

Hogwarts Trace & Output - Carlos A. Guevara									
method()	int y	int x	int z	call	return	receive	x =	println order	Output
bludger(2001)	2001	int x = y / 1000 x = (2001 / 1000);	int z = (x + y); z = (2 + 2001);	quaffle(z, y); quaffle(2003, 2001);	n/a n/a	z = 1001	x = z x = 1001		"bludger: x = " + x + ", y = " + y + ", z = " + z "bludger: x = " + x + ", y = " + 2001 + ", z = " + 2003
bludger memory		2	2003					3	bludger: x = 1001, y = 2001, z = 2003
quaffle(x, y);	2001	x = 2003	int z = snitch(x + y, y); int z = snitch(2003 + 2001, 2001); int z = snitch(4004, 2001); int z = y int z = 1001 y /= z y = 2001/1001 y = 1	snitch(x + y, y);		z = 1001 y = 1001			"quaffle: x = " + x + ", y = " + y + ", z = " + z quaffle: x = 2003, y = 1, z = 1001
quaffle memory								2	
snitch(x + y, y);	2001	x = 4004 y = x / (x % 10) y = 4004 / (4004 % 10) y = 4004 / 4 y = 1001							"snitch: x = " + x + ", y = " + y "snitch: x = " + x + ", y = " + y
snitch memory						y = 1001		1	snitch: x = 4004, y = 1001
<div>Final Output</div> <div> bludger: x = 1001, y = 2001, z = 2003  quaffle: x = 2003, y = 1, z = 1001  snitch: x = 4004, y = 1001 </div>									

### 3. Random circles

(Code uploaded to remote repository: [proj\\_assigns/assignment3](#))

Write a GraphicsProgram that draws a set of ten circles with different sizes, positions, and colors. Each circle should have a randomly chosen color, a randomly chosen radius between 5 and 50 pixels, and a randomly chosen position on the canvas, subject to the condition that the entire circle must fit inside the canvas without extending past the edge.

The following sample run shows one possible outcome: RandomCircles

***On some runs of this program you might not see ten circles. Why?***

It may happen that the RandomGenerator will select WHITE as a color.

### 4. Drawing lines

(Code uploaded to remote repository: [proj\\_assigns/assignment3](#))

Write a GraphicsProgram that allows the user to draw lines on the canvas. Pressing the mouse button sets the starting point for the line. Dragging the mouse moves the other endpoint around as the drag proceeds. Releasing the mouse fixes the line in its current position and gets ready to start a new line. For example, suppose that you press the mouse button somewhere on the screen and then drag it rightward an inch, holding the button down. What you'd like to see is the following picture: DrawLines If you then move the mouse downward without releasing the button, the displayed line will track the mouse, so that you might see the following picture: DrawLines Because the original point and the mouse position appear to be joined by some elastic string, this technique is called rubber-banding. Although this program may seem quite powerful, it is also simple to implement. The entire program requires fewer than 20 lines of code.