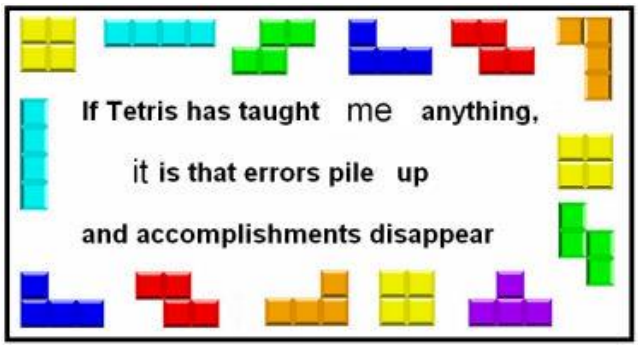
Tetris Lab



http://quoteinvestigator.com/wp-content/uploads/2013/06/culturerampmeme.jpg

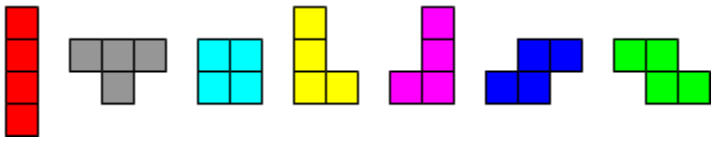
Exercise 1: Block Pop-Up Window

Create a new class called `Tetris`, which will be the main class of our Tetris game. It should have instance variables for keeping track of a `MyBoundedGrid<Block>` and a `BlockDisplay` (which displays the contents of the `MyBoundedGrid`). In the constructor, create the `MyBoundedGrid<Block>` to have 20 rows and 10 columns, and create the display. Use `BlockDisplay`'s `setTitle` method to set the window title to be "Tetris", and the `showBlocks` method to draw the window.

- Test to make sure empty Tetris board is displayed correctly. Get help if the board does not appear.

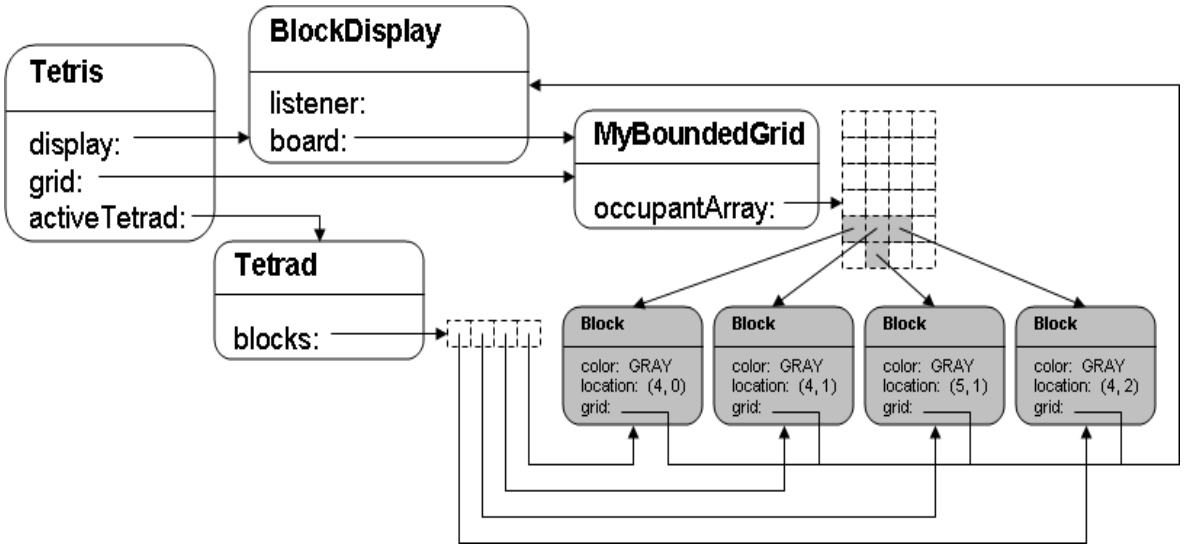
Exercise 2: The Communist Bloc

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 in their suggested colors.



Create a `Tetrad` class, which keeps track of two things: an array of four `Block`, and the `MyBoundedGrid<Block>` in which they live. An instance diagram showing the role of the `Tetrad` class appears on the next page. (Note that the environment only keeps track of the blocks and does not know anything about tetrads. Instead the `Tetris` class eventually keeps track of the tetrad being dropped.)

Tetris Lab



Go ahead and create the `Tetrad` class with appropriate instance variables. For example, as indicated by the diagram above, create an instance variable that is an array of four blocks that represents your tetrad blocks.

In `Tetrad`, give it a method called `addToLocations`, with the following signature:

```
private void addToLocations(MyBoundedGrid<Block> grid,
                            Location[] locs)
```

The purpose of this method is to put the tetrad blocks at the location given in the `locs` array. There is a precondition that the tetrad blocks are not in any grid. It is key that this method calls the `Block` method `putSelfInGrid` within a loop.

Now create the constructor, which takes in the grid as a parameter. The constructor should pick a random tetrad shape – one of the seven tetrad shapes. Use a different color for each shape, such as `RED`, `GRAY`, `CYAN`, `YELLOW`, `MAGENTA`, `BLUE`, and `GREEN`. Initialize the tetrad to appear in the middle of the top row of the environment. Be sure to use the `addToLocations` method. Remember that the constructor appears in the code after the instance variables are declared.

The next step is to create an active tetrad instance variable in `Tetris` – again, refer back to the diagram above. Initialize the active tetrad in the `Tetris` constructor.

- Now test that a random tetrad appears at the top of your tetris window. Make sure that your code can create all seven tetrad shapes. Get help if this is not happening.

Tetris Lab

Exercise 3: Lost in Translation

Add the following two methods to your `Tetrad` class.

```
private Location[] removeBlocks()

private boolean areEmpty(MyBoundedGrid<Block> grid,
                         Location[] locs)
```

The purpose of the helper method `removeBlocks` is to remove the tetrad blocks from the grid while returning the locations where the blocks were. Its precondition is that the tetrad blocks are in the grid. It is critical that the method uses the `Block` method `removeSelfFromGrid`.

The purpose of the helper method `areEmpty` is to answer the question whether or not all the locations in `locs` are valid and empty in the grid. It returns true only if all the locations are valid and empty.

Now write the method `translate`, making use of `removeBlocks`, `areEmpty` and `addToLocations`. The purpose of this method is to move the tetrad `deltaRow` down and `deltaCol` columns to the right, as long as the new positions are valid and empty. The return value is true only if the translate is successful. Here is its signature.

```
public boolean translate(int deltaRow, int deltaCol)
```

Before jumping into coding this method, think about how you are going to tap into the functionality of the following methods: `removeBlocks`, `areEmpty` and `addToLocations`. In particular, capture the `Location` array returned by `removeBlocks`. This returned array is a key to making this method work correctly.

- Test that you can translate a tetrad. Modify the `Tetris` constructor to translate the active tetrad, and make sure the tetrad appears in the correct location. Be sure to test a translation to an illegal position. If `translate` does not work, check that `areEmpty` works correctly – it is often the reason the culprit. Get help if `translate` is not doing its job.

Tetris Lab

Exercise 4: Block Party

Now that blocks have the potential to move, let's teach them to dance! The `BlockDisplay` class keeps track of an `ArrowListener`. Whenever the `BlockDisplay` window has "focus" and an arrow key is pressed, a message is sent to the `ArrowListener`. The `BlockDisplay` class doesn't actually care what the `ArrowListener` chooses to do with this message; hence `ArrowListener` has been defined as an interface, shown here.

```
public interface ArrowListener
{
     void upPressed();
     void downPressed();
     void leftPressed();
     void rightPressed();
     void spacePressed();
}
```

Modify the `Tetris` class so that it implements the `ArrowListener` interface. Each `ArrowListener` message should cause the `Tetris` game's active tetrad to move one row or column in the indicated direction. Be sure to call the display's `showBlocks` method to tell it to redraw itself whenever your tetrad moves. Test that your code handles when left arrow and right arrow are pressed at the grid's boundary.

When a space bar is pressed, a "hard drop" is triggered. The current tetrad drops straight down as far as possible.

In the `Tetris` class's constructor, when you create the `BlockDisplay`, call the `BlockDisplay` method `setArrowListener` so that the display can find the methods you just implemented. Check out the class `BlockDisplay` to find out the parameter it takes.

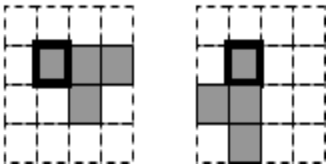
The "trick" is to realize that `Tetris` is the listener; therefore, in the constructor, when you set the `ArrowListener`, use the keyword `this` as the parameter. Isn't that cool?

- Now go ahead and test your code. A random tetrad should appear at the top of your Tetris window. You should be able to move it around with the arrow keys. Your program should prevent you from moving the tetrad outside of the window. Get help, if needed.

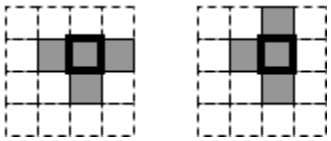
Tetris Lab

Exercise 5: Spin Cycle

Next, you'll make your tetrads rotate clockwise by 90 degrees about a particular point P_0. The following diagrams show a single rotation of a *T* tetrad for two different choices of P_0.

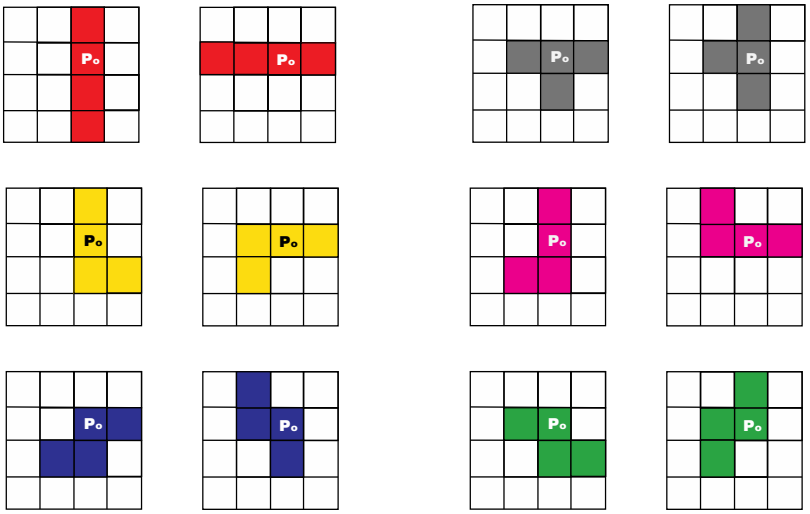


Leftmost block as P_0



Middle block as P_0

The game of Tetris uses the pivot point as shown on the right side; in other words, it uses the middle block to be P_0 for the *T* tetrad. The following illustrations show the pivot point – marked **P_0** – for six of the tetrad. For each pair of pictures, the left picture is the original orientation of the tetrad and the right picture illustrates the first 90-degree clockwise rotation. The *O* tetrad does not rotate.



In implementing your choice of P_0, one reasonable design is always to use `blocks[0]`. If you use this approach, you'll need to go back and look at your `Tetrad` constructor code, so that a centrally located block is always assigned to `blocks[0]`. Alternatively, you might keep track of the index in the `blocks` array where the P_0 block appears. However, as you approach this problem, be sure that each tetrad rotates about a centrally located point.

Given a block at position (*row*, *col*), there is a surprisingly simple formula (shown below) to find its new location (*row*', *col*'), following a 90-degree clockwise rotation about a point P_0 at (row_0, col_0).

$$row' = row_0 - col_0 + col$$
$$col' = row_0 + col_0 - row$$

Use these ideas to add the method `rotate` to the `Tetrad` class. This method's structure is almost identical to the `translate` method; meaning, you must use these methods: `removeBlocks`, `areEmpty` and `addToLocations`. The return value is true only if the rotation is successful. Here is its signature:

```
public boolean rotate()
```

Modify the `Tetris` class so that it rotates the active tetrad clockwise by 90 degrees whenever the up arrow is pressed (instead of shifting the tetrad up).

- Test to make sure your tetrads rotate appropriately, and that your game prevents you from rotating the tetrad off the edge of the window.

Exercise 6: The Sky Is Falling

Implement a method `play` in `Tetris`, which should repeatedly pause for 1 second (using the following code segment), move the active tetrad down one row, and redraw the display.

```
try
{
     //Pause for 1000 milliseconds.
     Thread.sleep(1000);
}
catch(InterruptedException e)
{
     //ignore
}
```

- When you test your program, you should find that you can still shift and rotate the tetrad, but that it now slowly drops on its own. When it gets to the bottom, the tetrad should stop falling (although you'll still be able to slide it around for now).

Exercise 7: Tetrad Comrades

Now modify the `play` method so that, when it is unable to shift the active tetrad down any further, it creates a new active tetrad. (Hint: Check `translate`'s return value.)

- Test your game and see how much you've accomplished!

Exercise 8: Death Row

Now it is time to add the code to clear any rows the user completes. Write the following helper methods in the `Tetris` class.

```
private boolean isCompletedRow(int row)

private void clearRow(int row)
```

The method `isCompletedRow` returns true if and only if every cell in a given row is occupied. The precondition is that `row` is in the range of [0, number of rows).

The method `clearRow` removes every block in a given row and every block above the row has been moved down one row. The `Block` method `moveTo` is great to use to move a block. Two preconditions exist for the method `clearRow`: the row is filled with blocks and `row` is in the range of [0, number of rows). (For those who know the game tetris, gravity is not in effect at this point. That is an additional option that you may add later.)

Now use the above helpers to implement the following `Tetris` method that clears all completed rows.

```
private void clearCompletedRows()
```

Whenever a tetrad stops falling, call `clearCompletedRows`. This method must clear all completed rows before exiting, even rows that are completed due to falling blocks.

- Now go play Tetris!

Tetris Lab

Beyond Exercise 8:

Exercises 1 through 8 must work correctly to earn at least a 96% in APCS A and 90% in APCS DS.

APCS A students must complete at least two additional options to earn 100%. Use the list below for ideas.

APCS DS students must implement the **Game End** as well as the **Score and Display** options. In addition, they must complete at least three more of the bolded options to earn 100%.

Have fun implementing any of the options or come up with your own ideas.

Additional Options:

- **Game End: end the game in a clean way, which does not include an infinite stacking of tetrads. Identify when a player has lost and respond accordingly.**
- **Score and Display: Keep score by tracking the number of rows cleared and displays a score. Additionally, the gridlines and scores are display, which require that `BlockDisplay` class is altered.**
- **Add background music.**
- **Increase the speed at which tetrads fall by introducing levels. The game begins with level 1. Every time the player clears 10 rows, advance to the next level. Blocks should fall a little faster on each successive level. Clearing 1 row earns 40 * *level* points. Clearing 2 rows at once earns 100 * *level* points. Clearing 3 at once earns 300 * *level* points, and 4 earns 1200 * *level* points. Show the player's level and score in the window title.**
- **Implement options to hold, pause, and restart the game. Instructions on how to use these options must be printed to the terminal window.**
- **Implement a look ahead by show what tetrad will be falling next.**
- **Swap out tetrad.**
- **Drop special kinds of blocks that act as bombs, etc.**
- When starting a new game, let the player choose to fill the bottom rows with random blocks.

Tetris Lab

- Use a gravity variant that supports chain reactions. See http://en.wikipedia.org/wiki/Tetris#Gravity.

- Improve the artwork, animation, effects, etc.

- In a new directory, implement Super Puzzle Fighter, or some other puzzle game.