Tetrix Project (Part 0).docx

Welcome to Tetris! In this multi-part assignment you will be building the Tetris game (with lots of help provided). It should take you approximately five days to complete the assignment, with one portion of the assignment due each day.

This is another "Pair Programming" Assignment

While you are working on this project, you will use the "Pair Programming" approach. In pair programming, two programmers share one computer. One student is the "driver," who controls the keyboard and mouse. The other is the "navigator," who observes, asks questions, suggests solutions, and thinks about slightly longer-term strategies. You should switch roles about every 20 minutes. For this assignment each partner will receive the same grade.

Be sure that both of you have duplicate saved copies of your work each day. You need to have something to work with tomorrow in the event your partner is absent.

---

acknowledgements to Dave Feinberg for this assignment
(and his hilarious sense of humor!)

To get started, download, extract, and save the Tetris project files to your computer. Be sure to compile all of these files before continuing.

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

In this first part of the Tetris project, you should become familiar with the `BoundedGrid` class that represents a grid of objects. Later we will put block objects into this grid. The grid is "bounded" in the sense that it is not infinite in size. The `BoundedGrid` class makes use of the `Location` class, which simply remembers a row and column index. A summary of constructor and other methods used in the `Location` class is given below, but you will probably only use the methods `getRow`, `getCol`, and `equals` in this project.

`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 compareTo(Object other)
public String toString()
```

Background: ***BoundedGrid***

The `BoundedGrid` class will allow us to place objects into a grid, as shown in the following example.

```
BoundedGrid grid;
grid = new BoundedGrid(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. The other 5 locations simply contain null values, indicating that those locations are empty.

Here is a summary of `BoundedGrid`'s constructor and other methods.

BoundedGrid class

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

`BoundedGrid` stores its objects in a 2-dimensional array. Your `BoundedGrid` class therefore has only a single instance variable.

```
private Object[][] occupantArray;
```

Thankfully, you will never need to interact with `occupantArray` directly. Instead, you'll call the methods `getNumRows`, `getNumCols`, `get`, and `put`, which have already been implemented for you.

Background: The ***Block*** *Class*

The first kind of thing we'll place in our grid is a colored `Block`, which is an object that keeps track of its color, location, and the `BoundedGrid` that contains it. 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 `BoundedGrid` 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 `BoundedGrid`.

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

Here is a summary of the `Block` class.

`Block` Class

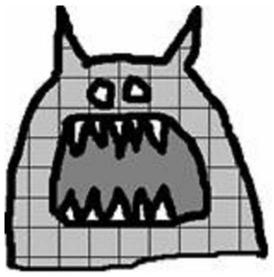
```
public Block()
public Color getColor()
public void setColor(Color newColor)
public BoundedGrid getGrid()
public Location getLocation()
public void putSelfInGrid(BoundedGrid gr, Location loc)
public void removeSelfFromGrid()
public void moveTo(Location newLocation)
public String toString()
```

Exercise 0.0: Coder's Block

Go ahead and complete the `Block` class you downloaded. Most of the methods are very easy to complete, but if you get stuck on some of the more difficult ones, follow the directions in the comments.

Exercise 0.1: Monster Test

Now that you've completed the `Block` class, 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 project. But beware! The `GridMonster` has a nasty habit of insulting any code that displeases him (or her). Good luck!