# 13331231 孙圣 Part3

**Set3**

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?

loc1.getRow();

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

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

b = 0

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

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

loc3 has row 4, column 4.

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

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

dir = 135

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

This method receives a location compass direction as parameter, so it first gets the location of the object, then based on the direction, it decides the adjacent location ( e.g. if the object is located at (3, 4), receiving SOUTH as the parameter, then it increases the row number by one and gets the answer (4, 4) ).

**Set4**

**Suppose the instance of the grid is *grid***

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?

   To get the count of objects:

   ```
                   grid.getOccupiedLocations().size()
   ```

   To get the count of empty locations:

   ```
   grid.getNumRows() * grid.getNumCols() -
   grid.getOccupiedLocations().size()
   ```

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

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

   If it returns true, then location(10, 10) is valid; otherwise it is invalid.

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

   Grid is a interface and there are three classes that implements it, so there is no need to put the code in the method.
   The implementations are found in AbstractGrid, BoundedGrid as well as 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.

   No. If we want to use array to store the object returned, we need to know the number of objects in advance, otherwise we have to make a large array, which wastes a lot of space. But if we use ArrayList, its size can dynamically change, so we don't need to worry about the size at the beginning. But accessing ArrayList is quite tedious, since we must use get() method. Overall, the advantages outweigh the disadvantages, so I think it is better to use ArrayList rather than simply using array.

**Set5**

1. Name three properties of every actor.

   Location, direction and color.

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

   By default, its color is blue and the direction is north.

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

   Because many classes have the same methods, such as putSelfInGrid, moveTo etc. If we treat Actor as an interface, we need to copy those code for many times, which produces duplicates.

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?

   No, if we try to move an actor that already stays in grid, there will be “IllegalStateException: This actor is already contained in a grid. ”.

   No, we will get “IllegalStateException: This actor is not contained in a grid. ”

   Yes, we can do this successfully.

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

   *actor.setDirection(actor.getDirection() + 90);*

**Set6**

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

   *Location next = loc.getAdjacentLocation(getDirection());*
   *if (!gr.isValid(next))*
   *return false;*

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

   *Actor neighbor = gr.get(next);*
   *return (neighbor == null) || (neighbor instanceof Flower);*

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

   g*etGrid();* is used to get the grid. If grid is null, then the bug is not in the grid, so it cannot move; secondly, it uses the method *isValid();* to decide whether the next location is valid or not. If it is invalid, the bug cannot move either.

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

   getAdjacentLocation(); receiving a direction parameter, returns the adjacent location of the bug, which helps decides whether the bug can move forward or not.

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

   *getGrid(); getLocation();* and *getDirection();*

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

   The bug calls *removeSelfFromGrid();* and it is no longer in the grid.

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

   Yes. loc facilitates getting the adjacent location of the bug and putting the flower into the grid. If it is not used, we have to write getLocation() many times in this method.

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

   *Flower flower = new Flower(getColor());*
   Because while creating the flower, it first gets the color of the bug and passes the color to the constructor of the flower.

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

   It depends.
   If the bug is removed because of out of grid, it still places a flower behind.
   If the bug is removed by calling *removeSelfFromGrid();* directly. Definitely, it will not leave a flower.

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

    *Location loc = getLocation();*
    *Flower flower = new Flower(getColor());*
    *flower.putSelfInGrid(gr, loc);*

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

    Each time it calls the turn method, it turns 45 degrees, so 4 times are needed to turn 180.