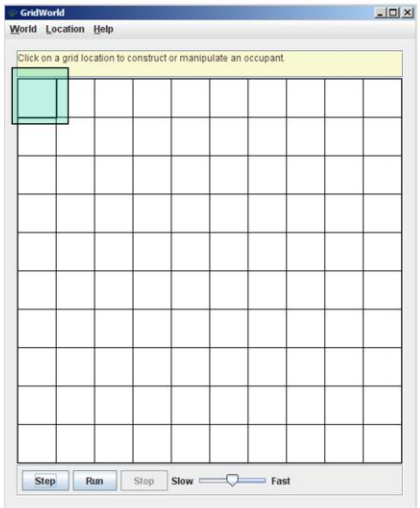

GridWorld

What is GridWorld?



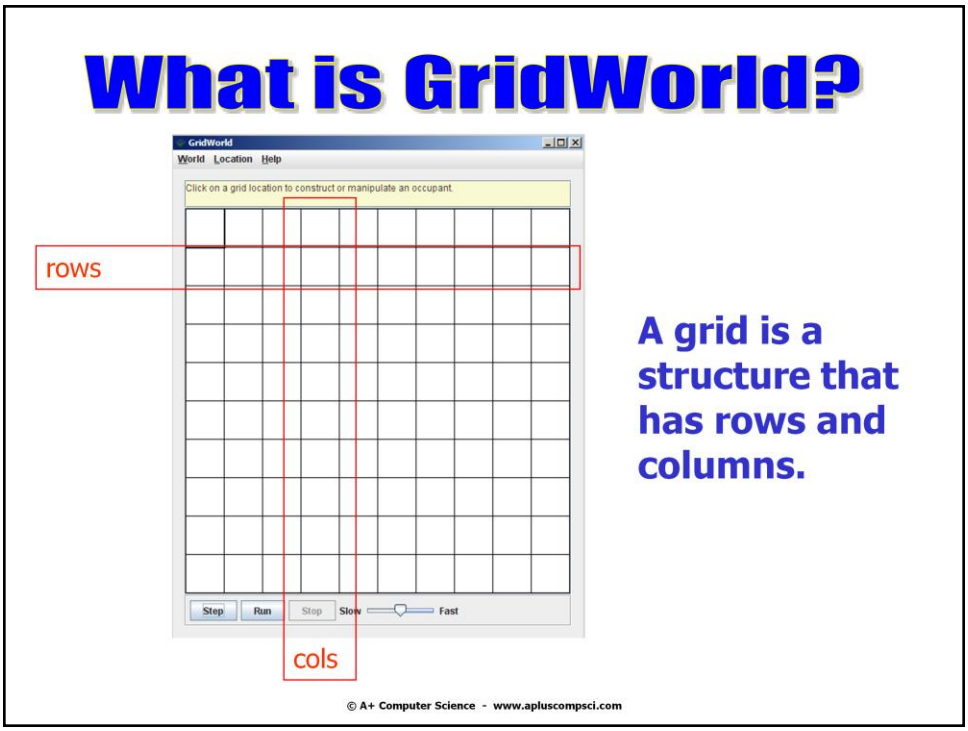
Row = 0

Column = 0

A grid is a structure that has rows and columns.

A spreadsheet is a grid.

A checker board is a grid.

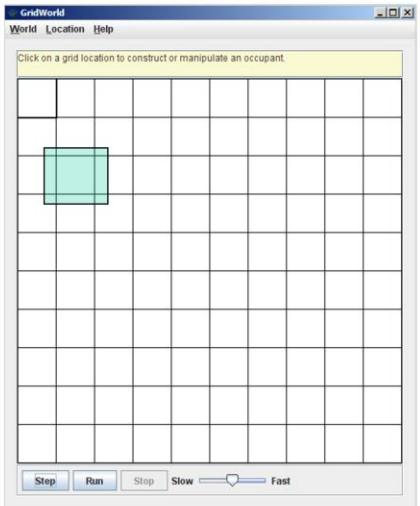


A grid is a structure that has rows and columns.

A spreadsheet is a grid.

A checker board is a grid.

What is GridWorld?



Row = 2

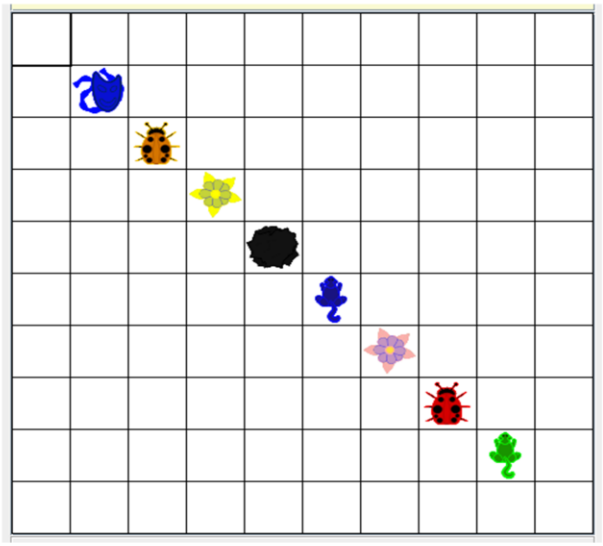
Column = 1

A grid is a structure that has rows and columns.

A spreadsheet is a grid.

A checker board is a grid.

What is GridWorld?



Grid

Grid

Grid is an interface that details the behaviors expected of a Grid.

Grid was designed as an interface because many different structures could be used to store the grid values.

An interface works perfectly due to the large number of unknowns.

Grid is a row / column structure that stores Objects.

The location of each Object is determined by the Location provided when putting the Object in the grid.

Grid
abstract methods

Name	Use
get(loc)	returns the ref at location loc
getEmptyAdjacentLocations(loc)	gets the valid empty locs in 8 dirs
getNeighbors(loc)	returns the objs around this
getNumCols()	gets the # of cols for this grid
getNumRows()	gets the # of rows for this grid
getOccupiedAdjacentLocations(loc)	gets the valid locs in 8 dirs that contain objs
getOccupiedLocations()	gets locs that contain live objs
getValidAdjacentLocations(loc)	gets the valid locs in 8 dirs
isValid(loc)	checks to see if loc is valid
put(loc, obj)	put the obj in grid at location loc
remove(loc)	take the obj at location loc out of the grid

```
import info.gridworld.grid.Grid;
```

Grid

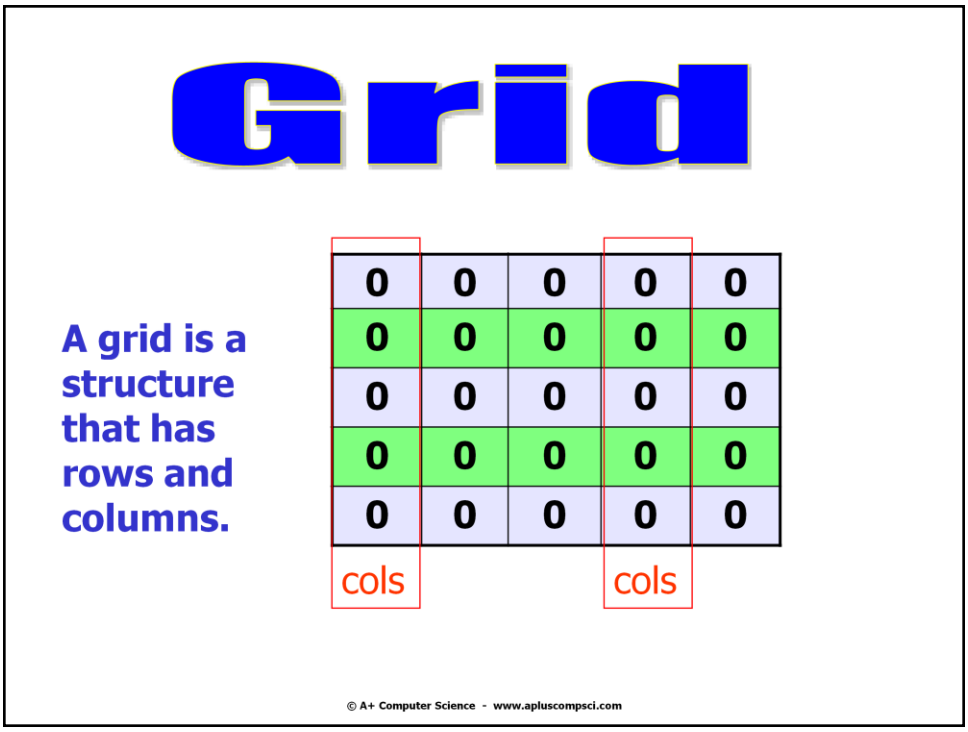
rows	0	0	0	0	0
	0	0	0	0	0
	0	0	0	0	0
	0	0	0	0	0
rows	0	0	0	0	0

A grid is a structure that has rows and columns.

A grid is a structure that has rows and columns.

A spreadsheet is a grid.

A checker board is a grid.



A grid is a structure that has rows and columns.

A spreadsheet is a grid.

A checker board is a grid.

Bug

Bug

Bug differs from actor in that a bug actually moves from cell to cell.

A bug moves to the cell immediately in front if possible. If a move is not possible, the bug turns in 45 degree increments until it finds a spot to which it can move.

Bug is a suped up actor.

Bug extends actor.

Bug has two constructors, one of which takes a Color parameter.

Bug will move when its act method is called.

Bug

extends Actor

frequently used methods

Name	Use
getColor()	gets the bug's color
getDirection()	gets the bug's direction
getLocation()	gets the bug's location
setColor(col)	sets the bug's color to col
setDirection(dir)	sets the bug's direction to dir

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


The methods listed below were inherited from actor.

The act method has been overridden as the behavior of a bug is quite different from an actor.

The other methods listed above that were inherited have not been changed.

Bug

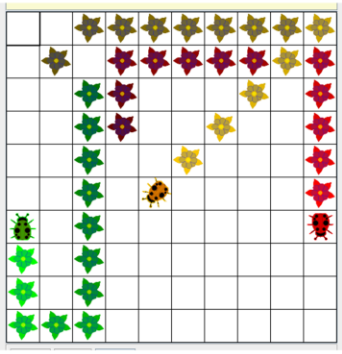
```
ActorWorld world = new ActorWorld();

Bug dude = new Bug();
world.add(new Location(3,3), dude);

Bug sally = new Bug(Color.GREEN);
sally.setDirection(Location.SOUTHEAST);
world.add(new Location(2,2), sally);

Bug ali = new Bug(Color.ORANGE);
ali.setDirection(Location.NORTHEAST);
world.add(new Location(1,1), ali);

world.show();
```



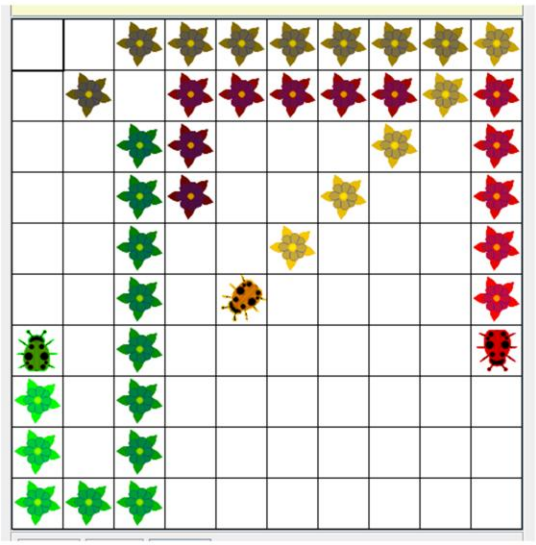
In this example, a default bug is created.

A default bug is red and facing NORTH.

The bug moves NORTH until it reaches the top of the grid. The bug them turns in 45 degree increments until it finds an empty location to which to move.

Two other bugs are created as well. One of the bugs is green and the other is orange.

Bug



open bugone.java

Bug

extends Actor

frequently used constructors

Name	Use
Bug()	make a new red bug going north
Bug(color)	make a new bug set to color

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


Bug is a suped up actor.

Bug extends actor.

Bug has a default parameter-less constructor and an additional constructor that receives a Color parameter.

Bug has overridden the act method inherited from actor.

Bug has some new methods that are unique to bug : canMove, move, and turn.

Bug

extends Actor

frequently used methods – Bug specific

Name	Use
act()	move if possible or turn
canMove()	check to see if a move is possible
move()	move forward and leave a flower
turn()	turn 45 degrees without moving

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

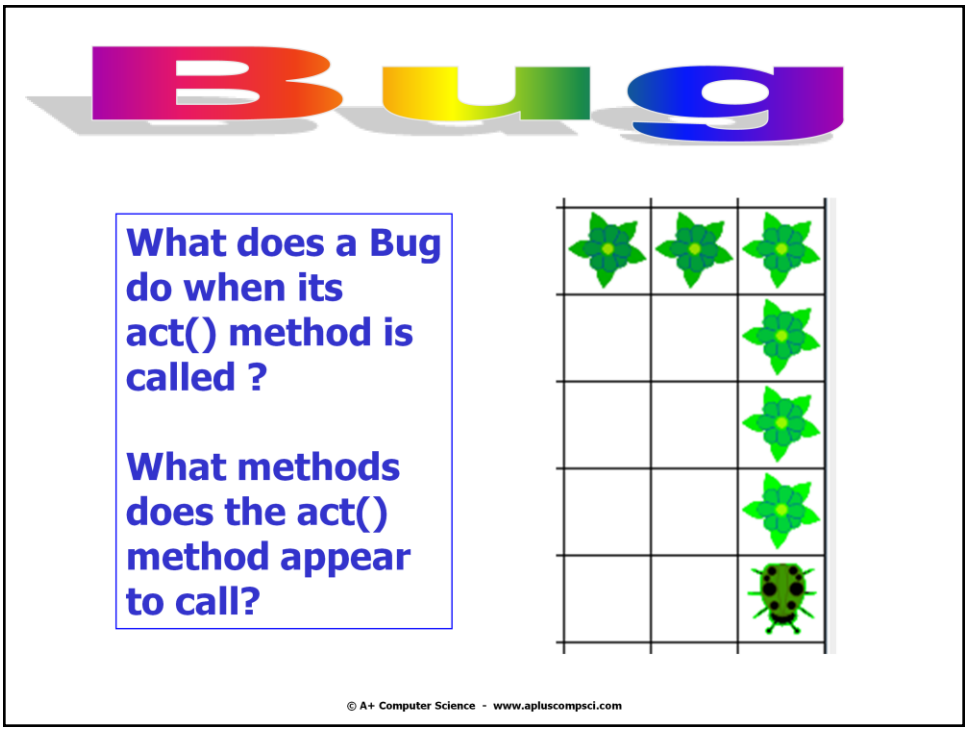

Bug is a suped up actor.

Bug extends actor.

Bug has a default parameter-less constructor and an additional constructor that receives a Color parameter.

Bug has overridden the act method inherited from actor.

Bug has some new methods that are unique to bug : canMove, move, and turn.



Each time the act method is called the bug will move to the location/cell immediately in front of it.

The bug will turn if the cell/location in front of it is occupied or invalid.

A flower of the same color as the bug is left in the cell/location vacated by the bug when it moves.

The bug act method calls the canMove method to see if the bug can move. If the bug can move, the bug move method is called. If the bug cannot move, the turn method is called.

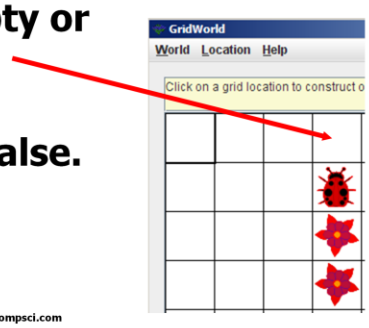
Open up Bug.java and look at the code. Much can be gained from looking at the bug code.

canMove

The bug act method looks to see if a move is possible by calling canMove.

canMove looks at the location in front of this bug to see if it is empty or if it contains a flower.

canMove returns true or false.



Each time the bug act method is called the bug will move to the location/cell immediately in front of it.

The bug will turn if the cell/location in front of it is occupied or invalid.

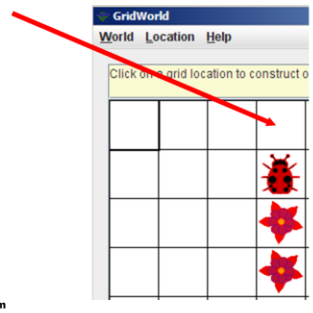
A flower of the same color as the bug is left in the cell/location vacated by the bug when it moves.

The default color of a bug is red and the default direction of a bug is NORTH.

move

The bug act method calls move if canMove returns true.

move calls moveTo to move the bug to the location in front of this bug. move leaves a flower in the old location.



Each time the bug act method is called the bug will move to the location/cell immediately in front of it.

The bug will turn if the cell/location in front of it is occupied or invalid.

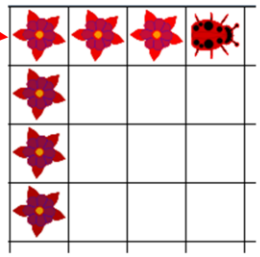
A flower of the same color as the bug is left in the cell/location vacated by the bug when it moves.

The default color of a bug is red and the default direction of a bug is NORTH.

turn

The bug act method calls turn if canMove returns false.

turn changes the direction of the bug by 45 degrees to the right.



Each time the bug act method is called the bug will move to the location/cell immediately in front of it.

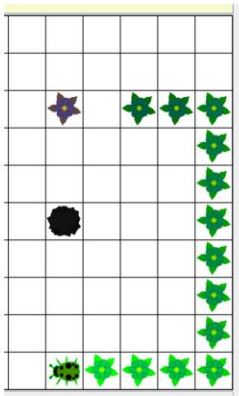
The bug will turn if the cell/location in front of it is occupied or invalid.

A flower of the same color as the bug is left in the cell/location vacated by the bug when it moves.

The default color of a bug is red and the default direction of a bug is NORTH.

Bug

```
ActorWorld world = new ActorWorld();
Bug dude = new Bug(Color.GREEN);
dude.setDirection(Location.EAST);
Location loc = new Location(5,5);
world.add(loc , new Rock());
loc = new Location(2,5);
world.add(loc, new Flower());
loc = new Location(2,7);
world.add(loc, dude);
world.show();
```



Each time the bug act method is called the bug will move to the location/cell immediately in front of it.

The bug will turn if the cell/location in front of it is occupied or invalid.

A flower of the same color as the bug is left in the cell/location vacated by the bug when it moves.

The default color of a bug is red and the default direction of a bug is NORTH.

open bugtwo.java

Extending Bug

Extending Bug

How will the new bug differ from the original bug?

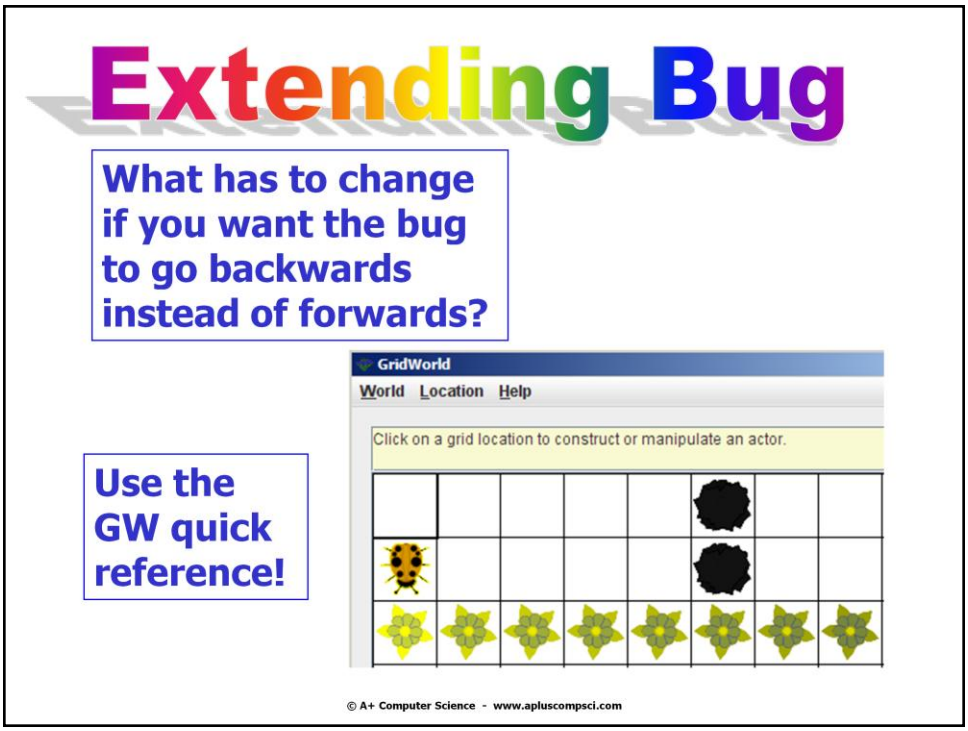
Can the new behavior be created using existing methods?

Which of the methods will be overridden?

Will new methods need to be added?

In order to make a new type of bug, you must understand all of the original actor and bug methods.

When creating a new bug, it is important to determine what the new bug will do and how it will differ from the original bug.



What must change from the original bug if the new bug is to move backwards rather than forwards?

How does a bug move forward?

Which method makes a bug move?

How does act use canMove, move, and turn?

Extending Bug

```
public class BackwardBug extends Bug
{
  //constructor

  public void act()
  {



  }

  //other methods

}
```

Is this the only way to write this class?

What methods could be changed?

What must change from the original bug if the new bug is to move backwards rather than forwards?

How does a bug move forward?

Which method makes a bug move?

How does act use canMove, move, and turn?

open
backwardbug.java
backwardbugrunner.java

AP Exam Info

You will be given Bug and BoxBug in the quick reference when taking the AP exam.

You CAN override any of the BUG methods when making a new Bug.

Move and CanMove provide great examples of how to use getAdjacentLocation(), isValid, and get().

Always look at the original Bug and BoxBug code when making a new Bug.

AP Exam Info

You will be given Bug and BoxBug in the quick reference when taking the AP exam.

BoxBug also provides examples of instance variables, method overriding, and constructors.

You CAN override any of the BUG methods when making a new Bug.

Always look at the original Bug and BoxBug code when making a new Bug.

open
boxbug.java
boxbugrunner.java

Start work on Bug Exercises and Labs