Tetrix Project (Part 1).docx

Don't forget that this is a Paired Programming assignment, so you should continue to work with your partner on this. If you have not yet switched roles between 'driver' and 'navigator', be sure to do so now. Remember to continue to switch roles early and often for this entire project.

Background: `Block`

We are going to be using the `Block` class that you created in Part 0 of this project. Each instance of the `Block` class will represent a single 1x1 block in the game of Tetris. For your reference, below is a summary of its constructor and methods:

`Block` Class

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

Background: `BlockDisplay`

The `BlockDisplay` class represents a window that shows the contents of a particular `BoundedGrid`. Here is a summary of its constructor and methods:

`BlockDisplay` Class

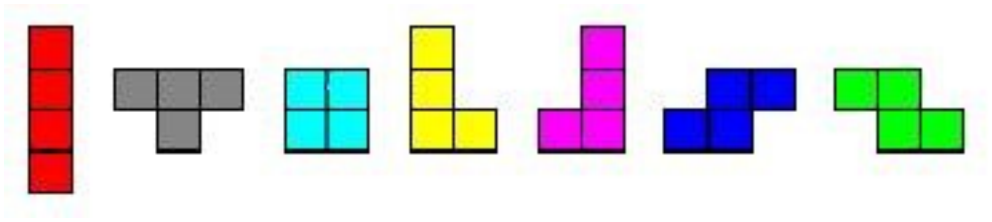
```
public BlockDisplay(BoundedGrid grid)
public void showBlocks()
public void setTitle(String title)
public void setArrowListener(ArrowListener listener)
```

Exercise 1.0: Block Pop-Up Window

Open the class called `Tetris`, which will be the main class of our `Tetris` game. Notice that it has instance variables for keeping track of a `BoundedGrid` and a `BlockDisplay` (which displays the contents of the `BoundedGrid`). In the constructor, it creates the `BoundedGrid` to have 20 rows and 10 columns, and creates the display. Go ahead and run `Tetris` to make sure the empty Tetris board is displayed correctly.

Exercise 1.1: The Communist Bloc

Shapes composed of four blocks each are called *tetrads*, the leading actors of the Tetris world. Tetrads come in seven varieties, known as *I*, *T*, *O*, *L*, *J*, *S*, and *Z*. They are shown here.



Open the `Tetrad` class, which keeps track of an array of four `Block` objects. (Note that the grid only keeps track of the blocks, and does not know anything about tetrads. Instead the `Tetris` class will eventually keep track of the tetrad being dropped.)

Recall that a `Block` is an object that keeps track of its color, location, and the `BoundedGrid` that contains it, all of which are properly managed by calling its `putSelfInGrid` method. Go ahead and complete the private helper method in the `Tetrad` class called `addToLocations`, which adds each of the four blocks (contained in the instance variable array `blocks`) to the four locations (in the parameter array `locs`) in the given `grid`, according to the following.

```
//precondition:  blocks are not in any grid;
//               blocks.length = locs.length = 4.
//postcondition: The locations of blocks match locs,
//               and blocks have been put in the grid.
private void addToLocations(BoundedGrid grid, Location[] locs)
```

(We won't be able to test this code just yet.)

Exercise 1.2: I, Tetrad

Now look at the constructor, which takes in the grid and initializes this `Tetrad` as an I-shaped block in the middle of the top row of the grid. Here is an outline of what the constructor needs to do:

1. Initialize the `Tetrad`'s array of `Block` objects so that it contains four new `Block` objects. There are two steps here; first instantiate the array to a length of four, and then initialize (load) each position in the array with a new `Block` object.
2. Declare a local temporary variable `color` of type `Color`, and set it to `Color.BLACK` by default.
3. Declare and instantiate a local temporary variable array `locs` of type `Location` with a length of four.
4. Declare a local temporary integer variable `shape` and set it equal to zero (the I-shaped tetrad).

5. Based on the shape number (from 0 - 6), assign a color to the temporary `Color` variable.
6. Also based on the shape number, assign the starting locations of that shape to the temporary array. For example, the locations (0, 3), (0, 4), (0, 5), and (0, 6) would make your `Tetrad` appear in a horizontal I-shape at the top of the grid.
7. Set the color of each `Block` to be the color indicated by the temporary `Color` variable.
8. Call `addToLocations` to add the blocks using the temporary array of `Location` objects.

Make sure you have designated the starting locations and color for each of the seven specific tetrads (shape numbers 0 - 6) and then branching statements based on shape number that will lead to the appropriate section of code. Remember that location (0,0) is in the upper left corner of the grid, and that the row and column numbers increase as you move down and right, respectively. You will want your tetrads to appear in the top row middle, so figure the starting locations of each tetrad accordingly. Each tetrad should be a unique color, but you may choose any colors that you like. The `Color` class has a list of predetermined color constants you can use (BLUE, CYAN, DARK_GRAY, GRAY, GREEN, LIGHT_GRAY, MAGENTA, ORANGE, PINK, RED, WHITE, YELLOW), or you can create your very own colors if you wish.

Finally, look at the instance variable in the `Tetris` class that stores the "active tetrad" (the one that's currently falling and being manipulated with the arrow keys, eventually). In the `Tetris` class constructor, this instance variable is initialized to refer to a new `Tetrad`. Now notice how the `play` method calls the `BlockDisplay` class `showBlocks` method, so that you can actually see the `Tetrad` you've created.

Go ahead and run `Tetris` to test that you can see an I-shaped `Tetrad` appear. When you are finished, demonstrate your work to me so that I can see you have completed this part of the assignment.