Do you know:

Set3.

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

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

Answer: loc1.getRow()

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

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

Answer: false

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

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

Answer: Location(4, 4)

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

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

Answer: 135(degrees)

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

Answer: The parameter of getAdjacentLocation method gives a direction of the adjacent neighbor to find. With this parameter, the method find the adjacent location in compress direction which is closest to the given direction.

Set4:

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 the getOccupiedLocations method to get a list of all objects in a gird. And the size of the list is equal to the count of the objects in the gird. The code is as follows:

```
int count = gird.getOccupiedLocations().size();
```

We can first find the number of locations of the gird and then subtract the number of occupied locations. The code is as follows:

```
int emptyCount = grid.getNumRows() * grid.getNumCols() - count;
```

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

Answer: We can use the isValid method. This method returns whether the given location is a valid location . The code is as follows:

```
boolean isLocationValid = gird.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: Grid is an interface. In java, an interface declare the methods that the class need to implement. So, the implement of these method is in the class which implement the

interface.   Like AbstractGrid, BoundGrid and UnBoundGrid classes (AbstractGrid not implements all methods of Grid).

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: ArrayList can grow dynamically. In the implements of these method, we don't know the count of objects of the grid. If we use an Array, we should first traversal the grid to get the number of objects of the grid. Which is inefficient. With an ArrayList, we can use add method to add elements to the list.
So, use ArrayList will be better.

Set 5:
1. Name three properties of every actor.

Answer: color, direction, location.

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

Answer: An actor's initial direction is North and its initial color is blue.

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

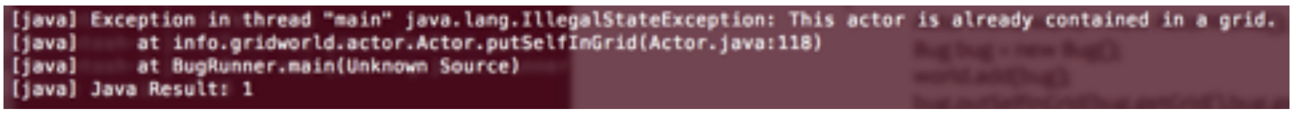
Answer: An actor has its own behavior like setColor and has its own state like its location. An interface can't implements its method

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: No. If an actor is already in a gird, it can't put itself into the grid again. I use the test code as follows:

```
public class ActorRunner
{
        public static void main(String[] args)
         {
                ActorWorld world = new ActorWorld();
                Actor actor = new Actor();
                world.add(actor);
                actor.putSelfInGrid(actor.getGrid(), actor.getLocation());
                world.show();
        }
}
```

And I get wrong message as follows:

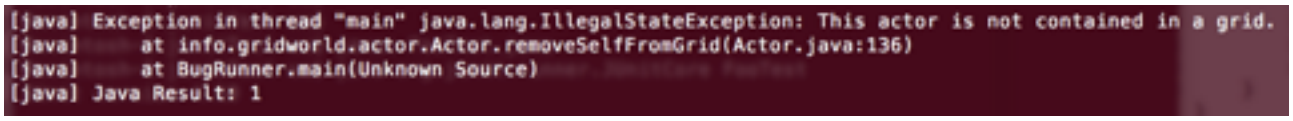


```
[java] Exception in thread "main" java.lang.IllegalStateException: This actor is already contained in a grid.
[java]      at info.gridworld.actor.Actor.putSelfInGrid(Actor.java:118)
[java]      at BugRunner.main(Unknown Source)
[java] Java Result: 1
```

We can the wrong message is: This actor is already contained in a grid.
//— — — — — — —
No. The removeSelfFromGrid method removes the actor from its grid and makes the actor's grid and location both null. After remove, there isn't exist this actor anymore. The test code is as follows:

```
public class ActorRunner
{
        public static void main(String[] args)
        {
                ActorWorld world = new ActorWorld();
                Actor actor = new Actor();
                world.add(actor);
                actor.removeSelfFromGrid();
                actor.removeSelfFromGrid();
                world.show();
        }
    }
```

And I get wrong message as follows:



```
[java] Exception in thread "main" java.lang.IllegalStateException: This actor is not contained in a grid.
[java]        at info.gridworld.actor.Actor.removeSelfFromGrid(Actor.java:136)
[java]        at BugRunner.main(Unknown Source)
[java] Java Result: 1
```

We can see the wrong message is: This actor is not contained in a grid.
//———————
Yes. But we should get the Grid and location of the actor so that we can put it back.
The test code is as follows:

```
    public class BugRunner
    {
        public static void main(String[] args)
        {
                ActorWorld world = new ActorWorld();
                Actor actor = new Actor();
                world.add(actor);

                Grid<Actor> grid = actor.getGrid(); // Get the Grid of actor
                Location location = actor.getLocation(); // Get the location of actor
                actor.removeSelfFromGrid();
                actor.putSelfInGrid(grid, location);
                world.show();
        }
    }
```

And the program works well.

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

Answer: We can use the setDirection method. First get the original direction and add 90 degrees. Then set the direction of the actor with this angle. The code is as follows:

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

Set6:

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

Answer: Use isValid method. The code is as follows:

```
    if (!grid.isValid(loc))
            return false;
```

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

Answer: First get the member in front of the bug. Then determined whether it is a Rock. The code is as follows (If it's a Rock, is not null or a flower, only it is null or flower the bug can move to):

```
        Actor neighbor = grid.get(nextLoc);
        return (neighbor == null) || (neighbor instanceof Flower);
```

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

Answer: isValid and get. We can found the reason in exercise 1 and 2. We need isValid method to ensures a bug not try to move out of grid and get method to determines whether the next location is null or has a flower, if not, the bug can not move to.

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

Answer: getAdjacentLocation method. This method returns a valid location if the location in front of the bug is a valid location. Or return null. When bug need to know whether the next location can move to, it needs to get the information of the location in front of it;

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

Answer: getGrid, getDirection and getLocation.

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

Answer: The bug will remove itself from the grid. I rewrite the can move method, like this:

```
        public boolean canMove() {
                return true;
        };
```

And there is only a bug in grid. First I make the grid have 10 rows and 10 columns. When bug is run out of the grid, I make the grid 100*100, but there is no bug in the grid. So the bug remove itself from grid.

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

Answer: Yes. After the bug move, move method will put a flower in its previous location. So move method need loc to store the previous location. The getLocation method can only get the current location of bug.

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

Answer: Maybe it is because we can find the path of the bug more easily. Because with the flower we can easily know the location the bug has moved to.

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

Answer: No. When bug remove itself from grid, it call removeSelfFromGrid method. And the removeSelfFromGrid method is inherit from Actor. In removeSelfFromGrid, there is no code to put flower.

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

Answer:

```
grid = bug.getGrid();
loc = bug.getLocation();
Flower flower = new Flower();
flower.putSelfInGrid(grid, loc);
```

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

Answer: 180 / 45 = 4. So it need to call turn method four times. Each turn is 45 degrees.