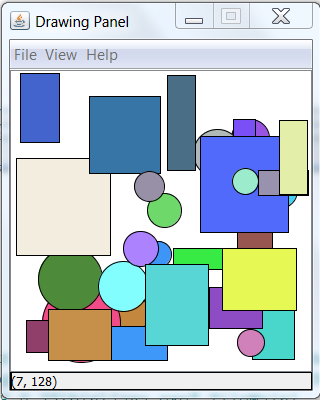
Assignment 11 – Tile Manager

### Overview

In this assignment you will write parts of a graphical program that will display tiles of varying shapes. At the program’s start the screen will be populated with a certain number of tiles. The tiles will be drawn on the screen in an order from back to front, with later tiles potentially obscuring all or part of tiles that were drawn earlier. This back to front “3 dimensional” ordering of the tiles is also known as *z-ordering*.

The graphical user interface (GUI) for this program will allow the user to manipulate the tiles. The user will be able to

* add more tiles to the screen
* shuffle the tiles already existing on the screen
* raise or lower a specific tile’s z-order
* remove a tile from the screen, or
* remove all tiles under a point from the screen.

### Learning Objectives

* Practice previous concepts: procedural decomposition, variables, datatypes, no magic numbers, good naming, loops, scoping, parameters, return values, using objects, fencepost loops, cumulative sum/product or min/max loops, roundoff errors, if/else statements/nested if/else statements, printf/format, Javadoc with pre and post conditions where needed, while/do-while loops, Random objects, Boolean operations, file I/O, try/catch blocks, line and token processing, arrays, reference semantics, object oriented programming with inheritance, polymorphism, abstract classes and interfaces, as well as unit testing and robust programs!
* Utilize techniques from Unit 11 such as
  + ArrayList
  + Comparing using the Comparable Interface

### External Requirements

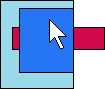
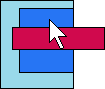
Part of your program's score will come from its "external correctness." External correctness measures whether the output matches exactly what is expected. We are very picky about the output matching exactly. In this program, the appearance of the tiles and the behavior of the program must match exactly. Programs that do not compile will receive no external correctness points. The specific requirements for external correctness are detailed below.

* To begin, you must download the supporting files from the course website, INCLUDING the new DrawingPanel.java. Once you can compile all of them you will then run TileMain.java to start the fully functional GUI. TileMainMini.java is provided so you can run a partial GUI for Milestone 1.
* When TileMain runs, it will create the graphical window on the screen, initially populated with the 20 tiles (as defined in TileMain or TileMainMini). The tile’s position, size and color are randomly generated by TileMain.
* The tiles will be drawn from back to front on the screen. For example, consider the tall (blue) that overlaps the (purple) circle and wider (green) rectangle in the screenshot at right. Part of three tiles occupy some of the same (x, y) pixels in the window but the wide one was drawn first because it occurred before the tall one in the list, and thus it is partially obscured by the tall rectangle which was drawn later. It is said that the blue rectangle has a different z-order than the circle or wider green rectangle.

The user interaction with the GUI is as such:

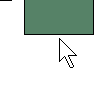
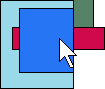
Note that if you use a Mac with a 1-button mouse, you can simulate a right-click with a Ctrl-click (or a two-finger tap on the touch pad).

* + If the user clicks the **left mouse button** while the mouse cursor points at a tile, that tile is moved to the very *top* of the z-ordering (the end of the tile list).



* + If the user clicks the **left mouse button** and holds down the **Shift key** while the mouse cursor points at a tile, that tile is moved to the very *bottom* of the z-ordering (the start of the tile list).



* + If the user clicks the **right mouse button** while the mouse cursor is pointing at a tile, that tile is removed from the tile list and disappears from the screen.
  + If the user clicks the **right mouse button** and holds the **Shift key**, *all tiles that occupy that pixel* are removed from the tile list and disappear from the screen.
  + If the user types the **N key** on the keyboard, a new randomly positioned tile is created and added to the screen.
  + If the user types the **S key** on the keyboard, the tiles' order and location are randomly rearranged (*shuffled*).
  + If there is no tile where the user clicks, nothing happens.
  + If the user clicks a pixel that is occupied by more than one tile, the top-most of these tiles is used. (Except if the user did a Shift-right-click, in which case it deletes all tiles touching that pixel, not just the top one.)

### Internal Requirements

Internal correctness means that your program uses the programming elements and structure that are detailed in the list of requirements below.

* Your files must be called TileManager.java and RectangleTile.java.
* You may modify the class constants in TileMain (or TileMiniMain) but your program must be able to run with these files in their unmodified condition.
* Your TileMain (or TileMiniMain) will interact with the TileManager that you write to handle the various behaviors of the tiles on the screen.
* You must correctly use the ArrayList collection class in your TileManager solution.
* The tiles shown on the screen are all subclasses of the class Tile. We have provided a sample subclass called CircleTile.java.
* Your TileManager class should store a list of Tile objects as a **field of type ArrayList**. Remember to import java.util.\*; to use ArrayList.
* The order of the tiles stored in the TileManager’s list determines tiles’ drawing order.
* Note that **your code does not need to directly detect mouse clicks or key presses**. Code in TileMain.java detects any user input. When user input is detected, methods that you define in TileManager.java are called.
* The methods in TileManager.java are to be implemented as described in the box on the next page.
  + The three methods (constructor, addTile and drawAll) comprise the basic functionality of the program.
  + The other five methods are called by the graphical user interface ("GUI") in response to various mouse clicks, passing you the x/y coordinates where the user clicked. If the coordinates passed do not touch any tiles, no action or error should occur. After any click, the GUI will clear the screen for you and call drawAll to re-draw all of the tiles in your list.
* A major focus of our style grading is **redundancy**. As much as possible, avoid redundancy and repeated logic in your code. One powerful way to avoid redundancy is to create "helper" method(s) to capture repeated code. It is legal to have additional methods in your TileManager class beyond those specified here. For example, you may find that multiple methods in your class do similar things. If so, you should create helper method(s) to capture the common code. (You should declare such methods to be private rather than public, so that outside code cannot call them.)

public **TileManager**()

This constructor is called every time a new tile manager object is created. Initially your manager is not storing any tiles.

public boolean **addTile**(Tile tile)

This method will add the given tile to the end of your tile manager's list of tiles. Returns true if the tile was added, false if the Tile object passed in was null.

public void **drawAll**(Graphics g)

This method should cause all of the tiles in the tile manager to draw themselves on the screen using the given graphical pen. You do not need to do this yourself directly by calling methods on the Graphics object; each subclass of Tile object already has a draw method that it can use to draw itself. Draw the tiles from bottom (start) to top (end) of your manager's list.

Recall that in order to refer to type Graphics, you must import java.awt.\*; in your code.

public void **raise**(int x, int y)

Called when the user left-clicks. It passes you the x/y coordinates the user clicked. If these coordinates touch any tiles (you can check via isHit()), you should move the topmost of these tiles to the very top (end) of the list.

public void **lower**(int x, int y)

Called when the user Shift-left-clicks. If these coordinates touch any tiles, you should move the topmost of these tiles to the very bottom (beginning) of the list.

public void **delete**(int x, int y)

Called when the user right-clicks. If these coordinates touch any tiles, you should delete the topmost of these tiles from the list.

public void **deleteAll**(int x, int y)

Called when the user Shift-right-clicks. If these coordinates touch any tiles, you should delete *all* such tiles from the list.

public void **shuffle**(int width, int height)

Called when the user types S. This method should perform *two actions*: (1) reordering the tiles in the list into a random order, and (2) moving every tile on the screen to a new random x/y pixel position. The random position should be such that the square's top-left x/y position is non-negative and also such that every pixel of the tile is within the passed in width and height. For example, if the width passed is 300 and the height is 200, a tile of size 20x20 must be moved to a random position such that its top-left x/y position is between (0, 0) and (280, 180).

You can use the built-in Java method Collections.shuffle to randomly rearrange the elements of your list.

* Your manager's list stores Tile objects. The Tile class we gave you is an abstract class, that is, there are some methods that are not implemented in that class. We have provided a sample concrete subclass of Tile called CircleTile.
* The Tile class is defined to have the following public methods:

public int getX()

public int getY()

public void setX(int x)

public void setY(int y)

These methods return or modify top-left x/y pixel position of the bounding box of the tile.

public int getWidth()

public int getHeight()

These methods return the tile's bounding box’s width and height in pixels.

public Color getColor()

Get the color of the Tile.

public abstract boolean isHit(int x, int y)

Figures out if the x/y coordinate being passed in is within the bounds of the actual Tile (not the Tile’s bounding box)

public abstract void draw(Graphics g)

Draws the tile on the screen using the given graphical pen.

public String toString()

Returns a text representation of the tile, such as "(x=57,y=148,w=26,h=53)".

It can be useful to print a tile (or a collection of tiles) with println to examine their state. Subclasses of the Tile class should print out the type of Tile it is, then the coordinates and dimension. For instance the circle would print out "Circle: (x=57,y=148,w=26,h=53)".

### Style Requirements

For EACH of the files you turn in you must have:

* A comment header for your class.
* All class variables and methods must have associated Javadoc comments.
* Tricky code and hard-coded values must be explained with in-line comments.
* Code is neatly indented and spaced.
* Give meaningful names to methods/variables, and follow Java's naming standards.
* Limit the lengths of all lines in your program to fewer than 100 characters.
* Remember, we will also deduct for magic numbers.
* Also remember that each class constant requires its own separate Javadoc statement.

### Development Recommendations

### Developing your code in stages and knowing how to test your solutions will be critical to your success.

### Start by writing empty "stub" versions of all the required methods so that the TileManager class will compile.

### Next, write your constructor, addTile first, then drawAll. You can then run the TileMainMini program to make sure that you can see the tiles appear on the screen.

### Add click-related methods one at a time and test each one individually to be sure it works before moving on to the next.

### One part of this program involves figuring out which rectangle tile(s), if any, touch a given x/y pixel. You can figure this out by comparing the x/y position of the click to the x/y area covered by the tile. For example, if a tile has a top-left corner of (x=20, y=10), a width of 50, and a height of 15, it touches all of the pixels from (20, 10) through (69, 24) inclusive. Such a tile contains the point (32, 17) because 32 is between 20 and 69 and 17 is between 10 and 24.

### If you have bugs or exceptions in your code, there are several things you can try.

### You can print out the state of your list and of each Tile object with temporary println statements. (Though this is a graphical program, println output does still appear on the console.) You must remove any such println statements before you turn in the assignment to receive full credit.

### You can also use a debugger as found in jGRASP or Eclipse to pause the code at a breakpoint in the middle of an operation. Stepping through line-by-line can help you to see what has gone wrong.

* + You can also modify TileMain to reduce the number of initial tiles shown on the screen.
* Once you have implemented TileManager such that it works with the full TileMain GUI, we suggest then you can implement the RectangleTile object.
* Once the RectangleTile is working appropriately, test it by modifying the TileManager to randomly generate these objects as well as the CircleTile objects.

### A sample solution is available on the course web site. You can use it to verify your program's behavior.

### Milestone 1

You must implement 3 methods of your TileManager for Milestone 1:

* The constructor
* addTile
* drawAll

You can test this by running TileMainMini.java: if your Milestone 1 is working correctly, TileMainMini will draw 20 randomly placed circles on the screen, and pressing the “N” key should add a new circle to the screen. We also recommend you write your Javadoc for the other methods in this file.

Upload your TileManager.java shell to the GradeIt link for Assignment 11 Design.

### Milestone 2

For Milestone 2 complete the TileManager.java and RectangleTile.java files meeting all internal and external requirements and turn it into the GradeIt link provided on the course website.

### Grading

You will be graded on the program’s "external correctness" (whether the program compiles and produces exactly the expected output), its "internal correctness" (whether your source code follows the stylistic guidelines in this document) as well as its style and documentation.

#### Milestone 1:

2 points – Implementing the first 3 methods of Tile Manager

#### Milestone 2:

8 points - external correctness   
7 points - internal design and efficiency  
3 points - style and documentation

### Optional Features

There are no optional features for this program

### Challenges

* Create other types of Tiles such as TriangleTile, HexagonalTile or other shapes