Do You Know? Set 1:

1. No, the bug can be blocked with another actor (such as a rock) or if it hits a wall and then it will turn.
2. Forward.
3. It turns 45 degrees to clockwise.
4. A flower.
5. The bug will rotate until it is not facing an edge (unless it is not facing an edge already), and then it will move forward if nothing impedes its path.
6. The bug rotates once (45 degrees clockwise) to face a direction the rock is not in. Then the bug will move forward.
7. No.
8. The flower is created when a bug leaves that square (and has that bug’s color). It loses color until it turns black over time as the bug does not come back over the square. The flower regains its color when a bug passes over it again.
9. It does not move, although it has the property of stopping bugs.
10. No.

Exercises:

|  |  |
| --- | --- |
| Degress | Compass Direction |
| 0 | North |
| 45 | Northeast |
| 90 | East |
| 135 | Southeast |
| 180 | South |
| 225 | Southwest |
| 270 | West |
| 315 | Northwest |
| 360 | North |

1. You can move the bug to any location in the grid. You can move the bug to any location in the grid. If you move the bug to a place not in the grid then an alert message pops up saying that there is an IllegalArgumentException error.
2. void setColor(java.awt.Color)
3. The rock squishes the bug, because when you move the rock on top of bug and move the rock away, the bug is gone.

Do You Know? Set 2:

1. sideLength limits the space in which the bug can travel. The bug then travels in a sidelength+1 sided-square.
2. The steps variable is a count for the number of moves forward that the bug is allowed to take based on its sideLength. If the number of steps taken so far is less than the sideLength of the bug, then the bug can move forward.
3. The turn method is called twice, because the bug is rotated 90o instead of only 45o as the bug will travel in a square of the given sideLength. If the bug only rotated 45o, the bug wouldn’t move in a regular square pattern.
4. Move() can be called in BoxBug, because BoxBug extends the Bug class and the Bug class has a public move method. Inheritance of the method allows it to be called.
5. Yes, the size of the square pattern is written into the code and cannot be changed during the run of the program.
6. Yes, because if there is an object or edge in the way of the bug, then the bug will move to form a new box of flowers.
7. Initially, and when steps equals sideLength (when the Bug has reached the end of one side of its box path). Or when the bug cannot move and has to start a new box.

Exercises Set 2:

1, 2, 3, and 4 are included in the BlueJ package.

1. To create another BoxBug actor, you create a BoxBug (with the argument sent into the BoxBug constructor being the desired sided length of the box) in the BoxBugRunner class like this:  
   BoxBug newBug=new BoxBug(4);  
   Then add:  
   world.add(newBug);  
   to the BoxBugRunner class (above world.show();). You could also send in a Location argument in the world.add() before the newBug in order to give the newBug a specific location in the grid.

Do You Know? Set 3:

1. loc1.getRow();
2. false
3. (4,4)
4. Southeast
5. The getAdjacentLocation needs a direction parameter to be passed into the method. It then returns the location of the adjacent location in the direction given in the parameter list.

Do You Know? Set 4:

1. You can do the following, assuming that the Grid is called grid.  
   For the occupied locations: grid.getOccupiedLocations().size()  
   For the unoccupied locations: grid.getNumCols()\*grid.getNumRows()-grid.getOccupiedLocations().size()
2. boolean check=grid.isValid(new Location(10,10))
3. Grid is an interface that must be implemented by each class. The implementations are in the AbstractGrid, BoundedGrid, and UnboundedGrid classes. The BoundedGrid and UnboundedGrid classes extend the AbstractClass by defining what the size and final properties of the grid are. The AbstractGrid gives the general properties of the grid (but is an abstract class).
4. No, I think the ArrayList is a better design, because with an array, the elements in the grid couldn’t be in a list of just actors/bugs, because you would have to include the empty elements in the array. The Grid interface needs to be able to know how many different objects are in the grid at that time, which an array (instead of an ArrayList) would have a hard time.

Do You Know? Set 5:

1. Three properties of an actor are it has a color, direction, and location.
2. The actor’s starting color is blue and initially is in the North direction.
3. The Actor class is a class instead of an interface, because otherwise it couldn’t be overridden and we couldn’t program specific types of actors like bugs, rocks, and more specifically the BoxBug and SpiralBug. You can’t create variables or methods with an interface.
4. No, an actor cannot be added to the grid twice. There may only be one instance of that specific actor on the grid at once.

import info.gridworld.actor.ActorWorld;

import info.gridworld.grid.Location;

import java.awt.Color;

public class BoxBugRunner

{

public static void main()

{

ActorWorld world = new ActorWorld();

BoxBug bob = new BoxBug(4);

world.add(bob);

bob.putSelfInGrid(bob.getGrid(),bob.getLocation());

world.show();

}

}

This will have a runtime IllegalStateException.

No, the actor cannot be removed from the grid twice.   
  
This will have a compiling IllegalStateException.

import info.gridworld.actor.ActorWorld;

import info.gridworld.grid.Location;

import java.awt.Color;

public class BoxBugRunner

{

public static void main()

{

ActorWorld world = new ActorWorld();

BoxBug bob = new BoxBug(4);

world.add(bob);

bob.removeSelfFromGrid();

bob.removeSelfFromGrid();

world.show();

}

}

This will have a compiling IllegalStateException.  
Yes, an actor can be placed in the grid, be removed, and then put itself back in the grid.

import info.gridworld.actor.ActorWorld;

import info.gridworld.grid.Location;

import java.awt.Color;

public class BoxBugRunner

{

public static void main()

{

ActorWorld world = new ActorWorld();

ZBug bob = new ZBug(4);

world.add(new Location(5, 5), bob);

bob.removeSelfFromGrid();

bob.putSelfInGrid(bob.getGrid(),new Location(0,0));

world.show();

}

}

1. To turn an actor 90 degrees to the right, the turn() method can be called in the bug twice. Or you could use:  
   setDirection(getDirection()+90);

Do You Know? Set 6:

1. if (!gr.isValid(next))  
    return false;
2. Actor neighbor = gr.get(next);  
   return (neighbor==null) || (neighbor instanceof Flower);  
   This checks to see if the square the bug is moving to is filled with either nothing or just a Flower.
3. It uses .isValid() and get(). This is to make sure that the position the bug is moving to is valid and that the next location is either empty or only has a flower in it.
4. The .getAdjacentLocation() (which used the .getDirection() to call the .getAdjacentLocation()) is called to find the square that the bug is moving to.
5. It invokes the .getLocation(), .getDirection(), and .getGrid() methods from the Actor.
6. If the next location is off of the grid, the bug will remove itself from the grid using .removeSelfFromGrid().
7. Yes, the variable loc is necessary in the move method. This is because the loc variable stores the position of the bug before it moves and then is used later to put a flower there, but if the .getLocation() were called again then the bug’s previous position (where the flower needs to be placed) is unknown.
8. This in order to keep track of multiple bugs of different colors that are placed on the grid.
9. When the .removeSelfFromGrid() method is called, no flower is placed in the previous location. However, if the move method calls the .removeSelfFromGrid(), a flower is put in the old location.
10. Flower flower = new Flower(getColor());  
    flower.putSelfInGrid(gr, loc);
11. turn should be called 4 times.

Group Activity:

1. Specify:
   1. It will move one space forward turn once (45o) in that space.
   2. It will move only one space forward and then turn.
   3. It will turn once and move forward once, if this direction is not facing an edge. Otherwise, the bug will turn twice.
   4. It will remove that other actor from the grid.
   5. It will remove this other actor from the grid.
   6. Test to see if the Jumper only jumps over rocks and flowers or if it jumps over all actors.
2. Design:
   1. It should extend Actor.
   2. Yes, the Bug class is similar to the Jumper class.
   3. Yes. There are no necessary parameters, so calling a default Jumper() constructor would be okay. However, to be in-line with the Bug constructor there should be a constructor that makes a certain color of Jumper. The desired color of the Jumper would be the parameter.
   4. The method act() should be overridden.
   5. A jump(), canJump(), move(), and canMove(), and turn() should be added.
   6. To test the code, I will create a Jumper and JumperRunner class. Then I will test the program in specific situations, making sure the Jumper can jump over flowers and rocks and see what it will do when it comes to an edge.
3. Code: in BlueJ package.
4. Test: done in BlueJ