13331235 TanXiao

1 Part 2 – Questions

1.1 What is the role of the instance variable sideLength?

The instance variable sideLength defines the max number of steps that a bug can move on each side. That is, sideLength defines the box's length of side.

1.2 What is the role of the instance variable steps?

The instance variable steps records how many steps the bug has moved on the current side (from the current side's starting point to the bug's last location on this side).

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

When steps becomes equal to sideLength, the bug has to change it's direction(turn right 90°). However, the turn method only turn right 45°, so that the bug has to call the turn method twice to turn right 90°.

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

The BoxBug class extends the Bug class, and the Bug class has a public method: move. The BoxBug class will inherit the move method from the Bug class because it is a subclass of the Bug class.

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

Yes. When we construct a BoxBug, we have to determine it's sideLength and we can't change it later.

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

Yes. If there is another Actor in front of the BoxBug and the BoxBug tries to move, it will turn and start a new box path.

1.7 When will the value of steps be zero?

1. The BoxBug is constructed.
2. The BoxBug's steps is equal to sideLength(current side has completed).
3. The BoxBug turns to start a new box path.

2 Part2 – Exercises

2.1 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?

The CircleBug turn right 45° when it's steps is equal to sideLength, so that the path of it is an octagon.

1) CircleBug class

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

public class CircleBug extends Bug {
   private int steps;
   private int sideLength;

   /**
    * Constructs a box bug that traces an octagon of a given side length
    *
    * @param length
    *           the side length
    */
   public CircleBug(int length) {
      steps = 0;
      sideLength = length;
   }

   /**
    * Moves to the next location of the octagon.
    */
   @Override
   public void act() {
      if (steps < sideLength && canMove()) {
         move();
         steps++;
      } else {
         turn();
         steps = 0;
      }
   }
}
```

2)CircleBugRunner class

```
import java.awt.Color;
```

```
import info.gridworld.actor.ActorWorld;
import info.gridworld.grid.Location;

/**
 * This class runs a world that contains circle bugs. <br />
 * This class is not tested on the AP CS A and AB exams.
 */
public final class CircleBugRunner {
   public static void main(String[] args) {
      ActorWorld world = new ActorWorld();
      CircleBug alice = new CircleBug(6);
      alice.setColor(Color.ORANGE);
      CircleBug bob = new CircleBug(3);
      world.add(new Location(7, 8), alice);
      world.add(new Location(5, 5), bob);
      world.show();
   }

   private CircleBugRunner() {
      // never call
   }
}
```

2.2 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.

1. SpiralBug class

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

public class SpiralBug extends Bug {
   private int steps;
   private int sideLength;

   /**
    * Constructs a spiral bug with a given initial side length
    *
```

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

    /**
     * Moves to the next location of the spiral.
     */
    @Override
    public void act() {
        if (steps < sideLength && canMove()) {
            move();
            steps++;
        } else {
            sideLength++;
            turn();
            turn();
            steps = 0;
        }
    }
}
```

2. SpiralBugRunner class

```
import info.gridworld.actor.Actor;
import info.gridworld.actor.ActorWorld;
import info.gridworld.grid.Location;
import info.gridworld.grid.UnboundedGrid;

/**
 * This class runs a world that contains spiral bugs. <br />
 * This class is not tested on the AP CS A and AB exams.
 */
public final class SpiralBugRunner {
```

```
    public static void main(String[] args) {
        ActorWorld world = new ActorWorld();
        SpiralBug spiralDemo = new SpiralBug(3);
        world.setGrid(new UnboundedGrid<Actor>()); // 设置无边界
        world.add(new Location(19, 16), spiralDemo); // 为了显示效果,只设置一只 bug
        world.show();
    }

    private SpiralBugRunner() {
        // never call
    }
}
```

2.3 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. (Hint: Notice that a *Zbug* needs to be facing east before beginning its "Z" pattern.)

1. ZBug class

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

public class ZBug extends Bug {
    private int steps;
    private int sideLength;
    private int sides; // 添加一个 private int 变量记录走过的边数,"Z"有 3 条边

    /**
     * Constructs a z bug that with a given side length
     *
     * @param length
     *            the side length
     */
    public ZBug(int length) {
        setDirection(Location.EAST); // 初始方向为东
```

```
        sides = 0;
        steps = 0;
        sideLength = length;
    }

    /**
     * Moves to the next location of the "z".
     */
    @Override
    public void act() {
        // 已走完的情况下,没有条件分支可以执行,因此 bug 不会移动
        if (sides <= 2 && steps < sideLength) {
            // 未走完的情况下如果不能移动,同样没有条件分支执行,bug 也不会移动
            if (canMove()) {
                move();
                steps++;
            }
        } else if (steps == sideLength) {
            if (sides == 0) {
                setDirection(Location.SOUTHWEST);
            } else if (sides == 1) {
                setDirection(Location.EAST);
            }
            steps = 0;
            sides++; // 若已走完,steps 与 sideLength 相等,则 sides 自增到 3
        }
    }
}
```

2. ZBugRunner class

```
import java.awt.Color;

import info.gridworld.actor.ActorWorld;
import info.gridworld.grid.Location;

/**
```

```
 * This class runs a world that contains z bugs. <br />
 * This class is not tested on the AP CS A and AB exams.
 */
public final class ZBugRunner {
   public static void main(String[] args) {
      ActorWorld world = new ActorWorld();
      ZBug alice = new ZBug(5);
      alice.setColor(Color.ORANGE);
      ZBug bob = new ZBug(3);
      world.add(new Location(7, 8), alice);
      world.add(new Location(5, 5), bob);
      world.show();
   }

   private ZBugRunner() {
      // never call
   }
}
```

2.4 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.

1. DancingBug Class

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

public class DancingBug extends Bug {
   private int steps;
   private int sideLength;
   private int[] turnList; // 转向数组
```

```
private int currentTurn; // 当前的转向(数组下标)

/**
 * Constructs a dancing bug with a given side length and a given direction
 * array.
 *
 * @param length
 *            the side length
 * @param turn
 *            the turn list
 */
public DancingBug(int length, int[] turn) {
    if (turn == null) {
        turnList = new int[0];
    } else {
        turnList = turn.clone();
    }
    currentTurn = 0;
    steps = 0;
    sideLength = length;
}

/**
 * Moves to the next location.
 */
@Override
public void act() {
    if (steps < sideLength && canMove()) {
        move();
        steps++;
    } else {
        // 不能移动或者 steps = sideLength 时,会按照 turnList 转向继续走
        // turn 的有效次数只有 0-7,8 次与 0 次等价,以此类推,故简化循环
        for (int i = 0; i < turnList[currentTurn] % 8; i++) {
            turn();
```

```
            }
            steps = 0;
            // 数组循环
            currentTurn = (currentTurn + 1) % turnList.length;
        }
    }
}
```

2. DancingBugRunner class

```
import java.awt.Color;

import info.gridworld.actor.Actor;
import info.gridworld.actor.ActorWorld;
import info.gridworld.grid.BoundedGrid;
import info.gridworld.grid.Location;

/**
 * This class runs a world that contains dancing bugs. <br />
 * This class is not tested on the AP CS A and AB exams.
 */
public final class DancingBugRunner {
    public static void main(String[] args) {
        // dancinA's turn list
        int turnA[] = { 1, 2, 3, 4, 5, 6, 7, 6, 5, 4, 3, 2, 1 };
        // dancinB's turn list
        int turnB[] = { 5, 7, 6, 4, 3, 9, 2, 0, 100, 85, 43, 20 };
        ActorWorld world = new ActorWorld();
        DancingBug dancingA = new DancingBug(6, turnA);
        dancingA.setColor(Color.ORANGE);
        DancingBug dancingB = new DancingBug(3, turnB);
        world.setGrid(new BoundedGrid<Actor>(20, 20));
        world.add(new Location(7, 8), dancingA);
        world.add(new Location(10, 14), dancingB);
        world.show();
    }
```

```
  private DancingBugRunner() {
    // never call
  }
}
```

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

1. Use BoxBug's constructor to create a BoxBug:

   BoxBug bugName = new BoxBug(defineSideLength);

2. Add this BoxBug to the world by using world.add method:

   1) world.add(bugName); // at a random location

   or

   2) world.add(new Location(row, column), bugName); // at this location