1.We can use the \*\*public int getRow()\*\* method in the Location class to access the row value of loc1. \*\*e.g int row = loc1.getRow();\*\*

//@file:info/gridworld/grid/Location.java

//@line:110~113

public int getRow()

{

return row;

}

2.\*\*b = false\*\* since that the row and column value of loc1 don't equal to the value of loc2.

//@file:info/gridworld/grid/Location.java

//@line:210~211\*

Location otherLoc = (Location) other;

return getRow() == otherLoc.getRow() && getCol() == otherLoc.getCol();

3.\*\*loc3 has row 4, column 4.\*\* Because the \*\*getAdjacentLocation(int direction)\*\* method returns the adjacent location of loc2 in a direction, since loc2 has row 3, column 4 and the direction is south, the value of loc3 will be known.

set the change of row and column to 0:

//@file:info/gridworld/grid/Location.java

//@line:138~139

int dc = 0;

int dr = 0;

set the row change according to the direction:

//@file:info/gridworld/grid/Location.java

//@line:147~148\*

else if (adjustedDirection == SOUTH)

dr = 1;

get the location accrording to it's row and column:

//@file:info/gridworld/grid/Location.java

//@line:168\*

return new Location(getRow() + dr, getCol() + dc);

4.\*\*dir has value 135(degrees)\*\* since the \*\*getDirectionToward(Location target)\*\* method returns a closet direction from this location toward target.

Since loc1 has row 4 and column 3 and the target has row 6 and column 5,from the method we can get the answer.

get the mathematical angle:

//@file:info/gridworld/grid/Location.java

//@line:180~183

int dx = target.getCol() - getCol();

int dy = target.getRow() - getRow();

int angle = (int) Math.toDegrees(Math.atan2(-dy, dx));

get the compass angle

//@file:info/gridworld/grid/Location.java

//@line:187~194

int compassAngle = RIGHT - angle;

compassAngle += HALF\_RIGHT / 2;

if (compassAngle < 0)

compassAngle += FULL\_CIRCLE;

return (compassAngle / HALF\_RIGHT) \* HALF\_RIGHT;

5.first it get the compassDirection from the parameter.

//@file:info/gridworld/grid/Location.java

//@line:133~137

int adjustedDirection = (direction + HALF\_RIGHT / 2) % FULL\_CIRCLE;

if (adjustedDirection < 0) adjustedDirection += FULL\_CIRCLE;

adjustedDirection = (adjustedDirection / HALF\_RIGHT) \* HALF\_RIGHT;

then initialize the row and column change = 0

//@file:info/gridworld/grid/Location.java

//@line:138~139

int dc = 0;

int dr = 0;

then according to the direction get the row and column change

//@file:info/gridworld/grid/Location.java

//@line:140~167\*

e.g. if (adjustedDirection == EAST) dc = 1;

then get the location according to the primitive row, column and the change of row, column

//@file:info/gridworld/grid/Location.java

//@line:168

return new Location(getRow() + dr, getCol() + dc);

6.We can use the \*\*getOccupiedLocations()\*\* to get all the occupiedLocations, since it returns an array, so we can use \*\*grid.getOccupiedLocations().size()\*\* to get the count of the objects(assume grid is a Grid object

We can use \*\*getNumRows() \* getNumCols()\*\* to get the count of all the locations in a grid.So we can use \*\*grid.getNumRows() \* grid.getNumCols() - grid.getOccupiedLocations().size()\*\* to get the count of the empty locations in a bounded grid

//@file:info/gridworld/grid/BoundedGrid.java

//@line:50 and 57

return occupantArray.length;

return occupantArray[0].length;

7.We can use the \*\*isValid()\*\* method to check whether a location is in a grid. e.g.\*\*grid.isValid(new Location(10, 10))\*\*

//@file:info/gridworld/grid/Grid.java

//@line:50

boolean isValid(Location loc);

8.Because \*\*Grid is an interface\*\*, an interface in Java can only have methods but don't implement them.

We can find the implementations in the classes which implements the Grid interface, such as \*\*AbstractGrid, BounedGrid and UnBoundedGrid\*\*.But the AbstractGrid only implements part of the methods while the BounedGrid and UnBoundedGrid implements the rest of the methods.

//@file:info/gridworld/grid/AbstractGrid.java

//@file:info/gridworld/grid/BoundedGrid.java

//@file:info/gridworld/grid/UnBoundedGrid.java

9.No.Although array is more efficient, but \*\*it's size can't change\*\*.We don't know the count of the objects the method returns, so we can't set the size of the array.

ArrayList is less efficient than array, but it's size can change, so it would be better to use in general cases.

10.\*\*grid, location, direction, color\*\*

//@file:info/gridworld/actor/Actor.java

//@line:31~34

private Grid<Actor> grid;

private Location location;

private int direction;

private Color color;

11.The initial direction is \*\*NORTH\*\* and the initial color is \*\*BLUE\*\*;

//@file:info/gridworld/actor/Actor.java

//@line:41~42

color = Color.BLUE;

direction = Location.NORTH;

12.Because an \*\*interface can't implement methods and can only have public static final members\*\*, but the Actor has private variables and implements methods.

[//@file:info/gridworld/actor/Actor.java](mailto:/@file:info/gridworld/actor/Actor.java)

13.(1)No. It will \*\*cause an IllegalStateException\*\*

//@file:info/gridworld/actor/Actor.java

//@line:117~119

if (grid != null)

throw new IllegalStateException(

"This actor is already contained in a grid.");

(2)No.It will also \*\*cause an IllegalStateException\*\*

//@file:info/gridworld/actor/Actor.java

//@line:135~137

if (grid == null)

throw new IllegalStateException(

"This actor is not contained in a grid.");

(3)Yes.It won't cause any error.

once it removes itself, the grid variable of the Actor becomes null:

//@file:info/gridworld/actor/Actor.java

//@line:143~144

grid.remove(location);

grid = null;

and since the grid variable is null, the push method can execute correctly:

//@file:info/gridworld/actor/Actor.java

//@line:124~125

gr.put(loc, this);

grid = gr;

14.We can use the \*\*setDirection()\*\* method in Actor.

e.g.\*\*actor.setDirection(getDirection() + 90)\*\*

//@file:info/gridworld/actor/Actor.java

//@line:82~84

direction = newDirection % Location.FULL\_CIRCLE;

if (direction < 0)

direction += Location.FULL\_CIRCLE;

15.We can use the \*\*setDirection()\*\* method in Actor.

e.g.\*\*actor.setDirection(getDirection() + 90)\*\*

//@file:info/gridworld/actor/Actor.java

//@line:82~84

direction = newDirection % Location.FULL\_CIRCLE;

if (direction < 0)

direction += Location.FULL\_CIRCLE;

16.The \*\*isValid(next)\*\* statement.

//@file:info/gridworld/actor/Bug.java

//@line:98~99

if (!gr.isValid(next))

return false;

17.The last statement which determines the next location must be null or a flower:

//@file:info/gridworld/actor/Bug.java

//@line:98~99

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

18.The \*\*isValid() and get()\*\* method.

Because we need to ensure that the next location won't be outside the grid and we need to get the bug in the next loaction to see whether the actor an move to the next location.

//@file:info/gridworld/actor/Bug.java

//@line:98 and 100

if (!gr.isValid(next))

Actor neighbor = gr.get(next);

19.The \*\*getAdjacentLocation()\*\* method.

Because we need to get the next location of the bug according to it's direction.

//@file:info/gridworld/actor/Bug.java

//@line:97

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

20.The \*\*getGrid(), getLocation() and getDirection()\*\* method.

//@file:info/gridworld/actor/Bug.java

//@line:93 and 96 and 97

Grid<Actor> gr = getGrid();

Location loc = getLocation();

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

21.The bug will remove itself from the grid.

//@file:info/gridworld/actor/Bug.java

//@line:80~81

else

removeSelfFromGrid();

22.Yes.The getLocation() method returns the loaction of the actor(the bug), and it's value is the same as the loc, so we could replace the loc by calling getLocation() multiple times.

//@file:info/gridworld/actor/Actor.java

//@line:71

return direction;

1. Because it can help us to find the path of a bug, different color will be useful to distinguish different paths.

24.If the bug removes itself by use the removeSelfFromGrid(), e.g.bug.removeSelfFromGrid(), then it won't place a flower into the previous location since the removeSelfFromGrid() can't do that.

But if the bug removes itself by move(), e.g.when the move() method causes the bug out of the grid, then according to the codes in move() method, it will place a flower into the previous location.

//@file:info/gridworld/actor/Bug.java

//@line:82~83

Flower flower = new Flower(getColor());

flower.putSelfInGrid(gr, loc);

25.The \*\*putSelfInGrid()\*\* method which call by a flower.

//@file:info/gridworld/actor/Bug.java

//@line:82~83

Flower flower = new Flower(getColor());

flower.putSelfInGrid(gr, loc);

26.\*\*4\*\* Times.Because the turn method only turn 45 degrees once.

`//@file:info/gridworld/actor/Bug.java

//@line:64

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