Part 2:Bug Variation

Set 2

1.The sideLength defines the number of steps that a BoxBug can move on one direction.
2.The steps defines the number of steps that a BoxBug has moved on one direction.
3.Because it has to turn 90 degrees to achieve the next side of a box, but the turn() method only can turn 45 degrees.
4.Because the BoxBug extends the Bug class and Bug class has a public move method.
So the BoxBug inherits the move method from the Bug class.
5.Yes,because the sideLength is fixed.
6.Yes,When a BoxBug try to move when it face another Actor, such as a Rock or another Bug, the BoxBug will turn its direction and start a new Box.
7.The value of steps will be zero when a BoxBug is constructed and when the steps is equal to sideLength.And the value also will be zero when the BoxBug can't move and turn to a new direction.

Exercises

In the following exercises, write a new class that extends the Bug class. Override the act method to define the new behavior.

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?

```
//CicleBug.java
import info.gridworld.actor.Bug;

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

   //constructor
   public CircleBug(int length) {
      steps = 0;
      sidelength = length;
   }

   public void act() {
      if (steps < sidelength && canMove()) {
         move();
         steps++;
      }
      else
      {
         //just one turn is oks
         turn();
         //restart to move
```

```
            steps = 0;
        }
    }
}

//CicleBugRunner.java
import info.gridworld.actor.ActorWorld;
import info.gridworld.grid.Location;

import java.awt.Color;
/**
 * This class runs a world that contains Circle bugs. <br />
 * This class is not tested on the AP CS A and AB exams.
 */
public class CircleBugRunner {
    public static void main(String[] args) {
            ActorWorld world = new ActorWorld();
            CircleBug ly = new CircleBug(3);
            ly.setColor(Color.ORANGE);
            CircleBug hqk = new CircleBug(5);
        world.add(new Location(3,4), ly);
            world.add(new Location(4,5), hqk);
            world.show();
    }
}
```

The path will be an octagon instead of a circle.

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 two times to change 90 degree
                turn();
                turn();
                steps = 0;
                //we need to add the sidelength according to the picture
                sidelength++;
            }
        }
}

//SpiralBugRunner.java
import info.gridworld.actor.ActorWorld;
import info.gridworld.grid.Location;

import java.awt.Color;
/**
 * This class runs a world that contains Spiral bugs. <br />
 * This class is not tested on the AP CS A and AB exams.
 */
public class SpiralBugRunner {
        public static void main(String[] args) {
                ActorWorld world = new ActorWorld();
                SpiralBug sb = new SpiralBug(3);
                sb.setColor(Color.RED);
                world.add(new Location(5,5), sb);
                world.show();
        }
}
```

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;

public class ZBug extends Bug {
    private int steps;
    private int sidelength;
    //the number of the bug's turn
    private int turnTime;
    public ZBug(int length) {
        steps = 0;
        sidelength = length;
        turnTime = 0;
    }
```

```
	public void act() {
		//a bug need to turn
		if (steps == sidelength) {
			turnTime++;
			steps = 0;
		}
		//turn 135 clockwise when the bug move enough steps
		if ((turnTime == 1 || turnTime == 2) && steps == 0) {
			turn();
			turn();
			turn();
		}
		//turn 90 clockwise when the bug start to move or the second time to turn
		if (getDirection() == 0 && (turnTime == 0 || turnTime == 2)) {
			turn();
			turn();
		}
			//time that the bug should move,ohterwise stay still
		if (turnTime < 3 && canMove()) {
			move();
			steps++;
		}
	}
}

//ZbugRunner.java
import info.gridworld.actor.ActorWorld;
import info.gridworld.grid.Location;

import java.awt.Color;
/**
 * This class runs a world that contains Z bugs. <br />
 * This class is not tested on the AP CS A and AB exams.
 */
public class ZBugRunner {
	public static void main(String[] args) {
			ActorWorld world = new ActorWorld();
			ZBug zb1 = new ZBug(4);
			zb1.setColor(Color.BLUE);
			ZBug zb2 = new ZBug(9);
		zb2.setColor(Color.LIGHT_GRAY);
			world.add(new Location(7,7), zb1);
			world.add(new Location(0,0), zb2);
			world.show();
	}
}
```

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;
import java.util.Random;

public class DancingBug extends Bug {
	private int move[] = new int[100];
	private int steps;
	private int sidelength;
	//produce random number for bug's move
	public DancingBug(int length) {
		Random random = new Random();
		for (int i = 0; i < length; i++) {
		    move[i] = Math.abs(random.nextInt()*10 % 10);
		}
		steps = 0;
		sidelength = length;
	}

	public void act() {
		int temp = move[steps];
		//change direction
		while (temp > 0) {
			temp--;
			turn();
			}
		move();
		steps++;
		//restart a new loop
		if (steps == sidelength) {
		   steps = 0;
		}
	}
}

//DancingBugRunner.java
import info.gridworld.actor.ActorWorld;
import info.gridworld.grid.Location;

import java.awt.Color;
/**
 * This class runs a world that contains Dancing bugs. <br />
```

```
 * This class is not tested on the AP CS A and AB exams.
 */
public class DancingBugRunner {
	public static void main(String[] args) {
		ActorWorld world = new ActorWorld();
		DancingBug db = new DancingBug(10);
		db.setColor(Color.CYAN);
		world.add(new Location(5,5), db);
		world.show();
	}
}
```

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

```
//create a new BoxBug
BoxBug anotherBoxBug = new BoxBug(4);
//add it to the world at a specific location
world.add(new Location(5,5),anotherBoxBug);
```