Part 2
Set 2
1.sideLength is used to record the each-side-length that the BoxBug moves.

2.steps is used to record how many steps the BoxBug has moved in this side.

3.Because when steps equal to sideLength, it must move out one side of the box, so it must turn twice to the other side.

4.Because BoxBug extends Bug, so class BoxBug can call the move method in the Bug.

5.No, because every BoxBug locate at different location.If some BoxBugs has moved to the edge of the box, it will change its square pattern.

6.Yes.When there is an actor in front of its path, the path will be changed.

7.When the bug move to the corner of the box, the steps will be set zero.

Exercise 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.
*/

/*   CircleBug.java   */
import info.gridworld.actor.Bug;
public class CircleBug extends Bug {
    private int steps;
    private int sideLength;
    public CircleBug(int length) {
        steps = 0;
        sideLength = length;
    }
    public void act() {
        if (steps < sideLength && canMove()) {
            move();
            steps++;
        } else {
            turn();
            steps = 0;
        }
    }
}
```

```
/*    CircleBugRunner.java    */
import info.gridworld.actor.ActorWorld;
import info.gridworld.grid.Location;
import java.awt.Color;
public class CircleBugRunner {
        private static final int SIX = 6;
        private static final int THREE = 3;
        private static final int ONE = 1;
        private static final int FIVE = 5;
        public static void main(String[] args) {
                ActorWorld world = new ActorWorld();
                CircleBug alice = new CircleBug(SIX);
                alice.setColor(Color.ORANGE);
                CircleBug bob = new CircleBug(THREE);
                world.add(new Location(ONE, ONE), alice);
                world.add(new Location(FIVE, FIVE), bob);
                world.show();
        }
}
// In CircleBug, the bug will move in a octagon path.
```

Exercise 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.

*/

```
/*    SpiralBug.java    */
import info.gridworld.actor.Bug;
public class SpiralBug extends Bug {
        private int steps;
        private int sideLength;
        public SpiralBug(int length) {
                steps = 0;
                sideLength = length;
        }
        public void act() {
                if (steps < sideLength && canMove()) {
                        move();
                        steps++;
                } else {
```

```
                turn();
                turn();
                steps = 0;
                sideLength++;
            }
        }
}

/*    SpiralBugRunner.java    */
import info.gridworld.actor.ActorWorld;
import info.gridworld.grid.Location;
import java.awt.Color;
public class SpiralBugRunner {
        private static final int SIX = 6;
        private static final int THREE = 3;
        private static final int ONE = 1;
        private static final int FIVE = 5;
        public static void main(String[] args) {
            ActorWorld world = new ActorWorld();
            SpiralBug alice = new SpiralBug(SIX);
            alice.setColor(Color.ORANGE);
            SpiralBug bob = new SpiralBug(THREE);
            world.add(new Location(ONE, ONE), alice);
            world.add(new Location(FIVE, FIVE), bob);
            world.show();
        }
}
```

Exercise 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.

*/

```
/*    ZBug.java    */
import info.gridworld.actor.Bug;
import info.gridworld.grid.Location;
public class ZBug extends Bug {
        private int steps;
```

```
    private int sideLength;
    private int side;
    private static final int THREE = 3;
    public ZBug(int length) {
        setDirection(Location.EAST);
        sideLength = length;
        steps = 0;
        side = 1;
    }
    public void act() {
        if (steps < sideLength && side <= THREE) {
            if (canMove()) {
                move();
                steps++;
            }
        } else if (side == 1) {
            setDirection(Location.SOUTHWEST);
            steps = 0;
            side++;
        } else if (side == 2) {
            setDirection(Location.EAST);
            steps = 0;
            side++;
        }
    }
}

/*  ZBugRunner.java  */
import info.gridworld.grid.Location;
import java.awt.Color;
public class ZBugRunner {
    private static final int FOUR = 4;
    private static final int THREE = 3;
    private static final int ONE = 1;
    private static final int FIVE = 5;
    public static void main(String[] args) {
        ActorWorld world = new ActorWorld();
        ZBug alice = new ZBug(FOUR);
        alice.setColor(Color.ORANGE);
        ZBug bob = new ZBug(FOUR);
        world.add(new Location(ONE, ONE), alice);
        world.add(new Location(FIVE, FIVE), bob);
        world.show();
    }
```

}

Exercise 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 aparameter to the DancingBug constructor.

*/

```
/*    DancingBug.java    */
import info.gridworld.actor.Bug;
public class DancingBug extends Bug {
    private int steps;
    private int[] dance;
    public DancingBug(int[] array) {
        steps = 0;
        if (array == null) {
        dance = new int[0];
        } else {
        dance = (int[]) array.clone();
        }
    }
    public void act() {
        if (steps == dance.length) {
            steps = 0;
        } else {
            for (int i = 0; i < dance[steps]; i++) {
                turn();
            }
            steps++;
            if (canMove()) {
                move();
            } else {
                turn();
            }
        }
    }
}

/*  DancingBugRunner.java  */
import info.gridworld.actor.ActorWorld;
import info.gridworld.grid.Location;
import java.awt.Color;
public class DancingBugRunner {
    private static final int ONE = 1;
    public static void main(String[] args) {
        int[] array = new int[]{1, 1, 2, 1};
        ActorWorld world = new ActorWorld();
        DancingBug alice = new DancingBug(array);
        alice.setColor(Color.ORANGE);
        world.add(new Location(ONE, ONE), alice);
        world.show();
    }
}
```

Exercise 5

Steps to add another BoxBug actor to the grid:

1. Create a BoxBug and declare its side length;

/* BoxBug alice = new BoxBug(6); */

2. Add the BoxBug to the world and decalre its location.

/* world.add(new Location(7, 8), alice); */