# Set 1

1. Does the bug always move to a new location? Explain.

No, sometimes it turns if it is in front of a wall.

1. In which direction does the bug move?

Forward in the direction that it is facing., unless it is in front of something, in which case it turns right by 45 degrees until it can move forward in the direction that it is facing.

1. What does the bug do if it does not move?

It turns right by 45 degrees

1. What does a bug leave behind when it moves?

A flower that is the same color as the bug, and bright.

1. What happens when the bug is at an edge of the grid? (Consider whether the bug is facing the edge as well as whether the bug is facing some other direction when answering this question>)

It turns right by 45 degrees each step until it can move forward again. If it is not facing towards the wall, then it moves forward like normal.

1. What happens when a bug has a rock in the location immediately in front of it?

It turns right by 45 degrees until it can move forward. In the case of the rock, it turns right 45 degrees and then starts moving diagonally.

1. Does a flower move?

No, it just darkens.

1. What behavior does a flower have?

It starts out bright, the same color as the bug, but then darkens until it is seemingly black. When the bug goes over it again, it either brings the color back to bright, or it overrides it with a new flower. It's hard to tell by just looking at the screen.

1. Does a rock move or have any other behavior?

The rock doesn't do anything by default because its “act” method has it stand still. The only thing that it can do is move locations to a specific point, but that only happens with user input.

1. Can more than one actor (bug, flower, rock) be in the same location in the grid at the same time?

No, because in the current behavior, no bug will move forward onto another bug or rock. This means that by running the simulation, there will never be two on the same grid. In addition to that, when I tried testing moving the rock onto the bug's location, then moving it off of it, the bug was gone, which means it probably only stores a pointer to one thing at a time for each grid space and by moving the rock onto the bug and then off of it again, the bug was erased.

# Set 1 Exercises (Page 8)

1. Test the setDirection method with the following inputs and complete the table, giving the compass direction each input represents

|  |  |
| --- | --- |
| Degrees | Compass Direction |
| 0 | North |
| 45 | North East |
| 90 | East |
| 135 | South East |
| 180 | South |
| 225 | South West |
| 270 | West |
| 315 | North West |
| 360 | North |

1. Move a bug to a different location using the moveTo method. In which directions can you move it? How far can you move it? What happens if you try to move the bug outside the grid?

You can move it to anywhere on the grid in any direction. If you try to move it outside the grid, it throws an exception

1. Change the color of a bug, a flower, and a rock. Which method did you use?

setColor()

1. Move a rock on top of a bug and then move the rock again. What happened to the bug?

The bug disappeared when the rock was moved off of it.

# Set 2 Do You Know?

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

It is the side length of the box the user wishes to have the bug trace

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

That's how far the bug is to completing one side of the box. Every time the steps gets to the side length, it resets to zero so it can start a new side.

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

Because the turn method only rotates by 45 degrees, and rotating a full 90 degrees is what is in a square.

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

Because it is inherited from the Bug class, and not overwritten.

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

Yes, because its sideLength variable is private and there is no method to change it.

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

Yes, because if it runs into an obstacle, then it turns prematurely. For example, if it hit a wall half way through .

1. When will the value of steps be zero?

Steps will be zero right after the creation of the object, and also directly after it has turned due to hitting an obstruction and hasn't moved yet.

# Set 2 Exercises (Page 13)

I . 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?

It traces out an octagon instead of a square.

1. 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.
2. 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 oflength 4. Hint: Notice that a ZBug needs to be facing east before beginning its "Z" pattern.

1. Write a class DaneingBug that "dances" by making different turns before each move. The Dane ingBug 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 Dane ingBugRunner class should create an array and pass it as a parameter to the DaneingBug constructor.

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

You would use the default constructor and pass an integer greater than zero to it and name it a charlie or eve. You would then call world.add(new Location(a, b), charlie) where a and b represent the position of your choice.

## Set 3

1. How would you access the row value for locl?

You would use the getRow() method.

1. What is the value of b after the following statement is executed?

false

**boolean b** locl.equals(loc2);

1. What is the value of loc3 after the following statement is executed?

**Location loc3** loc2.getAdjacentLocation( Location.SOUTH);

6, 6

1. What is the value of dir after the following statement is executed?

**int dir locl.getDirect ionToward(new Locat ion(6, 5));**

**135**

1. How does the getAdj acent Locat ion method know which adjacent location to return?

It knows which integer values represent which direction. For example, if it is passed Location.SOUTH, then it will know because that it is an integer and it knows what direction that integer is supposed to be.

## Set 4

1. How can you obtain a count of the objects in a grid? How can you obtain a count of the empty locations in a bounded grid?

Use getOccupiedLocations().size()

1. How can you check if location (10,10) is in a grid?

Use isValid(new Location(10, 10))

1. Grid contains method declarations, but no code is supplied in the methods. Why? Where can you find the implementations of these methods?

Grid does not contain any code because it is an interface, which cannot legally contain code. This is so that it doesn't interfere if a class extends another class and implements multiple interfaces which could have conflicting method implementations. The actual implementations of the method can be found in classes that implement it such as BoundedGrid and UnboundedGrid.

1. All methods that return multiple objects return them in an ArrayList. Do you think it would be a better design to return the objects in an array? Explain your answer.

I think it doesn't really matter since they would both function the same way in the end anyways. However, using an array might more sense in that you can't increase or decrease the size which means that it will always have the same length as the amount of objects originally returned. In addition to that, ArrayList might have space for extra data that will never be used, so it could be marginally better to return an array in terms of memory use. However, writing the method would be better with arraylist because then the programmer could go through all of the occupied locations, then add them to the list one at a time, instead of getting the total number of occupied locations, then making an arraylist, then setting each element of the array. So, from an implementation standpoint, ArrayList is better, but array is marginally better in terms of the user.

# Set 5

1. Name three properties of every actor.

color

direction

location

1. When an actor is constructed, what is its direction and color?

By default in the Actor constructor, color is Color.BLUE and direction is Direction.NORTH

1. Why do you think that the Actor class was created as a class instead of an interface?

Because it has code that it would be useful to inherit in multiple sub-classes, and Actor is its own classification of use in the program so it's important enough to make it its own class.

1. Can an actor put itself into a grid twice without first removing itself? Can an actor remove itself from a grid twice? Can an actor be placed into a grid, remove itself, and then put itself back? Try it out. What happens?

No, because it throws an IllegalStateException if grid is not equal to null.

1. How can an actor turn 90 degrees to the right?

The act() method contains the code necessary, it's just setDirection(getDirection() + Location.HALF\_CIRCLE)

# Set 6

1. Which statement(s) in the canMove method ensures that a bug does not try to move out of its grid?

It checks to see that the grid is on is not null with “if (gr == null)” and it checks to see if the direction it would move is not out of the grid with “if (!gr.isValid(next))”

1. Which statement(s) in the canMove method determines that a bug will not walk into a rock?

The return statement “return (neighbor == null) || (neighbor instanceof Flower)” means that it can only move if there is nothing occupying the next space that isn’t a flower.

1. Which methods of the Grid interface are invoked by the canMove method and why?

isValid(), and get(). These methods are invoked because the method needs to know if the location that it will move to is valid or not based on the specific grid. It also needs to find the actor at a location, specifically the one it would move into next

1. Which method of the Locat ion class is invoked by the canMove method and why?

getAdjacentLocation() is used to find the location that is in the direction the Bug is currently facing.

1. Which methods inherited from the Actor class are invoked in the canMove method?

getGrid() is used so the method has the current grid to work with. The getLocation() method is used to get the location the Bug is currently on the grid. getDirection() is used to get the direction the Bug is currently facing.

1. What happens in the move method when the location immediately in front of the bug is out of the grid?

Depending on the grid, the isValid method will return true or false depending on the implementation of the isValid method. If it’s on a bounded grid, then the canMove() method will return false if it would move outside of the grid because gr.isValid() would return false. If it’s an unbounded grid, then gr.isValid() would automatically return true, which means this wouldn’t be an issue.

1. ls the variable loc needed in the move method, or could it be avoided by calling get Location ( ) multiple times?

In the move() method, the variable loc is needed because in the second if statement, the location is either removed or changed which means that the getLocation() method could return a different result than flower.putSelfInGrid(gr, loc)

1. Why do you think the flowers that are dropped by a bug have the same color as the bug?

This is so that the user can tell which bug dropped the flower, and can see the path that the bug went through easier, and have it match up with the bug. The code reason is that when the flower is created, the result of getColor() is passed as an argument to the constructor.

1. When a bug removes itself from the grid, will it place a flower into its previous location?

It will if this happens as a result of the move() method, because the flower.putSelfInGrid() is called even if the bug is removed from the grid, meaning it will leave a flower. However, just the removeSelfFromGrid() method called alone outside of that method will not leave a flower.

1. Which statement(s) in the move method places the flower into the grid at the bug's previous location?

Flower flower = new Flower(getColor()) //this creates the flower with the color of the bug

Flower.putSelfInGrid(gr, loc) //puts it into the grid that the bug is in at the old location

1. If a bug needs to turn 180 degrees, how many times should it call the turn method?

It would call it four times because it uses Location.HALF\_RIGHT which is equal to 45 degrees, and 4 \* 45 = 180.

# Page 26: Jumper Bug Code

1 a it will go just one forward instead

B it will only move only forward instead

C it will turn until it can move

D only move forward one

E it will jump over it

2 a actor

B not sure, would have to look at all of the classes, but I don’t think so

C yes, just the initial color would be fine

D act(), possibly toString()

E canMoveTwo(), moveTwo(), canMoveOne(), moveOne(), turn()

F put a jumper into a grid, then see how it reacts when it reaches the end of the grid, whether or not it jumps over rocks, flowers, etc.

import info.gridworld.grid.\*;

import info.gridworld.actor.\*;

import info.gridworld.world.\*;

import java.awt.Color;

//A Jumper is an actor that jumps around by 2 spaces, and only one if it can't. If it can't move at all, then it turns.k

public class Jumper extends Actor {

//constructor that sets color to given arg

public Jumper(Color jumperColor) {

setColor(jumperColor);

}

//no arg constructor that defaults to red

public Jumper() {

this(Color.RED);

}

// moves 2 if it can, if it can't, then it moves 1, if it can't, then it turns

public void act() {

if (canMoveTwo()) {

moveTwo();

} else if (canMoveOne()) {

moveOne();

} else {

turn();

}

}

public boolean canMoveTwo() {

Grid<Actor> gr = getGrid();

if (gr == null)

return false;

Location loc = getLocation();

Location nextTwo = loc.getAdjacentLocation(getDirection()).getAdjacentLocation(getDirection());

if (!gr.isValid(nextTwo))

return false;

Actor atNextTwo = gr.get(nextTwo);

return (atNextTwo == null) || (atNextTwo instanceof Flower);

}

public boolean canMoveOne() {

Grid<Actor> gr = getGrid();

if (gr == null)

return false;

Location loc = getLocation();

Location next = loc.getAdjacentLocation(getDirection());

if (!gr.isValid(next))

return false;

Actor neighbor = gr.get(next);

return (neighbor == null) || (neighbor instanceof Flower);

}

public void moveTwo() {

Grid<Actor> gr = getGrid();

if (gr == null)

return;

Location loc = getLocation();

Location nextTwo = loc.getAdjacentLocation(getDirection()).getAdjacentLocation(getDirection());

if (gr.isValid(nextTwo))

moveTo(nextTwo);

else

removeSelfFromGrid();

}

public void moveOne() {

Grid<Actor> gr = getGrid();

if (gr == null)

return;

Location loc = getLocation();

Location nextTwo = loc.getAdjacentLocation(getDirection()).getAdjacentLocation(getDirection());

if (gr.isValid(nextTwo))

moveTo(nextTwo);

else

removeSelfFromGrid();

}

public void turn() {

setDirection(getDirection() + Location.HALF\_RIGHT);

}

}

import info.gridworld.world.\*;

import info.gridworld.grid.\*;

import info.gridworld.actor.\*;

import java.awt.Color;

public class JumperRunner {

public static void main(String[] args) {

ActorWorld world = new ActorWorld();

Jumper alice = new Jumper();

Jumper bob = new Jumper(Color.BLUE);

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

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

bob.setDirection(Location.SOUTHEAST);

world.show();

}

}

# Set 7: Do You Know

1. What methods are implemented in Critter?

Act() is the only method from Actor that is implemented

1. What are the five basic actions common to all critters when they act?

setColor(not sure if this is supposed to be one)

setDirection

putSelfInGrid

removeSelfFromGrid

moveTo

act

1. Should subclasses of Critter override the getActors method? Explain.

Probably not because the same behaviour is probably desirable in the subclass and there's probably not a reason to change it.

1. Describe three ways that a crlitter could process actors.

It could eat them

It could ignore them

it could move them

1. What three methods must be invoked to make a critter move? Explain each of these methods.

getMoveLocations(): gets the adjacent locations that can be moved to

selectMoveLocation(): returns a location from the list of possible move locations passed to it. The default behavior is to set it randomly.

MakeMove(): it removes itself from the grid if the location is null, and moves to it if not.

1. Why is there no Critter constructor?

There is no extra setup that needs to be done for the Critter, so it automatically calls the superclass's constructor.

# Set 8: Do You Know?

1. Why does act cause a ChameleonCr itter to act differently from a Critter even though ChameleonCritter does not override act?

The act method calls the processActors() method and the processActors method is different.

1. Why does the makeMove method of ChameleonCritter call super.makeMove?

Because it doesn't just move it to the spot, it turns it in that direction first.

1. How would you make the ChameleonCritter drop flowers in its old location when it moves?

You would override the act() method by first dropping the flower, then calling super.act();

1. Why doesn't ChameleonCritter override the getActors method?

Because getting the adjacent actors is the same and doesn't need to be changed for Chameleon critter.

1. Which class contains the getLocation method?

Actor

1. How can a Critter access its own grid?

It can call its getGrid() method.

# Set 9: Do You Know?

1. Why doesn't CrabCritter override the processActors method?

Because they still want that class to process actors in the same way or change the behavior.

1. Describe the process a CrabCritter uses to find and eat other actors. Does it always eat all neighboring actors? Explain.

It moves randomly left or right and eats things ahead of it in the forward and diagonal directions.

1. Why is the getLocationsinDirections method used in CrabCritter?

Because it is convenient to have a method that can check an arbitrary list of directions. It's used in getMoveLocations and getActors to do just that, and it uses a different list of directiions each time.

1. If a CrabCritter has location (3, 4) and faces south, what are the possible locations for actors that are returned by a call to the getActors method?

(4,3) (4,4), (4,5)

1. What are the similarities and differences between the movements of a CrabCritter and a Critter?

Crab Critter moves only left and right, which keeps it in a line.

1. How does a CrabCritter determine when it turns instead of moving?

If the location to move to is itself, then it randomly set a direction left or right, but it calls the supercall make move method if it is no.

1. Why don't the CrabCritter objects eat each other

because it's written into the code not to eat crab critters.

# Page 35 Exercises

1. Modify the processActors method in ChameleonCritter so that if the list of actors to process is empty, the color of the ChameleonCritter will darken (like a flower).
2. Create a class called ChameleonKid that extends ChameleonCritter as modified in exercise I. A ChameleonKid changes its color to the color of one of the actors immediately in front or behind. Ifthere is no actor in either of these locations, then the ChameleonKid darkens like the modified ChameleonCritter.
3. Create a class called RockHound that extends Critter. A RockHound gets the actors to be processed in the same way as a Critter. It removes any rocks in that list from the grid. A RockHound moves like a Critter.
4. Create a class BlusterCritter that extends Critter. A BlusterCritter looks at all of the neighbors within two steps of its current location. (For a BlusterCritter not near an edge, this includes 24 locations). It counts the number of critters in those locations. If there are fewer than c critters, the BlusterCritter's color gets brighter (color values increase). If there are c or more critters, the BlusterCritter's color darkens (color values decrease). Here, c is a value that indicates the courage of the critter. It should be set in the constructor.
5. Create a class QuickCrab that extends CrabCritter. A QuickCrab processes actors the same way a CrabCritter does. A QuickCrab moves to one of the two locations, randomly selected, that are two spaces to its right or left, if that location and the intervening location are both empty. Otherwise, a QuickCrab moves like a CrabCritter.
6. Create a class KingCrab that extends CrabCritter. A KingCrab gets the actors to be processed in the same way a CrabCritter does. A KingCrab causes each actor that it processes to move one location further away from the KingCrab . If the actor cannot move away, the KingCrab removes it from the grid. When the KingCrab has completed processing the actors, it moves like a CrabCritter .