Lab: Blocks

Background: `Location location =new Location(0, 0);`

In this lab, you will use your implementation of `MyBoundedGrid<E>` to represent a grid of objects of type `E`. The `MyBoundedGrid<E>` class makes use of the `Location` class, which simply remembers a row and column index. A summary of the `Location` class is given below, but you will probably only use `getRow`, `getCol`, and `equals` in this lab.

`Location` Class (`implements Comparable`)

```
public Location(int r, int c)
public int getRow()
public int getCol()
public Location getAdjacentLocation(int direction)
public int getDirectionToward(Location target)
public boolean equals(Object other)
public int hashCode()
public int compareTo(Object other)
public String toString()

NORTH, EAST, SOUTH, WEST, NORTHEAST, SOUTHEAST,
NORTHWEST, SOUTHWEST, LEFT, RIGHT, HALF_LEFT,
HALF_RIGHT, FULL_CIRCLE, HALF_CIRCLE, AHEAD
```

Go ahead and download all the posted lab files.

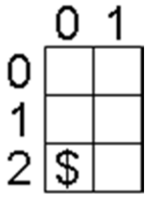
Background: `MyBoundedGrid<E>`

Do not create any packages. Put all of your code in the default package.

The `MyBoundedGrid<E>` class will allow us to place objects) into a grid, as shown in the following example.

```
MyBoundedGrid grid = new MyBoundedGrid<String>(3, 2);
grid.put(new Location(2, 0), "$");
```

This code creates a grid with 3 rows and 2 columns, containing a `"$"` in the lower left corner, as the following diagram shows.



Here is a summary of `MyBoundedGrid<E>`'s constructor and methods.

`MyBoundedGrid<E>` class

```
public MyBoundedGrid<E>(int rows, int cols)
public int getNumRows()
public int getNumCols()
public boolean isValid(Location loc)
public E put(Location loc, E obj)
public E remove(Location loc)
public E get(Location loc)
public ArrayList<Location> getOccupiedLocations()
```

Background: The `Block` Class

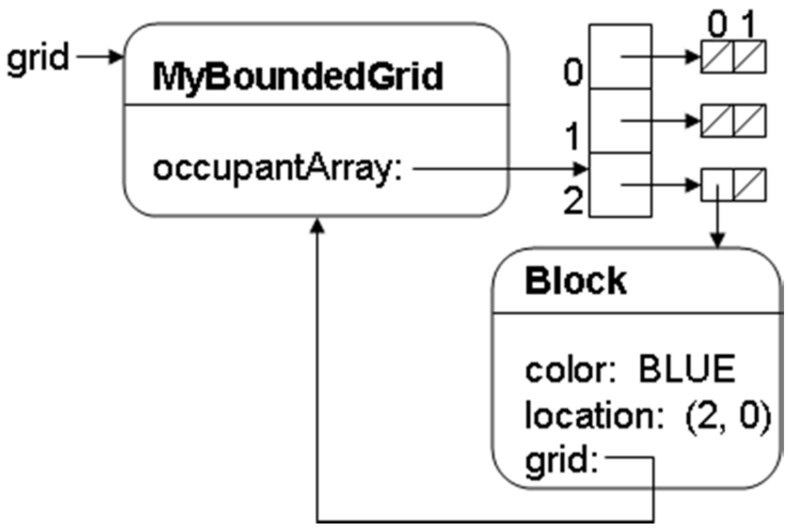
Do not create any packages. Put all of your code in the default package.

We are going to place a colored `Block`—an object which keeps track of its color, location, and the `MyBoundedGrid<Block>` that contains it — in our grid. When the `Block` is *not* in a grid, its `grid` and `location` instance variables are both `null`. When the Block *is* in a grid, its `location` instance variable must match the `Block`'s location within the `MyBoundedGrid` associated with its `grid` instance variable. This means that both the `Block` and its `grid` are keeping track of the `Block`'s location. It's up to the `Block` class to keep these consistent.

The following example shows how to place a `Block` in a `MyBoundedGrid<Block>`.

```
MyBoundedGrid<Block> grid;
grid = new MyBoundedGrid<Block>(3, 2);
Block block = new Block();
block.putSelfInGrid(grid, new Location(2, 0));
```

The following instance diagram shows the result of executing this code.



Here is a summary of the `Block` class.

`Block` Class

```
public Block()
public Color getColor()
public void setColor(Color newColor)
public MyBoundedGrid<Block> getGrid()
public Location getLocation()
public void putSelfInGrid(MyBoundedGrid<Block> gr, Location loc)
public void removeSelfFromGrid()
public void moveTo(Location newLocation)
public String toString()
```

Exercise 1: Coder's Block

Do not create any packages. Put all of your code in the default package.

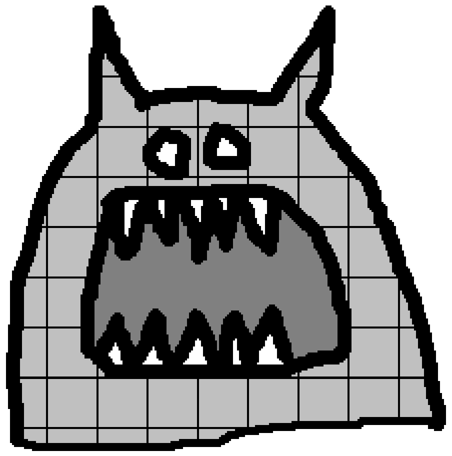
Do not create any packages during this lab. Use only the default package. If you have put `MyBoundedGrid` into a package, take it out!

Go ahead and complete the `Block` class you downloaded. You may test your work if you like, or wait to use the automated tester in the next exercise.

Exercise 2: Monster Test

Do not create any packages. Put all of your code in the default package.

Now that you've completed the `MyBoundedGrid<E>` and `Block` classes, it's time to see if you have what it takes to beat the dreaded `GridMonster`!



Go ahead and run the `GridMonster` class you downloaded. Your code must pass each of the `GridMonster`'s tests in order to complete the lab. But beware! The `GridMonster` has a nasty habit of insulting any code that displeases him (or her). Good luck!

When you have completed this lab, show your teacher your fully documented code and show that you have gotten past the dreaded `GridMonster.` Then create a zip archive of all your source files (and only your source files) and then upload the archive to Athena2. If your code is in a package, then your teacher will not be able to run it from the uploaded archive, and you will be assessed a penalty. If you upload something other than a zip archive (rar, for instance) your code will be considered late until it is properly uploaded.