EECS 1720
LAB 6 :: Exploring and Using MouseEvents, KeyEvents & Simple Threads (Timer)

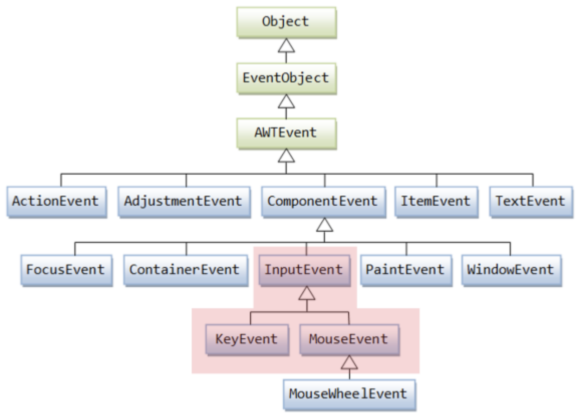
Prerequisite – labs 1-5

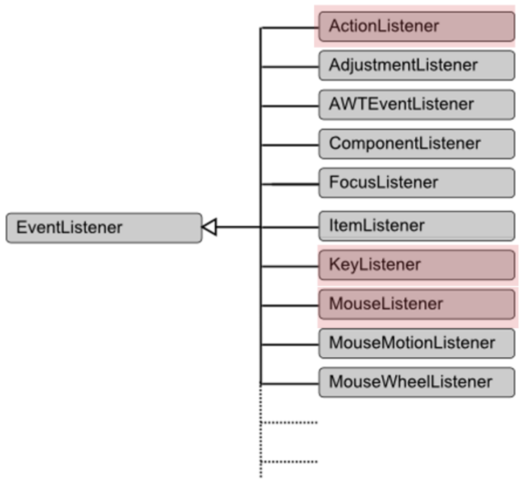
Lab Resources:

Java API: https://docs.oracle.com/javase/8/docs/api/
Java Swing: https://docs.oracle.com/javase/tutorial/uiswing/components/index.html

Lab Files: http://www.eecs.yorku.ca/course_archive/2022-23/W/1720/labs/lab6/lab6.zip

** A reminder of the Swing hierarchy of Events/Event Handler Interfaces.





STEP 1: Importing an Archived Project

In this step, you will import an archived project into your Eclipse workspace, (the files for each lab exercise are embedded). Each question refers to a separate java file. The instructions for what is required for each question is included in the document below (STEP 2 onward). It is a good idea to create a separate workspace (e.g. "EECS1720") for this course.

a. You can download the lab project file from the link above (see Resources) This file is a *zip file*, also known as an *archive file*. Click on the link below to open the URL in your browser. In the dialog that opens, choose **Save**, not **Open**. This file is typically saved into your *home* or *downloads* folder.

b. Open Eclipse and import this archive

IMPORTANT: this process is the same as the process you will use in lab tests, so please make sure you follow it, so that you do not have submission difficulties during the lab tests.

DO NOT DOUBLE CLICK ON THE FILE TO OPEN. By importing this project, you will add it into your *EECS1720* workspace. Do this by:

i. Open Eclipse (**Applications → Programming → Eclipse**) & set/choose your workspace
ii. Close "Welcome" tab, and navigate to **File → Import → General → Existing Projects into Workspace.** Hit "next"
iii. Choose the archive file (zip file you downloaded in (a)). After selecting, hit **Finish**, and the file will open up as a project in your Project Explorer.
iv. We are now ready to proceed with the Lab Exercises.

STEP 2: Lab Exercises

The game Tetris is a puzzle game where the player can move and spin falling blocks of various shapes. The goal of the game is to stack the blocks without leaving any gaps between the blocks. In this lab, you will complete some components of the classes relating to modeling the "blocks" we see in a typical game of Tetris, and will add some simple event-based handling (to keys and mouse) to allow these blocks to be "manipulated" (rotated/moved) on the screen, along with some basic functionality to animate the blocks.

If you have never played Tetris before, there is a wiki with lots of information about the game. There are 7 standard blocks (called Tetriminoes). The blocks are all defined on a square grid of size 3-by-3 or 4-by-4.

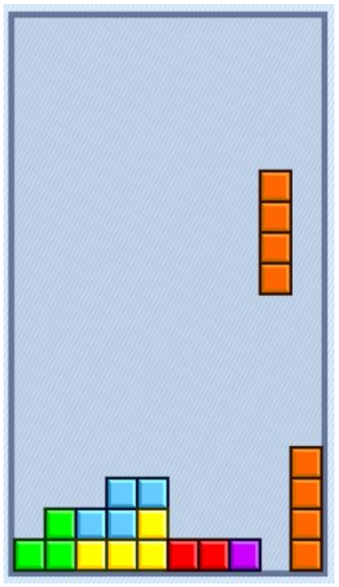
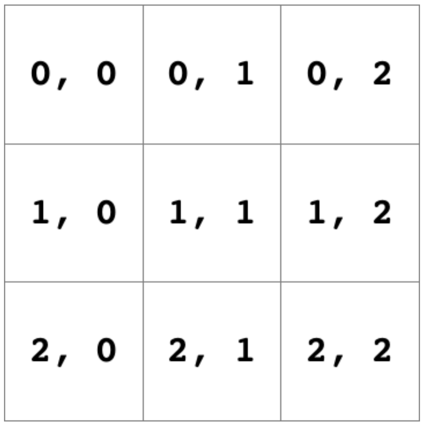
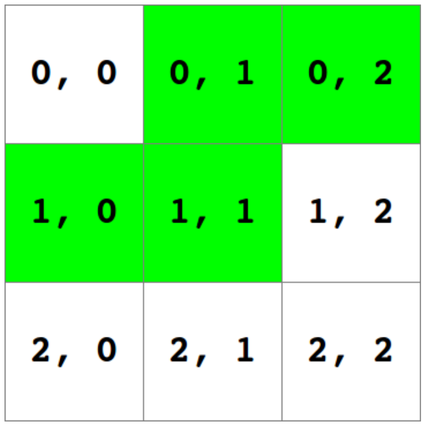


Figure 1: Tetris game (playing field)

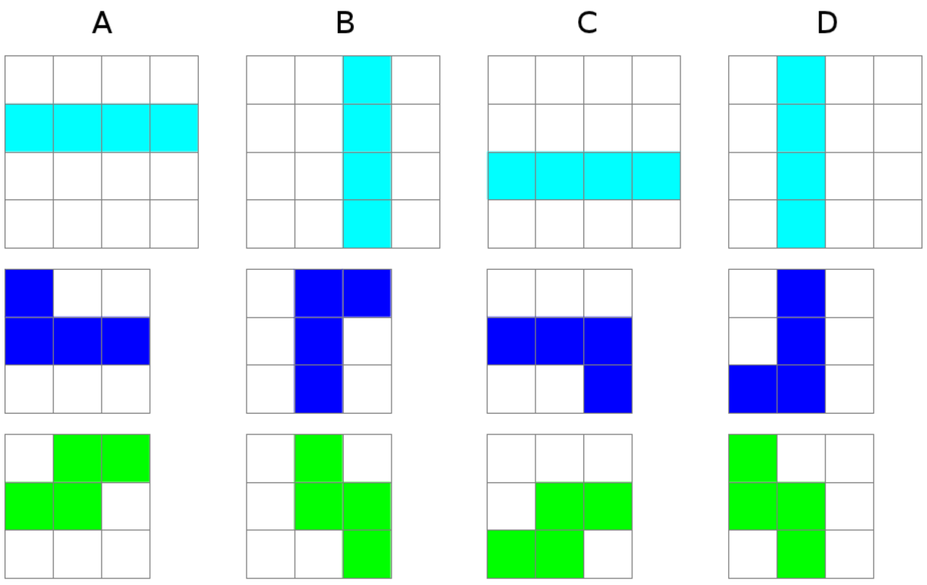
In this lab, you are given a class `BlockGrid` that helps you describe the shape of a block. `BlockGrid` lets you get, set, and clear the individual grid positions. A grid position is specified using a pair of zero-based integer indices (the row index and the column index). The figure below shows the indices for each grid location on a 3-by-3 `BlockGrid`:



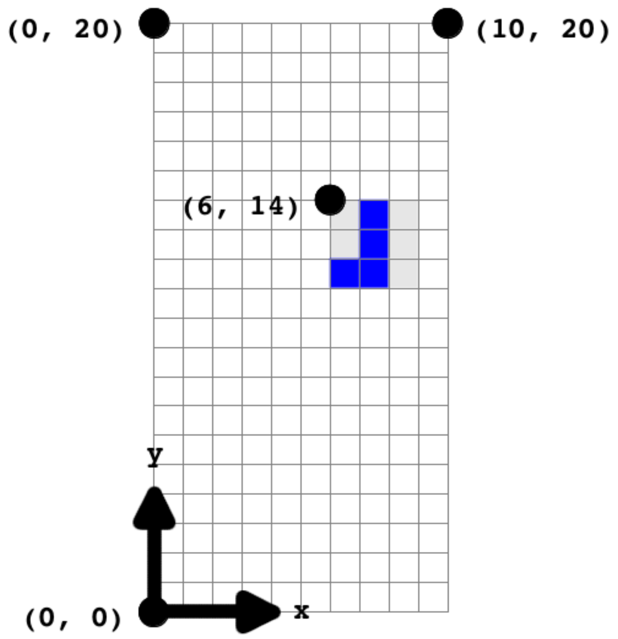


The shape of each block can be represented using a `BlockGrid` object. The figure above (right) shows the `BlockGrid` for an S-block; the white grid locations have the value `false` and the green grid locations have the value `true`.

The player is able to spin a block into any of 4 possible orientations. The figures below show the 4 possible orientations for the I-block, J-block, and S-block. Column A show the blocks in their starting orientation. Column B shows the blocks after they have been spun once to the right. Column C shows the blocks after they have been spun again to the right. Column D shows the blocks after they have been spun again to the right. Spinning the blocks once more returns the blocks to Column A. If the player spins the blocks to the left starting from Column A, then the orientations are given by Column D, then Column C, then Column B, returning to Column A.

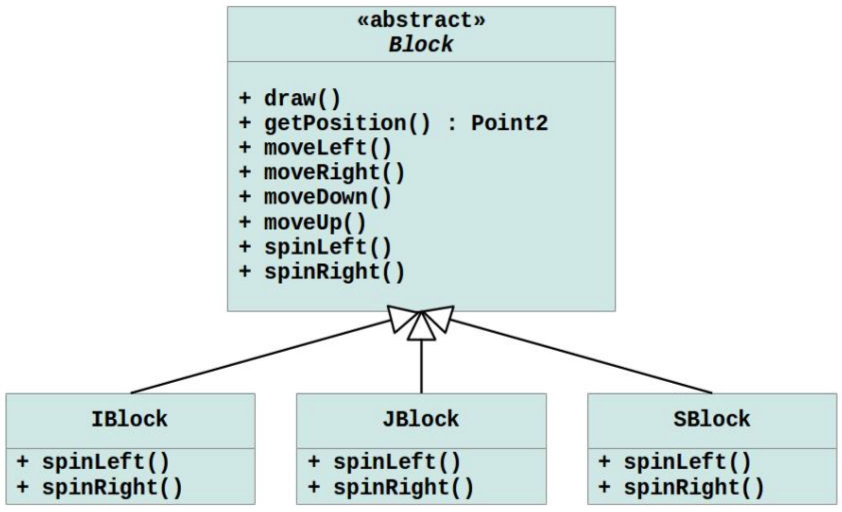


The Tetris playing field is typically a grid with 20 rows and 10 columns. The location of a block on the field is given by the coordinates of the top-left corner of its grid. The image below shows a J-block with position `(6, 14)` on the playing field.



For this lab you will first complete the implementation of three types of blocks: `IBlock`, `JBlock`, and `SBlock`. All Tetris blocks have three essential features: (1) they can be drawn on the playing field; (2) they can be moved; (3) they can be spun.

The information needed for every type of block such as the location of the block on the playing field, the color of the block, and the shape of the block; this information can be managed by an abstract base class `Block` that all blocks will extend. The relationships between the classes are shown in the following UML diagram:



Note, in the above definition, the Block class is already completed for you, however you will need to complete the child classes (IBlock, JBlock and SBlock) – namely their constructors, and overridden versions of spinLeft() and spinRight() methods. Please refer to notes on abstract classes (and the video walkthrough for this lab) – recording in lab session and to be posted on eClass.

Essentially an abstract class is one that is either explicitly declared as "abstract", and may have one or more methods that may also be declared as "abstract". This means that those abstract methods are only declared (and are not defined). As such, abstract classes function much like an interface (that also has partial definition). For the abstract features, these would need to be defined explicitly (made concrete) by a child class. So if a child class extends an abstract class, for it to be fully concrete, it must have a full implementation of every abstract feature inherited. **Only classes that have a full implementation can be instantiated as objects**. In this lab, we want to instantiate each of the child classes (as a specific, concrete sub-type of Block).

Exercise 01

In this lab, the first task is to complete the children classes (IBlock, JBlock and SBlock). To complete the Block class, you first need to complete the constructors for the IBlock, JBlock and SBlock classes, along with their spinLeft() and spinRight() methods.

The constructor for IBlock already completed for you as a reference. Note that the figure showing the rotated versions of the IBlock above (bottom of page 3 in this document), forms the basis for the constructor, which makes use of the existing BlockGrid class to instantiate and store 4 separate BlockGrid's to hold the shapes of the IBlock when rotated. These are each stored in an ArrayList called "grids". There is also a colour, position, blockgrid size, and gridIndex associated with each block (some of these variables are used to initialize the parent's class fields). gridIndex is a variable that indicates which version of the rotated shape is currently representing the IBlock (so a value of 0 represents the first state (the horizontal or A position of the IBlock in the above figures), while the gridIndex 3 represents the last D position of the IBlock).

The same is true of the SBlock and JBlock classes (though their states refer to 3x3 blocks illustrated in the same previous diagram – at the bottom of page 3 of this document). Complete the constructors for the classes IBlock, JBlock and SBlock, first, then run the TetrisBlockDemo.java file, and if working, a random block should be positioned near the top of the playing field. Each time you run, a different block should display (run enough times to confirm all 3 blocks are initialized and begin with their positions according to position A.

HINT: the constructors for JBlock and SBlock should create 3x3 (not 4x4) GridBlocks, then should use the mutators (setters) to set the appropriate grid squares for each rotated version of the block. The spinLeft() and spinRight() methods should be similar for all blocks as they all contain 4 states to switch between. A=>D if spinning left and D=>A if spinning right (for all blocks). Also, you can test your blocks as soon as you complete the constructors (as the spin behaviour will be triggered later with event handling)

In the remaining exercises, you will be working with the TetrisBlockDemo.java file, in order to add functionality to manage user interaction with the keyboard (key events) as well as mouse, then finally manage the animation of a block falling on the playing field (before it re-spawns after falling off the app window at the bottom). This is NOT a complete implementation of Tetris, but just adds some interaction and tests this interaction for managing the block behaviour while falling in the game of Tetris. No behaviours dealing with how the blocks aggregate/connect at the bottom to score points is handled in this lab.

Exercise 02
(Using KeyEvents to manage the spinLeft/spinRight behaviour of a block)

In this exercise, you will complete the implementation of the KeyListener interface for the TetrisBlockDemo class. Note, this is started for you (a shell of the methods that could be used is provided).

The behaviour should see the following key presses do the following:

"A" key => when pressed, should move the block (stored in block class field) left by 1 grid position (i.e. should invoke the moveLeft() method of the block)

"D" key => when pressed, should move the block right

"J" key => should spin the block left

"L" key => should spin the block right

"S" key => should set toggle the isFalling state from true to false, or false to true (this field will be used in Exercise04.

** remember the KeyListener should be registered against the JFrame ideally (and this should be set to have the focus, prior to being used. See lecture notes for more info.

Test your implementation, by running the TetrisBlockDemo.java file, and pressing keys. The drawn shape can be updated with a call to **this.playingField.repaint().**

Exercise 03
(Add events for mouse clicks)

In this exercise, you will add additional functionality for the mouse to achieve the spin left and spin right behaviours.

Essentially, a mouse click that occurs to the left of the blocks current position, should spin the block left, while a mouse click that occurs to the right of the current block position, should spin the block right. This behaviour should not interfere with the existing key events (i.e. both should work to achieve the same spin functionality). Note: you will need to reset the focus to the JFrame in order to capture key events and redirect them properly after clicking a mouse button.

Remember to register your mouse listener to the JFrame (or some aspect of it).

Exercise 04
(Add animation capability)

In this task, you should now complete the setup of a Timer thread, and register it to send regular action events for updating the position of the block and animate it when in the falling state.

The actionPerformed method is started for you, you will need to modify the constructor to instantiate a Timer, and configure it appropriately. Lecture 20 includes notes on the Timer object and how to use it. We will cover this in lectures (in the meantime, you can refer to those lecture notes).

Some hints are given in the comments of the TetrisBlockDemo class.

Essentially, after pressing the “S” key (setup in Exercise02), you should be able to see the block that was initially set at the top of the playing field, slowly start to fall on the screen (step in the downward direction). To enact this, you will need to update the block by invoking one of its mutator methods (already defined).

The paint() method of the TetrisCanvas class (also already completed for you) should take care of the painting of this object, which can also be invoked with a call to **this.playingField.repaint().**

Exercise 05 (if bored)

Not required, but if you finish this lab and want something to work on, you may think about how you would re-formulate/use the provided classes, to re-organize this application using the MVC design pattern discussed in lectures. I.e. create 3 new classes, TetrisModel, TetrisView, and TetrisController (that utilize all of the functionality discussed in the previous exercises, but effectively decouples the state from the gui, from the event handling).

STEP 3: Submission (Deadline: Friday March 31st , 2023 - 11:59pm)

You will formally submit the java files associated with all Exercises 01-05. SUBMIT ALL OF THE *.java FILES (**NO ZIP FILES OR CLASS FILES!!**).

Please submit your completed ***.java** files to “**lab6**” directory via web-submit.

Web-submit can be found at the link: https://webapp.eecs.yorku.ca/submit/