1.What is the role of the instance variable sideLength?

The instance variable sideLength defines the max number of a bug can move on each side.

2.What is the role of the instance variable steps?

The instance variable steps counts the step's number of a bug that has moved on the current side.

3.Why is the turn method called twice when steps becomes equal to sideLength?

When steps becomes equal to sideLength, the bug should change it direction, because the turn method only turn the bug 45 degrees, call it twice to turn 90 degrees.

4.Why can the move method be called in the BoxBug class when there is no move method in the BoxBug code?

Because the BoxBug class is extended from the Bug class, the Bug class has the move method, so the BoxBug class inherits the move method from Bug class.

5.After a BoxBug is constructed, will the size of its square pattern always be the same? Why or why not?

After a BoxBug is constructed, the side length is determine,so the size of its square pattern always be the same.

6.Can the path a BoxBug travels ever change? Why or why not?

Yes. If there is a rock or an actor in front of it, it will change its direction 45 degrees to the right and start a new path.

7.When will the value of steps be zero?

At first, the value of steps is set to zero by the constructor, after that, the value of steps will be set to zero only when the steps equal to the sideLength or the bug can not move.

1.Write a class CircleBug that is identical to BoxBug, except that in the act method the turn method is called once instead of twice. How is its behavior different from a BoxBug?

```
import info.gridworld.actor.Bug;

// CircleBug Class
// extend from Bug
public class CircleBug extends Bug
{
    private int steps;
    private int sideLength;
    /**
     * @param length the side length
     */
    public CircleBug(int length)
    {
        // initialize the steps and sideLength
        steps = 0;
        sideLength = length;
    }

    /**
     * Moves to the next location.
     */
    public void act()
    {
        // try to move to next location
        if (steps < sideLength && canMove())
        {
            move();
            steps++;
        }
        else
        {
            // turn only once, 45 degrees
            turn();
            steps = 0;
        }
    }

}
```

It is different from a BoxBug, because the bug will run as a octagon path when run it.

2.Write a class SpiralBug that drops flowers in a spiral pattern. Hint: Imitate BoxBug, but adjust the side length when the bug turns. You may want to change the world to an UnboundedGrid to see the spiral pattern more clearly.

```
import info.gridworld.actor.Bug;

// SpiralBug Class
// extend from Bug
public class SpiralBug extends Bug
{
    private int steps;
    private int sideLength;

    /**
     * @param length the side length
     */
    public SpiralBug(int length)
    {
        // initialize the steps and sideLength
        steps = 0;
        sideLength = length;
    }

    /**
     * Moves to the next location.
     */
    public void act()
    {
        // try to move to next location
        if (steps < sideLength && canMove())
        {
            move();
            steps++;
        }
        else
        {
            // adjust the side length when the bug turns
            sideLength++;
            turn();
            turn();
            steps = 0;
        }
    }
}
```

3.Write a class *ZBug* to implement bugs that move in a "Z" pattern, starting in the top left corner. After completing one "Z" pattern, a *ZBug* should stop moving. In any step, if a *ZBug* can't move and is still attempting to complete its "Z" pattern, the *ZBug* does not move and should not turn to start a new side. Supply the length of the "Z" as a parameter in the constructor. The following image shows a "Z" pattern of length 4. Hint: Notice that a *ZBug* needs to be facing east before beginning its "Z" pattern.

```
 */
import info.gridworld.actor.Bug;

// ZBug Class
// extend from Bug
public  class ZBug extends Bug
{
    private int steps;
    private int sideLength;
    private int moveTime;

    // define the constant
    private static final int EAST = 90;
    private static final int SOUTHWEST = 225;
    private static final int ZPATTERNLENGTH = 4;
    private static final int MAXMOVETIME = 3;
    private static final int FIRSTTURN = 1;
    private static final int SECONDTURN = 2;

    /**
     * @param length the side length
     */
    public ZBug()
    {
        steps = 0;
        sideLength = ZPATTERNLENGTH;
        moveTime = 0;
        // at first it should face east
        setDirection(EAST);
    }

    /**
     * Moves to the next location of the square.
     */
    public void act()
    {
        // if complete the Z pattern or ZBug can't move it stop
        if (moveTime >= MAXMOVETIME || (!canMove() && steps != ZPATTERNLENGTH))
        {
            return;
        }
        // move to next location
        if (steps < sideLength)
        {
            move();
            steps++;
            if (steps == ZPATTERNLENGTH && moveTime != MAXMOVETIME)
            {
                moveTime++;
            }
        }
        else
        {
            // complete the first '-'
            if (moveTime == FIRSTTURN)
            {
                setDirection(SOUTHWEST);
            }
            // complete the '/'
            if (moveTime == SECONDTURN)
            {
                setDirection(EAST);
            }
            // start a new way
            steps = 0;
        }
    }
}
```

4.Write a class *DancingBug* that "dances" by making different turns before each move. The *DancingBug* constructor has an integer array as parameter. The integer entries in the array represent how many times the bug turns before it moves. For example, an array entry of 5 represents a turn of 225 degrees (recall one turn is 45 degrees). When a dancing bug acts, it should turn the number of times given by the current array entry, then act like a Bug. In the next move, it should use the next entry in the array. After carrying out the last turn in the array, it should start again with the initial array value so that the dancing bug continually repeats the same turning pattern.
The DancingBugRunner class should create an array and pass it as a parameter to the DancingBug constructor.

```
import info.gridworld.actor.Bug;
import java.util.Arrays;

// DancingBug Class
// extend from Bug
// making different turns before each move.
public class DancingBug extends Bug
{
    private int turnIndex;
    private int[] turnOrder;
    /**
     * @param turnList is the turn Order
     */
    public DancingBug(int[] turnList)
    {
        // initialize the turnIndex
        turnIndex = 0;
        // get turnOrder from the parameter
        if(turnList == null) {
            this.turnOrder = new int[0];
        } else {
            this.turnOrder = Arrays.copyOf(turnList, turnList.length);
        }
    }

    /**
     * Moves to the next location.
     */
    public void act()
    {
        // turn turnOrder[turnIndex] times
        for (int i = 0; i < turnOrder[turnIndex]; i++) {
            turn();
        }
        turnIndex++;
        // After carrying out the last turn in the array
        //it start again with the initial array value
        if (turnIndex >= turnOrder.length) {
            turnIndex = 0;
        }
        // move to next location
        if (canMove())
        {
            move();
        }
    }
}
```

5.Study the code for the BoxBugRunner class. Summarize the steps you would use to add another BoxBug actor to the grid.

(1)Creat a BoxBug object by pass a parameter sidelength to it.
(2)Add this BoxBug object to the ActorWorld by pass the BoxBug object or pass a location and the BoxBug object to the object actorworld's add method.

code:
```
BoxBug newOne = new BoxBug(5);
world.add(newOne);
or
world.add(new Location(5, 5), newOne);
```