Part 3:GridWorld Classes and Interfaces

Set 3
1.loc1.getRow()
2.false
3.(4,4)
4.135(degrees)
5.The parameter in getAdjacentLocation indicate the direction, and It returns the adjacent location which is closest to the original location in the compass direction.

Set 4
1.Assume gr is an instance of Grid
   gr.getOccupiedLocation().size() is the number of the objects in this grid.
   gr.getNumRows() * gr.getNumCols() - gr.getOccupiedLocation().size() is the number of the empty location in a bounded gird.
2.gr.isValid(10,10)
3.Because Grid is an interface, it only supply some methods to be implement and don't need to have specific code for these methods.And we can find the implementations of these methods in the AbstractGrid and the BoundedGrid and the UnboundedGrid classes.
4.An ArrayList does not need to know its' size before we filling it, but an array does.So if we first keep track of the number of occupied locations, it's also easy to filling with an array.But using an ArrayList is more convenient than an array at most time.

Set 5
1.Color, direction and location.
2.initial color is blue and initial direction is North.
3.Because an actor has both state and behavior, and an interface does not allow to declare instance or implement methods.
4.An actor can't put itself into a grid twice without first removing itself, and it will throw an IllegalStateException(“This actor is already contained in a grid”)
   An actor can't remove itself from a grid twice, and it will throw an IllegalStateException(“This actor is not contained in a grid”)
   An actor can be placed into a grid, remove itself, and then put itself back.It is ok.
5.using method setDirection(getDirection() + Location.RIGHT) or setDirection(getDirection() + 90)

Set 6
1.if (!gr.isValid(next))
```
        return false;
```
2.Actor neighbor = gr.get(next);
```
        return (neighbor == null) || (neighbor instanceof Flower);
```
3.getGrid() and isValid().To ensure the next location is a valid location and the location is empty or contain an actor that can be replaced by the bug.
4.getAdjacentLocation().To find the possible locations for next step.
5.getLocation(), getDirection() and getGrid().
6.The bug will remove itself from the grid.
7.Yes, because loc stores the bug's location so that we can insert a flower in the bug's old location.

8.Yes,the color is same.
```
   Flower flower = new Flower(getColor());
   flower.putSelfInGrid(gr, loc);
```
9.If just call removeSelfFromGrid method, no.But when the removeSelfFromGrid is called in the mov method,yes.

10.
```
Flower flower = new Flower(getColor());
     flower.putSelfInGrid(gr, loc);
```
11.four times.