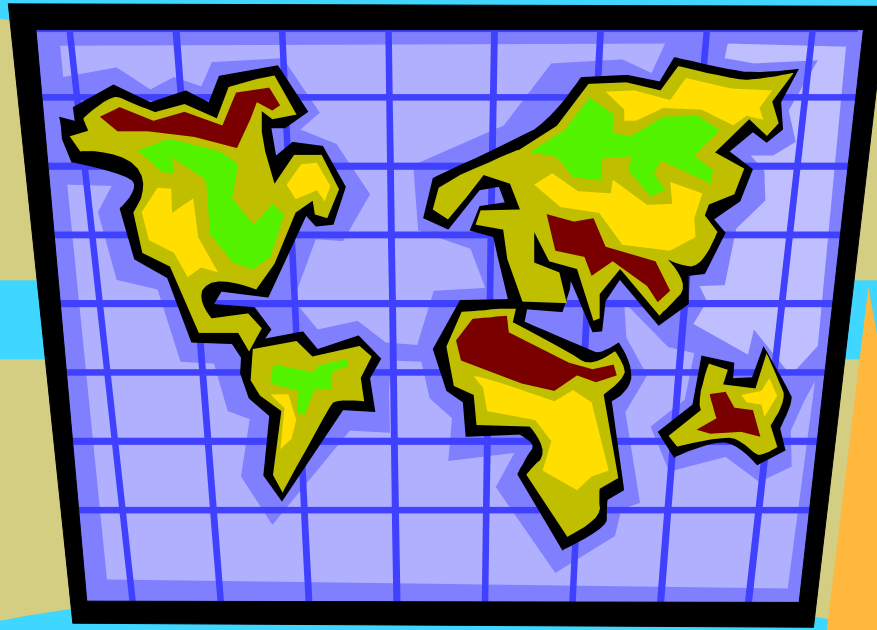
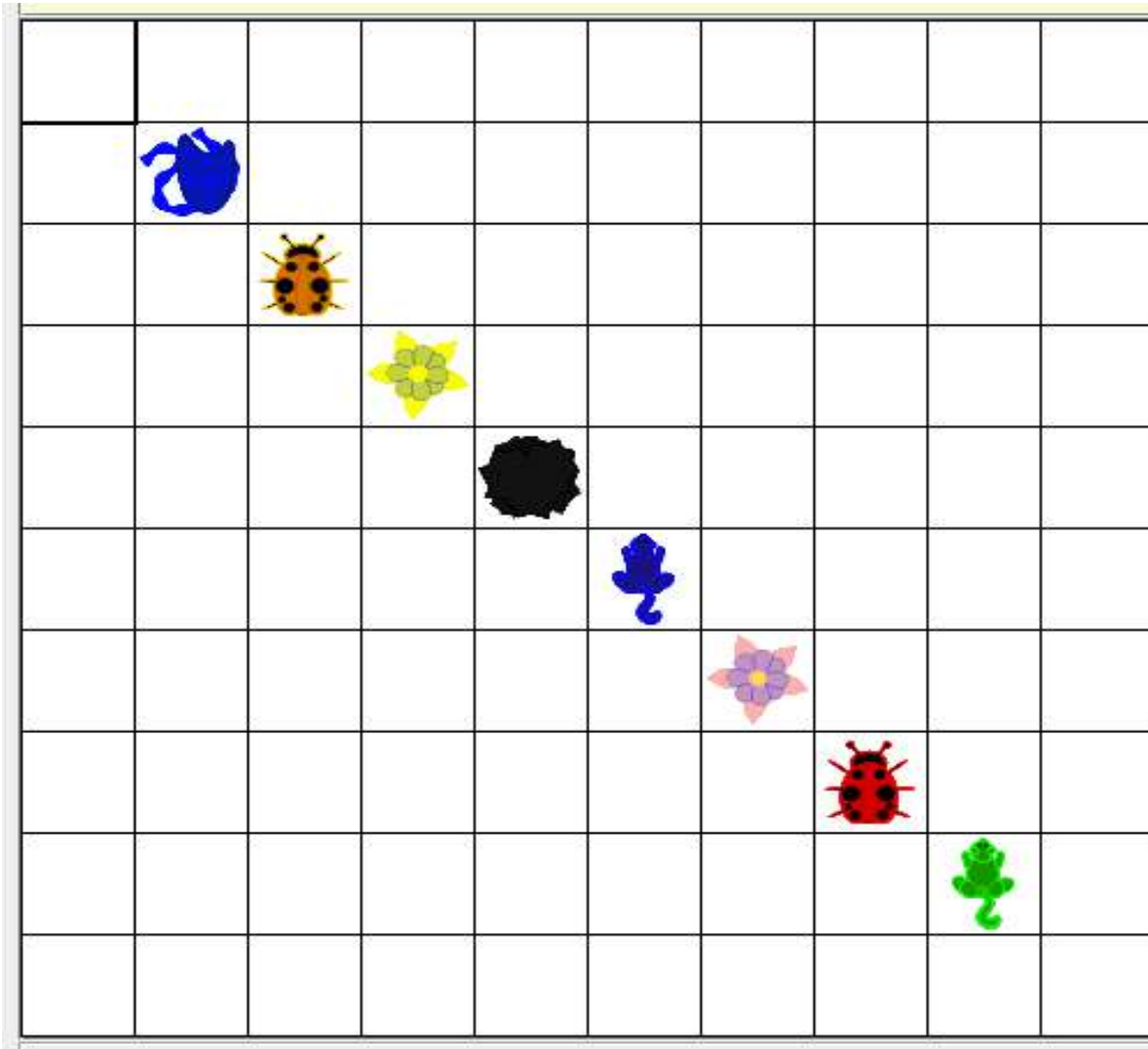


GridWorld



What is GridWorld?



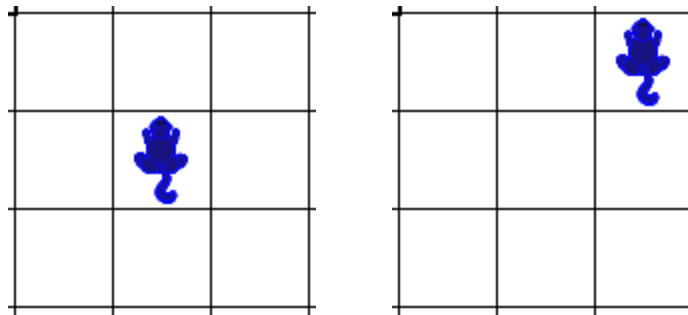
Critter

Default Critter Behavior

Critter differs from Actor in that a critter eats specific types of other actors adjacent to it.

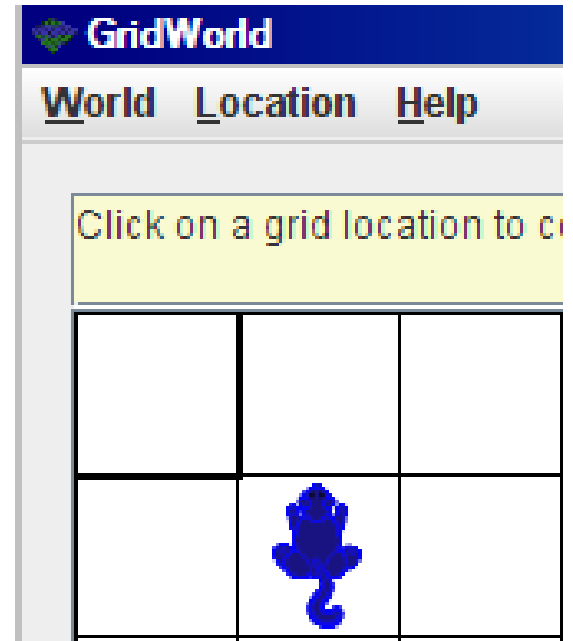
Critter randomly picks one of its valid adjacent empty locations and moves to that location.

The result is that Critter moves around the grid and eats specific Actors.



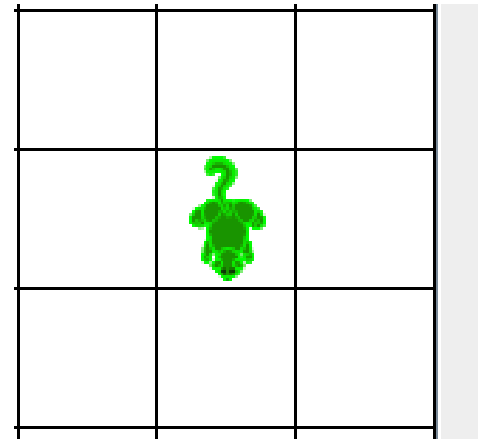
Critter

```
ActorWorld world = new ActorWorld();  
Critter thing = new Critter();  
world.add(new Location(1,1), thing);  
world.show();
```



Critter

```
ActorWorld world = new ActorWorld();  
Critter thing = new Critter();  
thing.setColor(Color.GREEN);  
thing.setDirection(180);  
Location loc = new Location(2,2);  
world.add(loc, thing);  
world.show();
```



Critter

extends Actor

frequently used methods from Actor

Method name	How is it used?
getColor()	gets the critter's color
getDirection()	gets the critter's direction
getLocation()	gets the critter's location
setColor(col)	sets the critter's color to col
setDirection(dir)	sets the critter's direction to dir

```
import info.gridworld.actor.Critter;
```

Critter Behavior

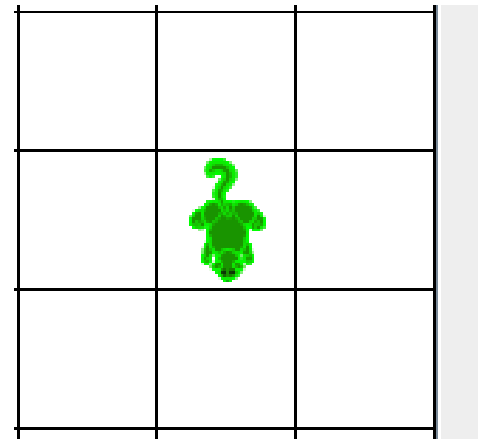
Different types of critters may select move locations in different ways, may have different ways of processing the actors, and may vary the actions they take when they make the move.

Examples:

- **One type of critter might get all its neighboring actors and perform some action on each one of them (change color, change direction, make them move, etc.)**
- **Another critter may only select actors in front of it to process**

act()

```
public void act()  
{  
if (getGrid() == null)  
return;  
ArrayList<Actor> actors = getActors();  
processActors(actors);  
ArrayList<Location> moveLocs = getMoveLocations();  
Location loc = selectMoveLocation(moveLocs);  
makeMove(loc);  
}
```



Critter

extends Actor

frequently used methods – Critter specific

Method name	How is it used?
act()	calls the methods listed below
getActors()	returns an ArrayList of all actors around this location
processActors(actors)	does something to selected actors
getMoveLocations()	returns an ArrayList of possible move locations
selectMoveLocation(locs)	picks location to move to
makeMove(loc)	moves this critter to location loc or removes itself from grid if loc is null

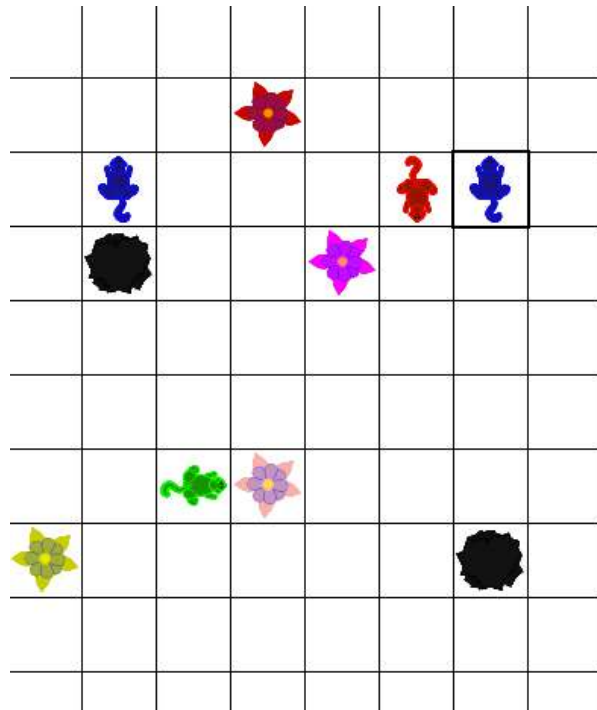
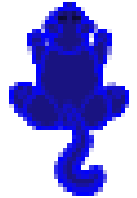


if no grid present – stop

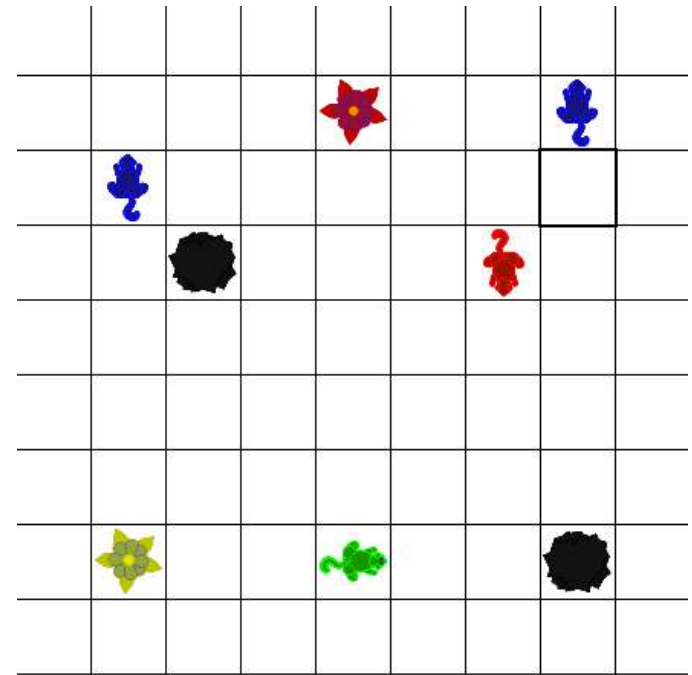
- call **getActors** to get an array list of actors to process
- call **processActors** to process the array list of actors received from getActors
- call **getMoveLocations** to get an array list of locations where the critter might move
- call **selectMoveLocation** to select new location from the array list of possible locations
- call **makeMove** to move to selected location

Critter

- Compile and run CritterRunner.java.



After
one
step



Extending Criticter

getActors

The getActors method returns an ArrayList containing all of the actors around this critter using the 4 cardinal (N,S,E,W) and 4 intercardinal directions (NE, NW, SE, SW).

In order to change which actors are returned by getActors, override this method and provide a different approach of selecting actors.

getActors must not modify any actors; that is the job of processActors

getActors

```
public ArrayList<Actor> getActors()
{
    return getGrid().getNeighbors(getLocation());
}
```

Gets all neighbors adjacent to it

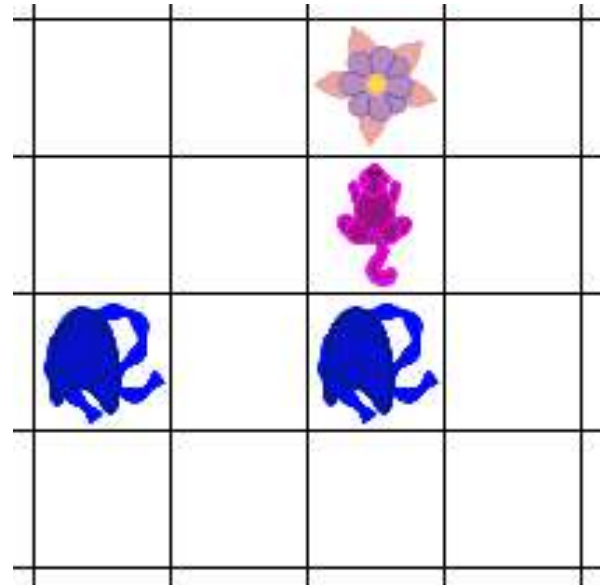
Extending Critter



Use the GW
quick reference!

What has to change if you want a
critter to only get actors in front
and behind?

**Make sure you satisfy the post
conditions of each method you
override!!**



Extending Critter

```
public class GetInFrontBehindCritter extends Critter
{
    //constructor

    public ArrayList<Actor> getActors()
    {
        ArrayList<Actor> actors = new ArrayList<Actor>();
        Grid<Actor> grid = getGrid();
        Location inFront = getLocation().getAdjacentLocation(getDirection());
        Location behind = getLocation().getAdjacentLocation(getDirection() +
            180);
        if(grid.isValid(inFront) && grid.get(inFront) != null)
            actors.add(grid.get(inFront));
        if(grid.isValid(behind) && grid.get(behind) != null)
            actors.add(grid.get(behind));
        return actors;
    }
}
```

What code is needed?

processActors

The processActors method will do something to some or all of the actors around this critter.

The processActors receives a list of all actors around this actor based on this actor's getActors method.

The critter act method calls getActors and passes the returned ArrayList to processActors.

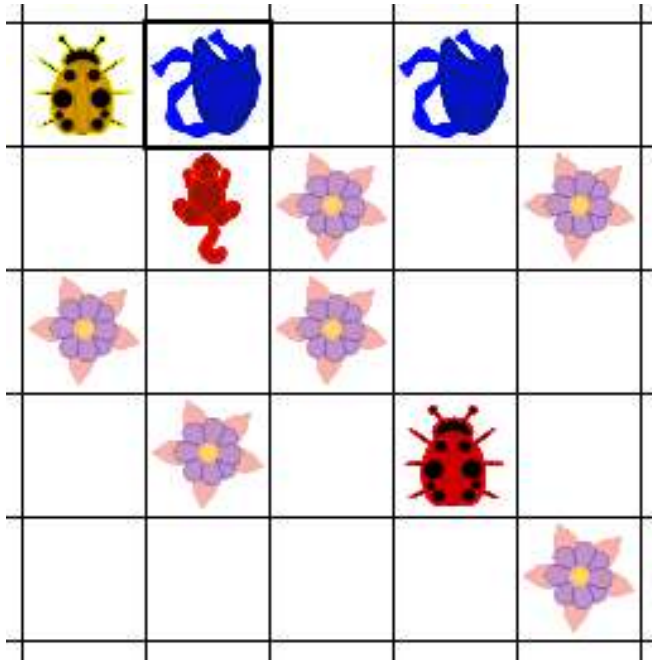
processActors must only change the actors received in the ArrayList parameter.

processActors

```
public void processActors(ArrayList<Actor> actors)
{
    for (Actor a : actors)
    {
        if(!(a instanceof Rock) && !(a instanceof Critter))
            a.removeSelfFromGrid();
    }
}
```

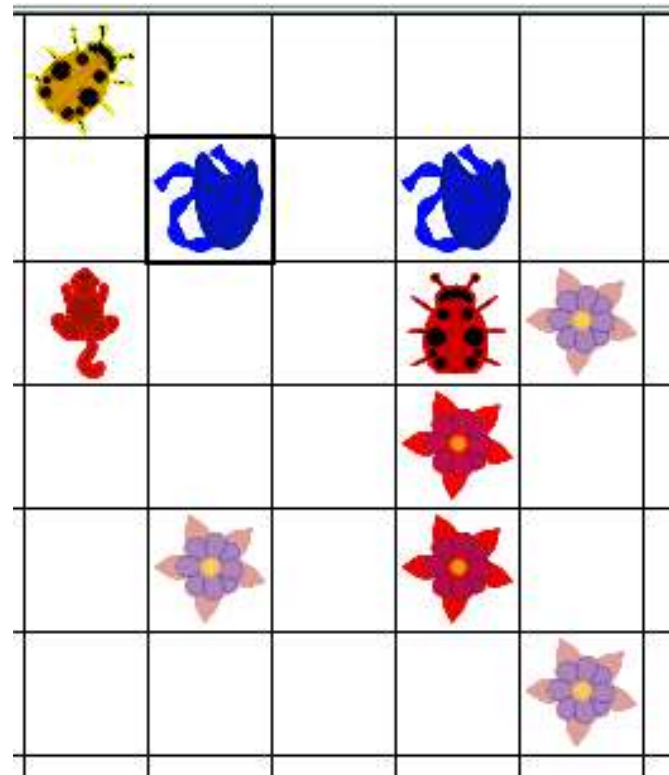
Removes all neighbors adjacent to it

Extending Critter



Use the GW
quick
reference!

What has to change if
you want a critter to
only eat flowers?



Extending Critter

```
public class FlowerEatingCritter extends Critter
{
    //constructor

    public void processActors(ArrayList<Actor> actors)
    {
        for(Actor a : actors)
        {
            if(a instanceof Flower)
                a.removeSelfFromGrid();
        }
    }
}
```

What code is needed?

getMoveLocations

The getMoveLocations method returns a list of all empty adjacent locations to which this critter could move.

In order to change which locations are returned by getMoveLocations, override the method and provide a different method of selecting move locations.

getMoveLocations must not modify any actors.

getMoveLocations

```
public ArrayList<Location> getMoveLocations()
{
    return getGrid().getEmptyAdjacentLocations(getLocation());
}
```

Gets all possible locations where the critter could move

selectMoveLocation

The selectMoveLocation method selects a possible move location from the list of locations returned by getMoveLocations.

The selectMoveLocation receives a list of all actors around this actor based on this actor's getMoveLocations method.

The critter act method calls getMoveLocations and passes the returned ArrayList to selectMoveLocation.

selectMoveLocation must not modify any actors.

selectMoveLocation

```
public Location selectMoveLocation(ArrayList<Location> locs)
{
    int n = locs.size();
    if (n==0)
        return getLocation();
    int r = (int) (Math.random() * n);
    return locs.get(r);
}
```

Selects the one location where the critter will move

makeMove

The makeMove method receives a location parameter.

If the parameter is null, the critter is removed from the grid.

If the parameter is not null, the critter moves to the new location. If an actor was in the location the critter is moving to, the actor is removed.

makeMove must only modify the actors at this critter's new and old locations.

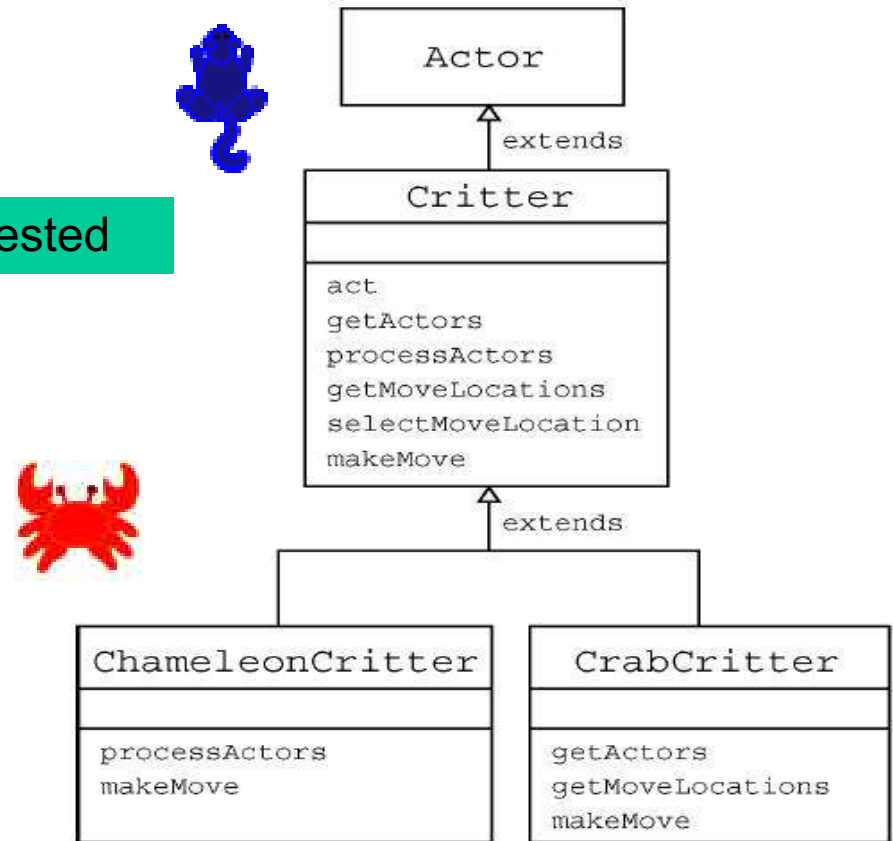
makeMove

```
public void makeMove(Location loc)
{
    if (loc == null)
        removeSelfFromGrid();
    else
        moveTo(loc);
}
```

Moves the critter to the selected location

Extending the Critter Class

- **ChameleonCritic** p. C-6
 - Overrides `processActors` to pick a random actor and change its current color to the actor's color
 - Overrides `makeMove` to also turn toward the new location
- **CrabCritic**
 - Overrides `getActors` to get actors straight ahead, diagonal left, and diagonal right (from front)
 - Overrides `getMoveLocations` to only move to the left or right
 - Overrides `makeMove` so that if it doesn't move it randomly turns left or right



Not Tested

Overview of GridWorld Creatures

- Actors
 - Bugs
 - extending Bug: override Act
 - Rocks
 - Flowers
 - Critters
 - **DO NOT OVERRIDE ACT!!!**
 - extending `Critter` normally means that you are overriding one or more of the 5 methods

Summary of what is tested

API

- Actor
- Flower
- Rock
- Grid
- Location



API stands for “**Application Programming Interface**”. In Java, an API is the definition of how a programmer can access the functionality contained within a code library.

Summary of what is tested

Implementation

- Bug
- BoxBug
- Critter
- ChameleonCritter



An **Implementation** is a class that implements or extends the given API in order to perform a specific function.

Exercises

1. Create a class named `EnhancedChameleon` that extends `ChameleonCritter`. If there are no actors around this critter to process, its color should darken (like a flower). Hint: get the color of the critter and use some of the `Color` methods to darken each of its RGB values by subtracting `.05` from each of them.
2. Create a class named `ChameleonKid` that extends `ChameleonCritter` as modified in exercise 1. A `ChameleonKid` changes its color to the color of one of the actors immediately in front or behind. If there is no actor in either of these locations, then the `ChameleonKid` darkens like the modified `EnhancedChameleon`.

Exercises

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 called `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.

Exercises

5. Create a class `QuickCrab` that extends `CrabCritic`. A `QuickCrab` processes actors the same way a `CrabCritic` 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 `CrabCritic`.
6. Create a class `KingCrab` that extends `CrabCritic`. A `KingCrab` gets the actors to be processed in the same way a `CrabCritic` 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 `CrabCritic`.

Group Activity

- Specify
 - Create a new creature that extends Critter
- Design
 - What variables are needed? What algorithms are needed?
- Code
 - Implement the code
- Test
 - Write test cases for the creature specified