实训报告

作业名称	Part 3: GridWorld Classes and Interfaces					
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 04 日		
QQ	736459905		E-mail	limkuan@mail2.sysu.edu.cn		

In our example programs, a grid contains actors that are instances of classes that extend the Actor class. There are two classes that implement the Grid interface: BoundedGrid and UnboundedGrid. Locations in a grid are represented by objects of the Location class. An actor knows the grid in which it is located as well as its current location in the grid.

The Location Class

Every actor that appears has a location in the grid. The Location class encapsulates the coordinates for an actor's position in a grid. It also provides methods that determine relationships between locations and compass directions.

Do You Know?

Assume the following statements when answering the following questions.

```
Location loc1 = new Location (4, 3);
Location loc2 = new Location (3, 4);
```

1. How would you access the row value for loc1?

   Answer: We can get row value for loc1 by calling the method getRow() like this:

   ```
   row = loc1.getRow();
   ```

2. What is the value of b after the following statement is executed?

   ```
   boolean b = loc1.equals(loc2);
   ```

   Answer: Obviously loc1 is not equals to loc2, so the value of b after this statement is executed is false.

3. What is the value of loc3 after the following statement is executed?

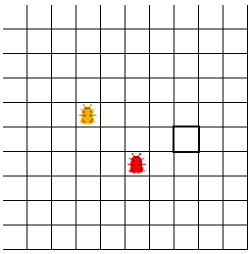
   ```
   Location loc3 = loc2.getAdjacentLocation(Location.SOUTH);
   ```

   Answer: loc3 is the position which is in the south of loc2, so the value of loc3 is (4, 4).

4. What is the value of dir after the following statement is executed?

   int dir = loc1.getDirectionToward(new Location(6, 5));

   Answer: In the grid below (the yellow bug is in loc1, and the red one is in (6, 5)):



   Obvious, we can know that the value of dir is 135

5. How does the getAdjacentLocation method know which adjacent location to return?

   Answer: The getAdjacentLocation method can only return 8 locations. Suppose that the location loc is (x, y), and call method by loc.getAdjacentLocation(parameter):
   If the parameter is Location.NORTH or 0, it returns the location that (x-1, y).
   If the parameter is Location.EAST or 90, it returns the location that (x, y+1).
   If the parameter is Location.SOUTH or 180, it returns the location that (x+1, y).
   If the parameter is Location.WEST or 270, it returns the location that (x, y-1).
   If the parameter is Location.NORTHEAST or 45, it returns the location that (x-1, y+1).
   If the parameter is Location.SOUTHEAST or 135, it returns the location that (x+1, y+1).
   If the parameter is Location.SOUTHWEST or 225, it returns the location that (x+1, y-1).
   If the parameter is Location.NORTHWEST or 315, it returns the location that (x-1, y-1).
   Others just find the closest direction and return the adjacent location.

The Grid Interface

The interface Grid<E> specifies the methods for any grid that contains objects of the type E. Two classes, BoundedGrid<E> and UnboundedGrid<E> implement the interface.

Do You Know?

1. How can you obtain a count of the objects in a grid? How can you obtain a count of the empty locations in a bounded grid?

   Answer: We can use method getOccupiedLocations to get a location list of all occupied locations in this grid. And just multiply row and column to get the number of location in the grid. Subtract them can get the number of empty location.

   We can call these methods like below:

```
num_occupied = grid.getOccupiedLocations().size();
num_empty = grid.getNumRows() * grid.getNumCols() - num_occupied;
```

2. How can you check if location (10, 10) is in a grid?

   Answer: Use isValid method. Call this method like this:

```
valid = grid.isValid(new Location(10, 10));
```

3. Grid contains method declarations, but no code is supplied in the methods. Why? Where can you find the implementations of these methods?

   Answer: Because Grid is just an interface, it doesn't need to implement these methods. And which class inherits the interface Grid need to implement these methods, in this condition, the classes are BoundedGrid and UnboundedGrid.

4. All methods that return multiple objects return them in an ArrayList. Do you think it would be a better design to return the objects in an array? Explain your answer.

   Answer: Because the array is fixed size, and the size of ArrayList can be changed. We don't know how many objects it will return, so we can use a fixed size array. On the other hand, ArrayList is a class include in java library, it has lots of functions and can make us control it more convenience than use an array.

The Actor Class

Do You Know?

1. Name three properties of every actor.

   Answer: The three properties of each actor are color, direction and location.

2. When an actor is constructed, what is its direction and color?

   Answer: We can see the document of the class actor or the source code of its constructed function, we know that initializing color is blue and initializing direction is NORTH.

```
/**
 * Constructs a blue actor that is facing north.
 */
public Actor()
{
    color = Color.BLUE;
    direction = Location.NORTH;
    grid = null;
    location = null;
}
```

3. Why do you think that the Actor class was created as a class instead of an interface?

   Answer: An interface in java is abstract, it can't have any implement of methods. But there have lots of methods of Actor is fixed, like the method setters and getters or other methods. The class which inherits Actor only needs to implement its own methods like act and doesn't need to implement all methods.

4. Can an actor put itself into a grid twice without first removing itself? Can an actor remove itself from a grid twice? Can an actor be placed into a grid, remove itself, and then put itself back? Try it out. What happens?

Answer: It has three sub questions:

✓ If an actor put itself into a grid twice without first removing itself:

```
public class testRunner{
    public static void main(String[] args) {

        BoundedGrid<Actor> grid = new BoundedGrid<Actor>(20, 20);
        Actor actor = new Actor();
        actor.putSelfInGrid(grid, new Location(10, 10));
        actor.putSelfInGrid(grid, new Location(5, 5));
        ActorWorld world = new ActorWorld(grid);
        world.show();
    }
}
```

And after compile, it will throw an IllegalStateException. So an actor can't put itself twice in the grid without first removing itself.

```
Exception in thread "main" java.lang.IllegalStateException: This actor is already contained in a grid.
        at info.gridworld.actor.Actor.putSelfInGrid(Actor.java:118)
        at testRunner.main(testRunner.java:20)
```

✓ If an actor remove itself from a grid twice:

```
public class testRunner{
    public static void main(String[] args) {

        BoundedGrid<Actor> grid = new BoundedGrid<Actor>(20, 20);
        Actor actor = new Actor();
        actor.putSelfInGrid(grid, new Location(10, 10));
        actor.removeSelfFromGrid();
        actor.removeSelfFromGrid();
        ActorWorld world = new ActorWorld(grid);
        world.show();
    }
}
```

And after compile, it will throw an IllegalStateException. So an actor can't remove itself from a grid twice.

```
Exception in thread "main" java.lang.IllegalStateException: This actor is not contained in a grid.
        at info.gridworld.actor.Actor.removeSelfFromGrid(Actor.java:136)
        at testRunner.main(testRunner.java:21)
```

✓ If an actor be placed into a grid, remove itself, and then put itself back:

```
public class testRunner{
    public static void main(String[] args) {

        BoundedGrid<Actor> grid = new BoundedGrid<Actor>(20, 20);
        ActorWorld world = new ActorWorld(grid);
        Actor actor = new Actor();
        world.add(new Location(10, 10), actor);
        actor.removeSelfFromGrid();
        actor.putSelfInGrid(grid, new Location(5, 5));
        world.show();
    }
}
```

The program is run correctly, so an actor can be placed into a grid, remove it, and then put it back

5. How can an actor turn 90 degrees to the right?

   Answer: Just use the method setDirction like below:

```
actor.setDirection(actor.getDirection() + Location.RIGHT);
```

Extending the Actor Class

The Bug, Flower, and Rock classes extend Actor in different ways. Their behavior is specified by how they override the act method.

- **The Rock Class**

  A rock acts by doing nothing at all. The act method of the Rock class has an empty body.

- **The Flower Class**

  A flower acts by darkening its color, without moving. The act method of the Flower class reduces the values of the red, green, and blue components of the color by a constant factor.

- **The Bug Class**

  A bug acts by moving forward and leaving behind a flower. A bug cannot move into a location occupied by a rock, but it can move into a location that is occupied by a flower, which is then removed. If a bug cannot move forward because the location in front is occupied by a rock or is out of the grid, then it turns right 45 degrees.

Do You Know?

1. Which statement(s) in the canMove method ensures that a bug does not try to move out of its grid?

   Answer: In the statements show below:

```
if (!gr.isValid(next))
    return false;
```

   If the Bug is moving out of the grid, its location is invalid.

2. Which statement(s) in the canMove method determines that a bug will not walk into a rock?

   Answer: In the statements show below:

```
return (neighbor == null) || (neighbor instanceof Flower);
```

   If the bug is walking into a rock, the location neighbor is rock and not equal null, obviously rock is not instance of flower, so if neighbor is rock, this statement will return false, and the bug can't move.

3. Which methods of the Grid interface are invoked by the canMove method and why?

Answer: In the source code, we can see that:

```
if (!gr.isValid(next))
    return false;
Actor neighbor = gr.get(next);
return (neighbor == null) || (neighb
```

The method isValid and get is invoked by the canMove method. Because we need use isValid method to ensure that next location is in the grid. And use get method to get the object in next location to ensure that next location is empty or only has flower.

4. Which method of the Location class is invoked by the canMove method and why?

Answer: In the source code, we can see that:

```
Location loc = getLocation();
Location next = loc.getAdjacentLocation(getDirection());
```

The method getAdjacentLocation is invoked by canMove method. Because we use the method getAdjacentLocation get next location which the bug towards to.

5. Which methods inherited from the Actor class are invoked in the canMove method?

Answer: In the source code, we can see that:

```
Grid<Actor> gr = getGrid();
if (gr == null)
    return false;
Location loc = getLocation();
Location next = loc.getAdjacentLocation(getDirection());
```

The method getGrid, getLocation and getDirection are invoked in canMove method.

6. What happens in the move method when the location immediately in front of the bug is out of the grid?

Answer: In the source code, we can see that:

```
Location next = loc.getAdjacentLocat
if (gr.isValid(next))
    moveTo(next);
else
    removeSelfFromGrid();
```

It will remove itself from the grid.

7. Is the variable loc needed in the move method, or could it be avoided by calling getLocation() multiple times?

Answer: Yes, the variable loc is needed in the move method. Because we need to know which location the bug is before moving, and put a flower into the location. It can avoided by calling getLocation() multiple times.

8. Why do you think the flowers that are dropped by a bug have the same color as the bug?

Answer: Yes, because while put a flower into grid, the color of flower is using the method getColor for bug, so there color are the same. And the same color with a bug and its flower, we can distinguish difference bug's path more convenience.

9. When a bug removes itself from the grid, will it place a flower into its previous location?

Answer: A bug remove itself from the grid, it will place a flower into its previous location. Because of the source code:

```
if (gr.isValid(next))
    moveTo(next);
else
    removeSelfFromGrid();
Flower flower = new Flower(getColor());
flower.putSelfInGrid(gr, loc);
```

After the if-else statement, the method put flower in grid is be call, so whether the bug is remove from grid, the flower is here.

10. Which statement(s) in the move method places the flower into the grid at the bug's previous location?

Answer: In the source code, we can see that:

```
Flower flower = new Flower(getColor());
flower.putSelfInGrid(gr, loc);
```

This is the statements that place the flower into grid.

11. If a bug needs to turn 180 degrees, how many times should it call the turn method?

Answer: Obviously, call the turn method once time bug can turn 45 degrees, so while call four times method turn the bug can turn 180 degrees.