CSE 11 Winter 2015 **START EARLY!**
Program Assignment #5 (100 points)
Due: 23 May 2015 at 1159pm (2359, Pacific Standard Time) SATURDAY!

PROGRAM #6 : BlockGrid

READ THE ENTIRE ASSIGNMENT BEFORE STARTING

The program focuses on using java Swing and Threads. You will be modifying a program to achieve different results. The goal is for you to become more familiar with a non-trivial Swing program, then modify it to achieve a simple animation.

The supplied class was taken from an online post about how to create a logical grid so that each x,y coordinate in the grid is really a rectangular box. The program can only make a grid of a specific size, using hard-coded constants, and other items. The first thing you need to do is *understand* how this program works. You need to figure out how the classes (all coded as helper classes) work with each other, how the method `paintComponent()` is utilized, how the grid is created, and more.

BlockGrid – As supplied for your homework

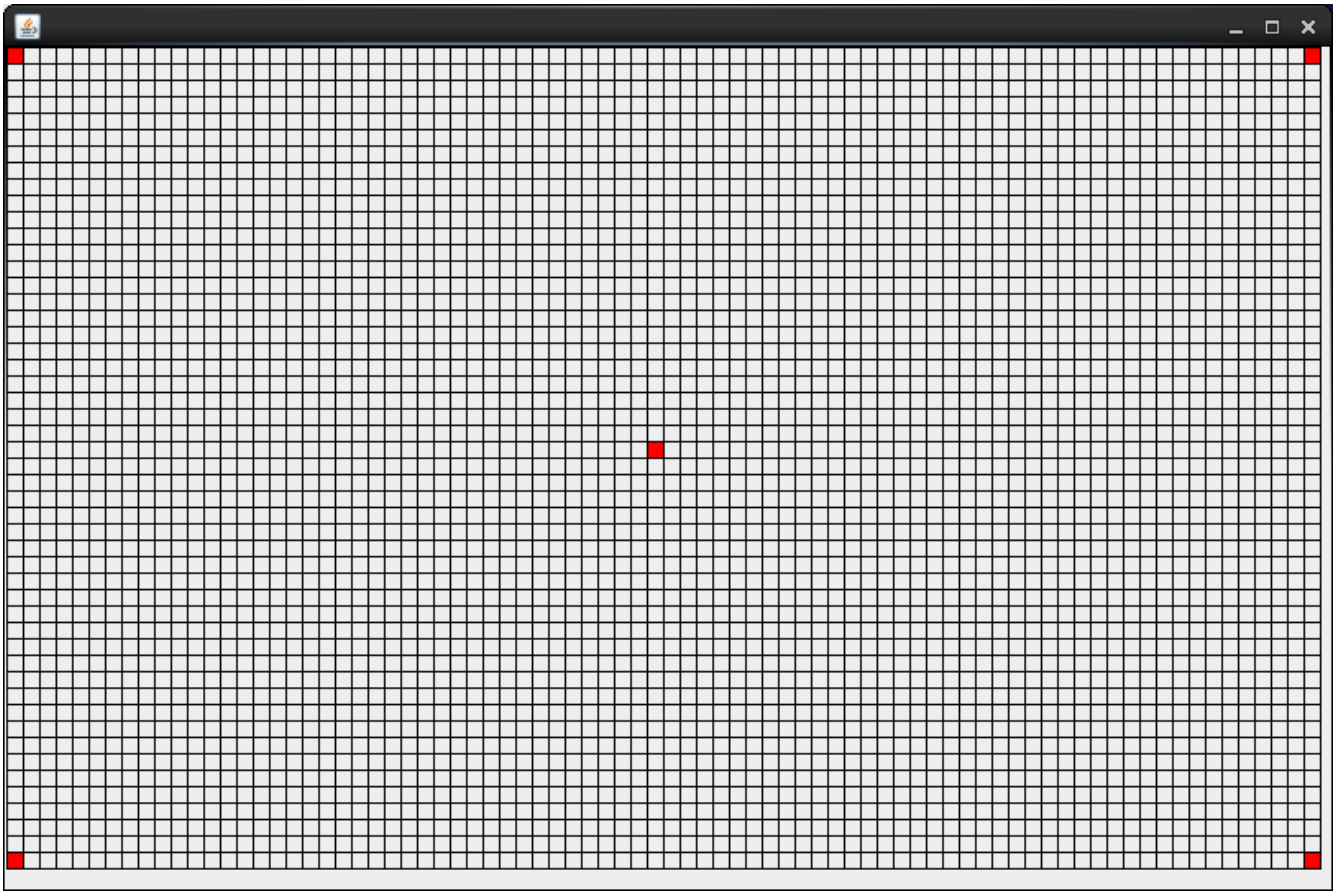

Figure 1 - BlockGrid. As Supplied for the homework

Figure 1 shows BlockGrid. In a nutshell, what you are looking at

- A grid of 10x10 pixel cells
- The grid is 800 pixels wide and 500 pixels tall
- The grid is logically 80 x 50
- Each corner of the grid is filled with a red-colored cell
- The "center" of the grid is also a box colored red.

If you look at the implementation, you will notice that the grid and cells are inside of java Swing JPanel object.

Modifying BlockGrid Part 1

There are three sets of major modifications that you need to make to BlockGrid. You should do these one at a time.

Modification 1: Make BlockGrid generate any M x N logical grid, where the cells are K x K pixels and then fills in the corners and "center" pixel with red squares. The actual grid will be (M*K) x (N*K) pixels. BlockGrid should take a command line arguments.

- Here is the example usage statement (which your modified program must provide)

```
$ java BlockGrid help
Usage: BlockGrid [ width height pixels ]
```

The parameters are as follows

- width = logical number of blocks in the width direction
- height = logical number of block in the height direction
- pixels = size of each grid square

Defaults: If no arguments are given to BlockGrid, then it should default to

- width = 40
- height = 20
- pixels = 20

This creates a 40x20 logical grid. The grid is 800 pixels wide X 400 pixels high

Errors that you must check for:

- width, height, pixels must all be >= 2

⇨ If any of the above errors occur, print the usage statement and exit.

Other errors that you must handle

- If neither 0 nor exactly 3 arguments are given
- If a the user types in non-integers as arguments for any of width, height or pixels (use Exception handling to make this nearly trivial)

Example Outputs of Modification 1:

- BlockGrid called with Defaults: `java BlockGrid`

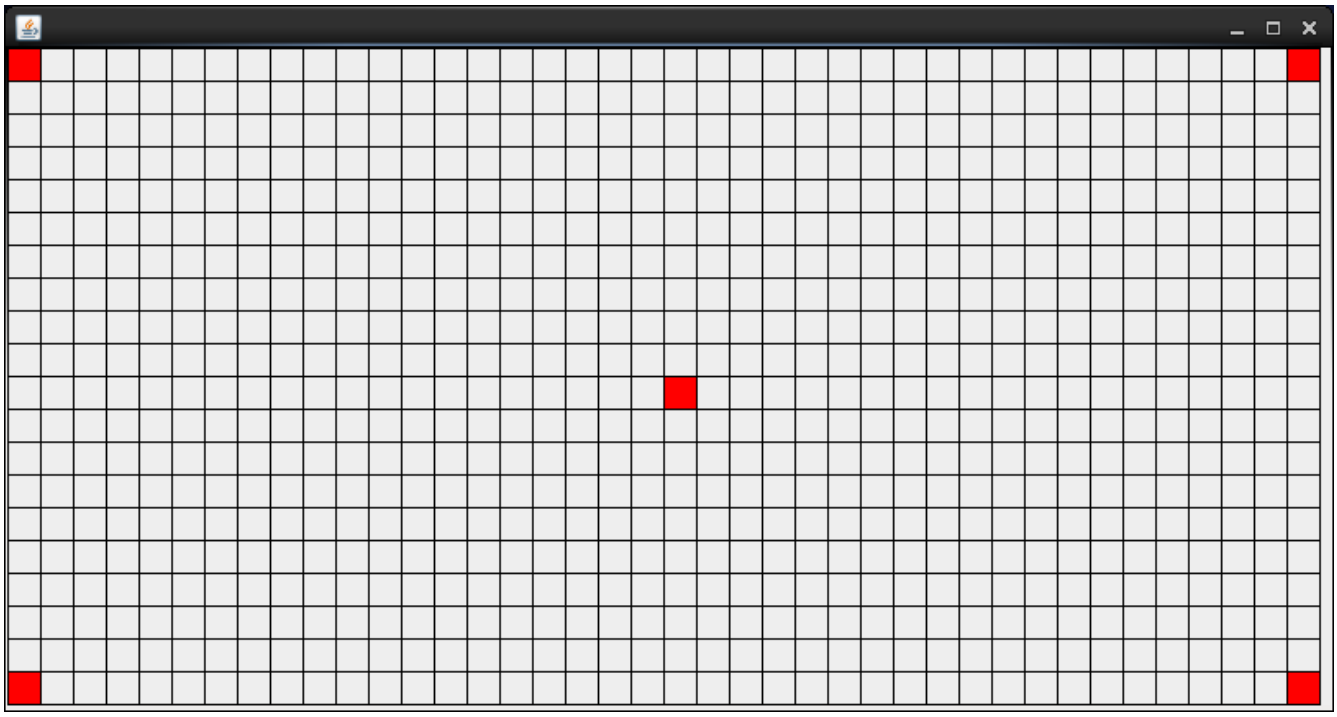

Figure 2 Modified BlockGrid with defaults (Notice that the center row could be the row shown or the one above, either is acceptable)

- BlockGrid that is 400x400 with 5 pixel blocks

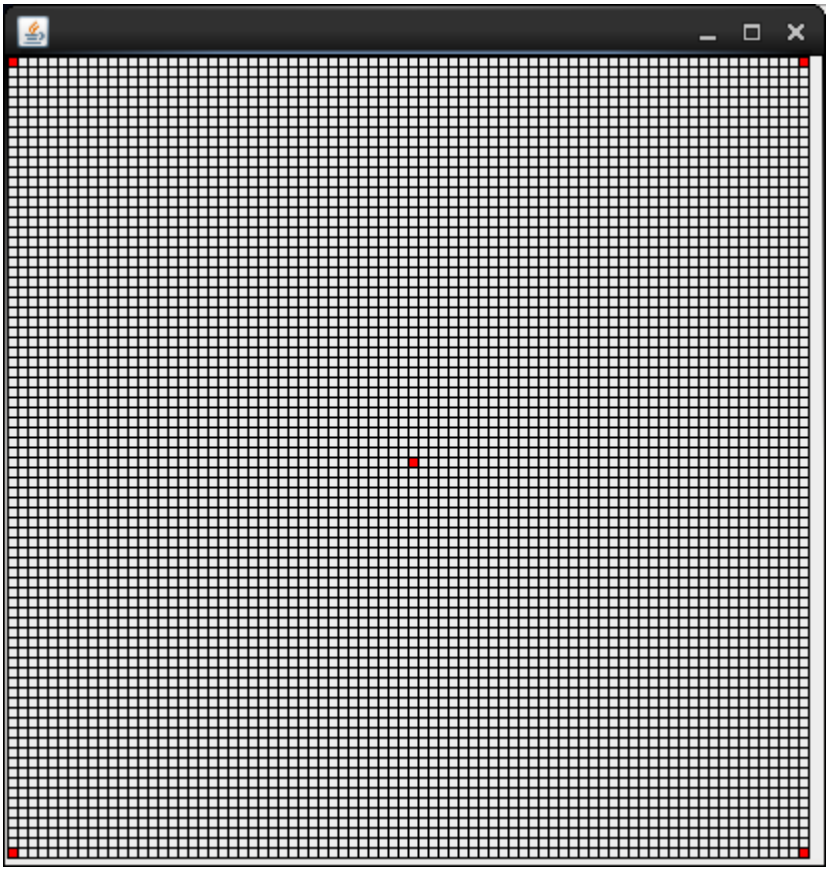

Figure 3 Modified BlockGrid at 400x400 pixels with 5 pixel blocks

Hints for part 1:

- You will have to write new constructor both the `MyWindow` and `Grid` classes.
- You may need to add some public methods in either `MyWindow` or `Grid` to make part 2 work more easily. You may find the `Dimension` and `Point` Classes (already defined by AWT) to be useful

Modifying BlockGrid Part 2

The goal of the second modification is to make a 2x2 box move back-and-forth (left-to-right, then right-to-left). You will also remove the colored blocks at the corners of grid. When your modified BlockGrid starts up with the default settings it should look like the following picture

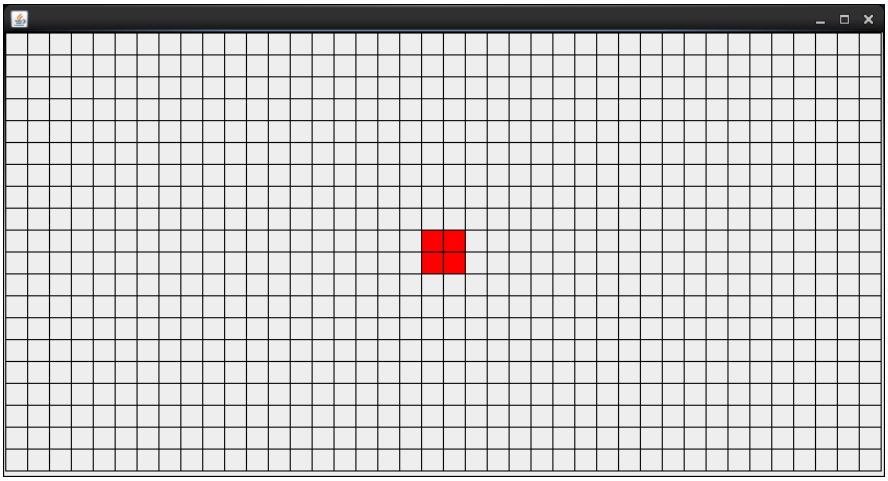

Figure 4 - BlockGrid after second modification using default 40x20 grid with 20 pixel cells.

In this Figure , you will notice that 2 x 2 box is placed in the "center" of the grid. To compute where to place the box, the upper-left corner of the box should be in logical grid location:
(width/2 – 1), (height/2 – 1).

This grid uses graphics coordinates, the upper left box is (0,0), the lower right hand box (width-1,height-1)

Once you have created the initial box, it should move from left-to-right, when the box would hit the right hand side of the grid, the box should reverse direction and move right-to-left. When the box would hit the left side of the grid, it should again reverse direction and move left-to-right. It should continue doing this until the user exits the program.

To accomplish this animation, you should create a new helper class called Mover(). A single Mover instance **must be run in a separate thread**. Before writing Mover, you should modify your Grid class to have a new public method called `clearCell()`. It should do the opposite of `fillCell()` in that it should it should "clear" the logical cell given as an argument.

How Mover should work

- When constructed, the Mover instance should be initialized with which cell is the "upper left of the box" (it should NOT recalculate this), and a reference to the constructed Grid instance.
- The box should move right –to-left until the right-hand side of the box gets to the right-most cell, it should then reverse direction, until it hits the left-most cell, then reverse again. The box bounces back and forth from side-to-side..
- In a loop, until told to stop it should
    - fill the appropriate cells (using `fillCell` method defined in Grid)
    - wait 150ms (code this as a constant)
    - clear the cells
    - define where the next cells should be filled (notice that the loop will redraw four cells that define the box immediately.
- The loop must be coded in the run method of the Mover. (that is, the motion of the

cell is controlled by the thread).

Hints:

- For your program to exit, the Mover thread must be told to stop. See sample programs from lecture to see how you might accomplish that task
- We haven't covered the `for (Point fillCell : fillCells)` construction in class. This says that a for loop should be executed where each time through the loop an element in the `ArrayList<Point> fillCells` is used.
- The `remove()` method of `ArrayList` (Lists in general) can be used to remove the appropriate element from the `ArrayList`

Modification Part 3

⇨ Comment your public methods and public class with Javadoc-style comments. See many of your previous assignments for examples

Make Copies of your Program Files as you go along, If you make a big mistake you can go back to the previously working code

Turning in your Program

YOU MUST BE ON THE LAB MACHINES FOR THIS TO WORK. PLEASE VERIFY WELL BEFORE THE DEADLINE THAT YOU CAN TURNIN FILES

You will be using the "bundlePR6" program that will turn in the file

BlockGrid.java

No other files will be turned in and they **must be named exactly as above.** BundlePR6 uses the department's standard turnin program underneath.

To turn-in your program, you must be in the directory that has your source code and then you execute the following

$ **/home/linux/ieng6/cs11s/public/bin/bundlePR6**

The output of the turnin should be similar to what you have seen in your previous programming assignments

You can turn in your program multiple times. The turnin program will ask you if you want to overwrite a previously-turned in project. **ONLY THE LAST TURNIN IS USED!**

Don't forget to turn in your best version of the assignment.

Frequently asked questions

What if my programs don't compile? Can I get partial credit? No. The bundle program will not allow you to turn in a program that does not compile.

Is there a video of various inputs? See the download site.

If there are an even number of blocks in the width or height, there isn't an exact center, what should I do? use integer division to find the closest block to the center. See the details in the assignment

My Grid doesn't exactly fit in the JFrame, what should I do? Get it as close as possible. When your Grid is added to the Window, the Window should be packed, using the pack() method defined in AWT.

What public methods can I add? Anything you want (methods and constructors). Don't use public class or instance variables.

Why are we doing this program? Roughly, this is preparation for program 7, where moving multiple blocks will be required.

If the user just give bad input, do just print usage and no other error message? Yes. Just the usage message. You may print the specific Exception (if one was generated) to System.err.

Do I have to check for all kinds of crazy inputs? Programming with Exceptions can help make this a fairly trivial task.

Will you grade program style? Yes. In particular, indentation should be proper, variable names should be sensible. We will also look for code clarity, too. Overly long or complex codes are frowned upon. The initial size of the code is 95 lines, the professor's code (without Javadoc comments) is 205 lines.

I don't understand paintGraphics or how the original program really works, can you explain? paintComponent() is how AWT/Swing redraw a graphics screen. AWT decides when to ask your component to repaint itself (via paintComponent()). It might decide to this if you move the window on your screen, if it becomes uncovered via mouse click or any number of (events) reasons. paintComponent() can be called at anytime. In the sample code, the method repaint() is called. This puts a repaint() request onto Event Dispatch Queue (controlled by AWT) and then returns. When the Dispatch thread gets around to handling the repaint request, your paintComponent method will be invoked.

What does SwingUtiltilities.invokeLater() do? When you give invokeLater and object reference (say its called myGrObj), it puts a request onto the Event Dispatch Thread to invoke the run() method of myGrObj (only objects that implement Runnable are valid objects for invokeLater). At some point in time "later", the run() method of myGrObj will be called.

What IS the Event Dispatch Thread? Graphics programs have all kinds of events (mouse movement, keyboard, windows closing, resizing, etc). The Dispatch thread is in charge of informing Objects that a particular event has actually occurred. When an event occurs (say a mouse press), the event is put onto the event dispatch queue (We call this enqueing a request

to the dispatch thread). Think of the queue as an inbox of work to-be-done. The Event Dispatch Thread takes an event off its dispatch queue (out of its inbox) and tells every Object that has registered interest in that particular event that the event (and its particulars) has occurred. Say it is a MousePress event, and three Objects have interest in the MousePress event, each of the three Object's appropriate handler is called. They are called one-at-a-time. When all of the handlers of all the Objects have completed for a particular event, the dispatch thread goes on to the next event in its dispatch queue.

How should I exit the program? Prompt in main() for the user to hit a key.