to change direction when they move. They both move randomly.

Differences: Crab only sidles but Critter can move to any it's neighbor locations if the location is valid. If there is no location to move, crab will turn it's direction to another

side but critter will not.

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

When the location to move is equal to it's current location, it will randomly turn left or right.

3.7 Why don't the CrabCritter objects eat each other?

This class inherits from the Critter class. That is, a CrabCritter is a Critter.

What else, CrabCritter's processActors method inherits from the Critter class and this method doesn't allow a critter to eat another critter.

4 Part4 - Exercises

4.1 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).

```
// Define a dark level like flower class
private static final double DARKENING_FACTOR = 0.33;


/**
 * Randomly selects a neighbor and changes this critter's color to be the
 * same as that neighbor's. If there are no neighbors, no action is taken.
 */
@Override
public void processActors(ArrayList<Actor> actors) {
   int n = actors.size();
   if (n == 0) {
      // if actors list is empty, make the color darker(like flower class)
      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());
}
```

4.2 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.

```
 * The ChameleonKid only select the front or behind neighbors to process, so
 * I should override the getActors method.
 *
 * @return a set of actors the kid need to process
 */
@Override
public ArrayList<Actor> getActors() {
   ArrayList<Actor> actors = new ArrayList<Actor>();
   int[] dirs = { Location.AHEAD, Location.HALF_CIRCLE };
   for (Location loc : getLocationsInDirections(dirs)) {
      Actor a = getGrid().get(loc);
      if (a != null) {
         actors.add(a);
      }
   }
   return actors;
}

/**
 * Finds the valid adjacent locations of this chameleonkid in different
 * directions.
 *
 * @param directions
 *           - an array of directions (which are relative to the current
 *           direction)
 * @return a set of valid locations that are neighbors of the current
 *         location in the given directions
 */
public ArrayList<Location> getLocationsInDirections(int[] directions) {
```

```
    ArrayList<Location> locs = new ArrayList<Location>();
    Grid gr = getGrid();
    Location loc = getLocation();
    for (int d : directions) {
      Location neighborLoc = loc.getAdjacentLocation(getDirection() + d);
      if (gr.isValid(neighborLoc)) {
        locs.add(neighborLoc);
      }
    }
    return locs;
}
```

4.3 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.

```
@Override
public void processActors(ArrayList<Actor> actors) {
    for (Actor a : actors) {
      if (a instanceof Rock) {
        a.removeSelfFromGrid();
      }
    }
}
```

4.4 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.

```
private int courage;

public BlusterCritter(int c) {
```

```
    courage = c;
}

/**
 * Gets the actors for processing. Implemented to return the actors that
 * occupy neighboring grid locations. Override this method in subclasses to
 * look elsewhere for actors to process.<br />
 * Postcondition: The state of all actors is unchanged.
 *
 * @return a list of actors that this critter wishes to process.
 */
@Override
public ArrayList<Actor> getActors() {
    // current location
    Location curLoc = getLocation();
    ArrayList<Actor> actors = new ArrayList<Actor>();
    // 内圈 8 个 cells
    actors = getGrid().getNeighbors(curLoc);
    // 内圈四个角的 neighbors, 涵盖整个两圈
    actors.addAll(getGrid().getNeighbors(
            new Location(curLoc.getRow() - 1, curLoc.getCol() - 1)));
    actors.addAll(getGrid().getNeighbors(
            new Location(curLoc.getRow() - 1, curLoc.getCol() + 1)));
    actors.addAll(getGrid().getNeighbors(
            new Location(curLoc.getRow() + 1, curLoc.getCol() - 1)));
    actors.addAll(getGrid().getNeighbors(
            new Location(curLoc.getRow() + 1, curLoc.getCol() + 1)));
    // 使用 Set 去重
    Set<Actor> actorSet = new HashSet<Actor>();
    for (Actor eachActor : actors) {
        // exclude itself
        if (eachActor != this) {
            actorSet.add(eachActor);
        }
    }
```

```
    // 最终返回 actors
    actors.clear();
    actors.addAll(actorSet);
    return actors;
}

/**
 * Changing this critter's color to be brighter or darker.
 */
@Override
public void processActors(ArrayList<Actor> actors) {
    // 选择 Critters
    int critCount = 0;
    for (Actor eachActor : actors) {
        if (eachActor instanceof Critter) {
            critCount++;
        }
    }
    if (critCount < courage) {
        setColor(brighter(getColor()));
    } else {
        setColor(darker(getColor()));
    }
}

/**
 * make the color brighter
 */
private Color brighter(Color c) {
    int red = c.getRed();
    int green = c.getGreen();
    int blue = c.getBlue();
    // if RGB = 254, it can't be added with 2.
    red += (red < 254) ? 2 : 0;
    green += (green < 254) ? 2 : 0;
```

```
    blue += (blue < 254) ? 2 : 0;
    return new Color(red, green, blue);
}

/**
 * make the color darker
 */
private Color darker(Color c) {
    int red = c.getRed();
    int green = c.getGreen();
    int blue = c.getBlue();
    // if RGB = 1, it can't be subed with 2.
    red -= (red > 1) ? 2 : 0;
    green -= (green > 1) ? 2 : 0;
    blue -= (blue > 1) ? 2 : 0;
    return new Color(red, green, blue);
}
```

4.5 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.

```
public QuickCrab() {
    setColor(Color.BLUE);
}

/**
 * Finds the valid two adjacent locations of this critter in different
 * directions.
 *
 * @param directions
 *            - an array of directions (which are relative to the current
 *            direction)
 * @return a set of valid locations that are neighbors of the current
 *         location in the given directions
 */
```

```
@Override
public ArrayList<Location> getLocationsInDirections(int[] directions) {
    ArrayList<Location> locs = new ArrayList<Location>();
    Grid gr = getGrid();
    // current location
    Location currentLoc = getLocation();
    // current direction
    int currentDir = getDirection();
    for (int d : directions) {
        Location firstLoc = currentLoc.getAdjacentLocation(currentDir + d);
        if (gr.isValid(firstLoc)) {
            // 相邻的 d 侧格子有效->加入 list
            locs.add(firstLoc);
            if (gr.get(firstLoc) == null) {
                // 相邻的下一个 d 侧格子有效->加入 list
                Location secondLoc = firstLoc
                        .getAdjacentLocation(currentDir + d);
                if (gr.isValid(secondLoc)) {
                    locs.add(secondLoc);
                }
            }
        }
    }
    return locs;
```

4.6 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.

```
public KingCrab() {
    setColor(Color.GREEN);
}

/**
 * Processes the elements of <code>actors</code>. New actors may be added to
```

```
 * empty locations. Implemented to "eat" (i.e. remove) selected actors that
 * are not rocks or critters. Override this method in subclasses to process
 * actors in a different way. <br />
 * Postcondition: (1) The state of all actors in the grid other than this
 * critter and the elements of <code>actors</code> is unchanged. (2) The
 * location of this critter is unchanged.
 *
 * @param actors
 *            the actors to be processed
 */
@Override
public void processActors(ArrayList<Actor> actors) {
    for (Actor a : actors) {
        if (!canMoveAway(a)) {
            a.removeSelfFromGrid();
        }
    }
}

/**
 * 判断目标位置是否距离 KingCrab 足够远,是则为 true
 *
 * @param target
 *            location
 * @return true or false
 */
public boolean ValidateLocation(Location loc) {
    // current location
    Location curLoc = getLocation();
    // 在外围第二圈意味着 loc 的 col 和 row 至少有一个与 curLoc 相差 2
    if (Math.abs(loc.getCol() - curLoc.getCol()) == 2
            || Math.abs(loc.getRow() - curLoc.getRow()) == 2) {
        return true;
    }
    return false;
```

```
}

/**
 * 判断目标能否移开,能则直接移动
 *
 * @param target
 *            actor
 * @return true or false
 */
public boolean canMoveAway(Actor actor) {
    ArrayList<Location> locs = getGrid()
            .getEmptyAdjacentLocations(actor.getLocation());
    for (Location loc : locs) {
        if (ValidateLocation(loc)) {
            actor.moveTo(loc);
            return true;
        }
    }
    return false;
```