**CSC 229 Test #3 Prep**

**(10). 1. Design a method normalizedArray that given a one-dimensional array a returns an array b where:**

**a[i] = b[i] – standard deviation (a) 0 <= i <= length of a**

**public** **int**[] normalizedArray(int[] array)

{

**int**[] b = **new** **int**[array.length];

**int** ave = 0;

**for** (**int** i=0; i<array.length; i++)

{

ave = ave + array[i];

}

ave = ave/array.length;

**double** sum = 0;

**for** (**int** i=0; i<array.length; i++)

{

sum = sum + (array[i]-ave)\*(array[i]-ave);

}

**int** std = (**int**)Math.*sqrt*(sum/array.length);

System.***out***.println("STD = "+std);

**for** (**int** i=0; i<array.length; i++)

{

b[i] = array[i] - std;

}

**return** b;

}

**2. Design a method arrayProduct that given two one-dimensional arrays a and b calculates and returns an integer number p where:**

**P = sum (a[i]\*b[i] 0 <= I <= length of a**

**NOTE: first make sure a and b have the same length**

**public int arrayProduct(int[] a, int[] b)**

**{**

**int c = 0;**

**if(a.length != b.length)**

**{**

**c =-1;**

**}**

**else**

**{**

**for(int i = 0; i < a.length; i++)**

**{**

**c= c + a[i] \* b[i];**

**}**

**}**

**return c;**

**}**

**(10) 3. Develop a method that given a 2D array of integers A as the input argument stores the 2D array in a 1D array row by row and returns the 1D array.**

|  |  |  |  |
| --- | --- | --- | --- |
| **1** | **2** | **3** | **4** |
| **5** | **6** | **7** | **8** |
| **9** | **10** | **11** | **12** |

|  |
| --- |
| **1** |
| **2** |
| **3** |
| **4** |
| **5** |
| **6** |
| **7** |
| **8** |
| **9** |
| **10** |
| **11** |
| **12** |

**public int[] toArray(int[][] array)**

**{**

**int[] a = new int[array.length \* array[0].length];**

**int index = 0;**

**for (int i=0; i<rows; i++)**

**for (int j=0; j<columns; j++)**

**{**

**a[index] = array[i][j];**

**index++;**

**}**

**return a;**

**}**

**(10) 4. Develop a method *average* that receives a 2D integer array A[N][N] and returns a 2D array B[N][N] of integers that is created using the following formulas:**

**B[i][j] = integer average of neighbors of A[i][j] (**cells immediately next

to A[i][j] including A[i][j] **)**

**B[i][j] = 0 for all border cells**

**A B**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **5** | **2** | **4** | **1** | **3** |
| **1** | **2** | **3** | **1** | **2** |
| **2** | **1** | **3** | **4** | **2** |
| **4** | **2** | **1** | **2** | **1** |
| **3** | **1** | **1** | **1** | **3** |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **0** | **0** | **0** | **0** | **0** |
| **0** | **2** | **2** | **2** | **0** |
| **0** | **2** | **2** | **2** | **0** |
| **0** | **2** | **1** | **2** | **0** |
| **0** | **0** | **0** | **0** | **0** |

**public** **int** [][] average(int[][] array)

{

**int**[][] c = **new** **int**[array.length][array[0].length];

**for** (**int** i=1; i<array.length-1; i++)

**for** (**int** j=1; j<array[rows].length-1; j++)

{

**c[i][j] = (array[i-1][j-1] + array[i-1][j] + array[i-1][j+1] +**

**array[i][j-1] + array[i][j] + array[i][j+1] +**

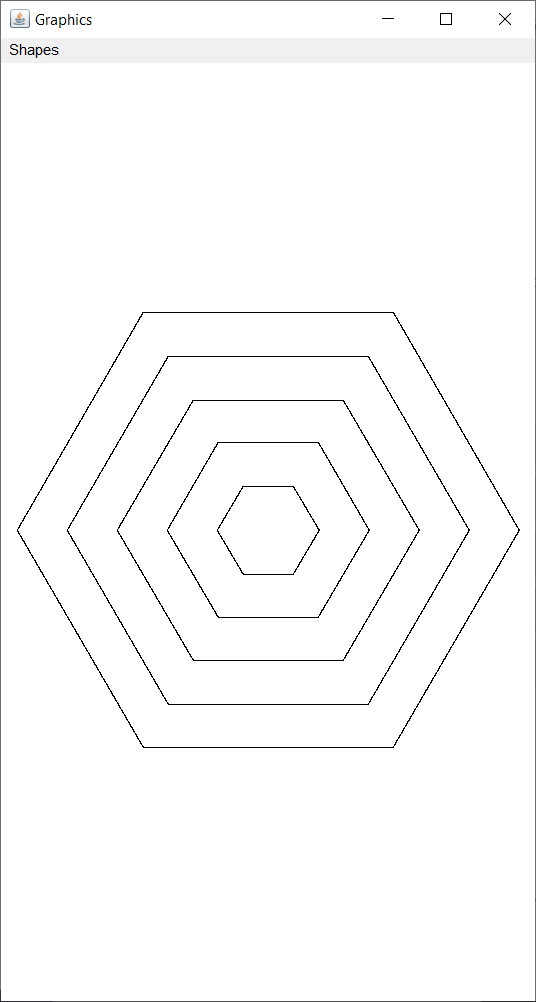
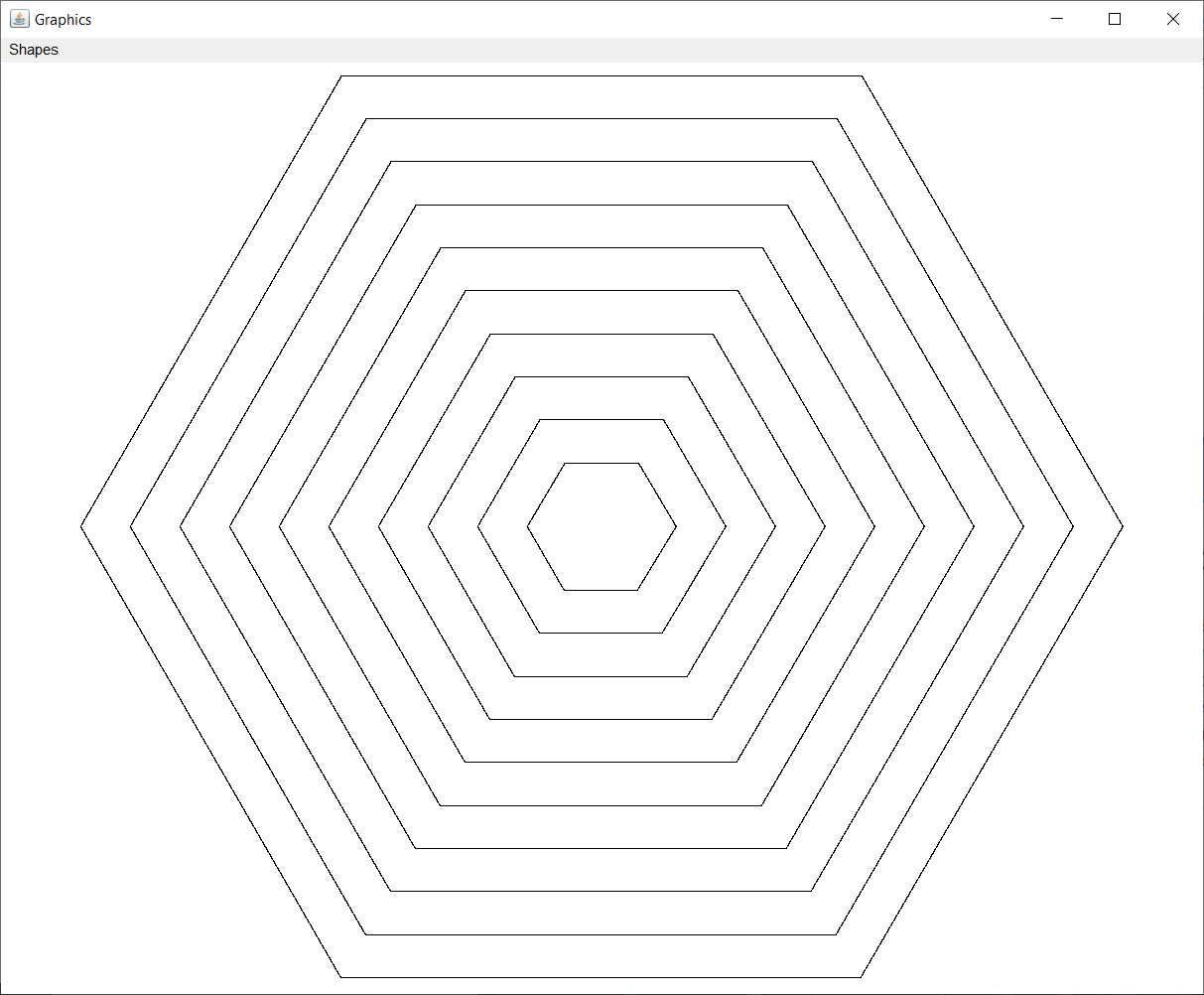
**array[i+1][j+1] + array[i+1][j] array[i+1][j+1])/9;**

}

**return** c;

}

**(10) 5. Develop a paint method that draws concentric regular polygons separated by 20 pixels based on current width and height of the visible screen as long as they fit on the screen.**



**case** "Problem 5":

{

**int** r = 0;

**int** x = ww/2;

**int** dh = (wh-40);

**int** y = 40+dh/2;

// ((wh-80)/2)/sin(60 degree)

**int** rv = (**int**)((wh-80)/(1.73205080757));

**int** rh = (ww-40)/2;

r = Math.*min*(rv, rh);

RegularPolygon p = **new** RegularPolygon(x,y,6,r);

**int**[] xv = **new** **int**[6];

**int**[] yv = **new** **int**[6];

**while** (r>=20)

{

p = **new** RegularPolygon(x,y,6,r);

xv = p.getX();

yv = p.getY();

g.drawPolygon(xv, yv, 6);

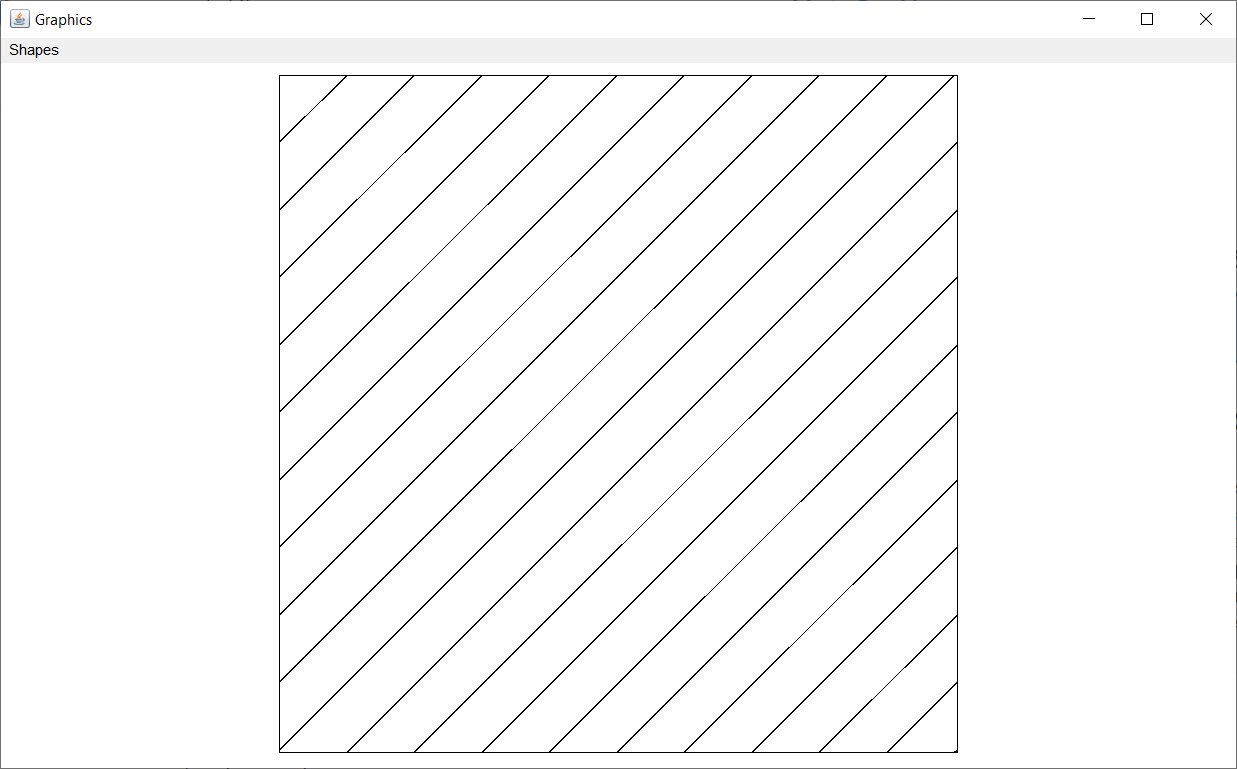
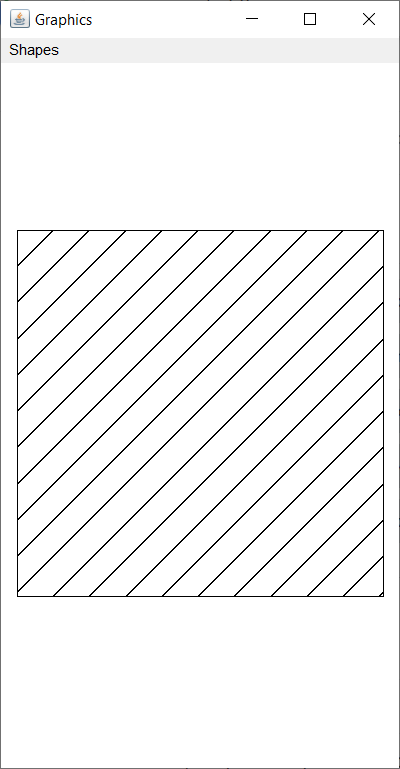
r = r-20;

}

**break**;

}

**(10) 6. Develop a paint method that draws the following shape based on current width and height of the visible screen. The square is the largest square that fits on the screen, lines are equally spaced.**



**case** "Problem 6":

{

**int** dh = wh-80;

**int** w = Math.*min*(dh, ww-40);

**int** xs = (ww-w)/2;

**int** ys = 60+(dh-w)/2;

g.drawRect(xs, ys, w, w);

**int** gap = w/10;

**for** (**int** i = 1; i <=10; i++)

{

g.drawLine(xs+i\*gap, ys, xs, ys+i\*gap);

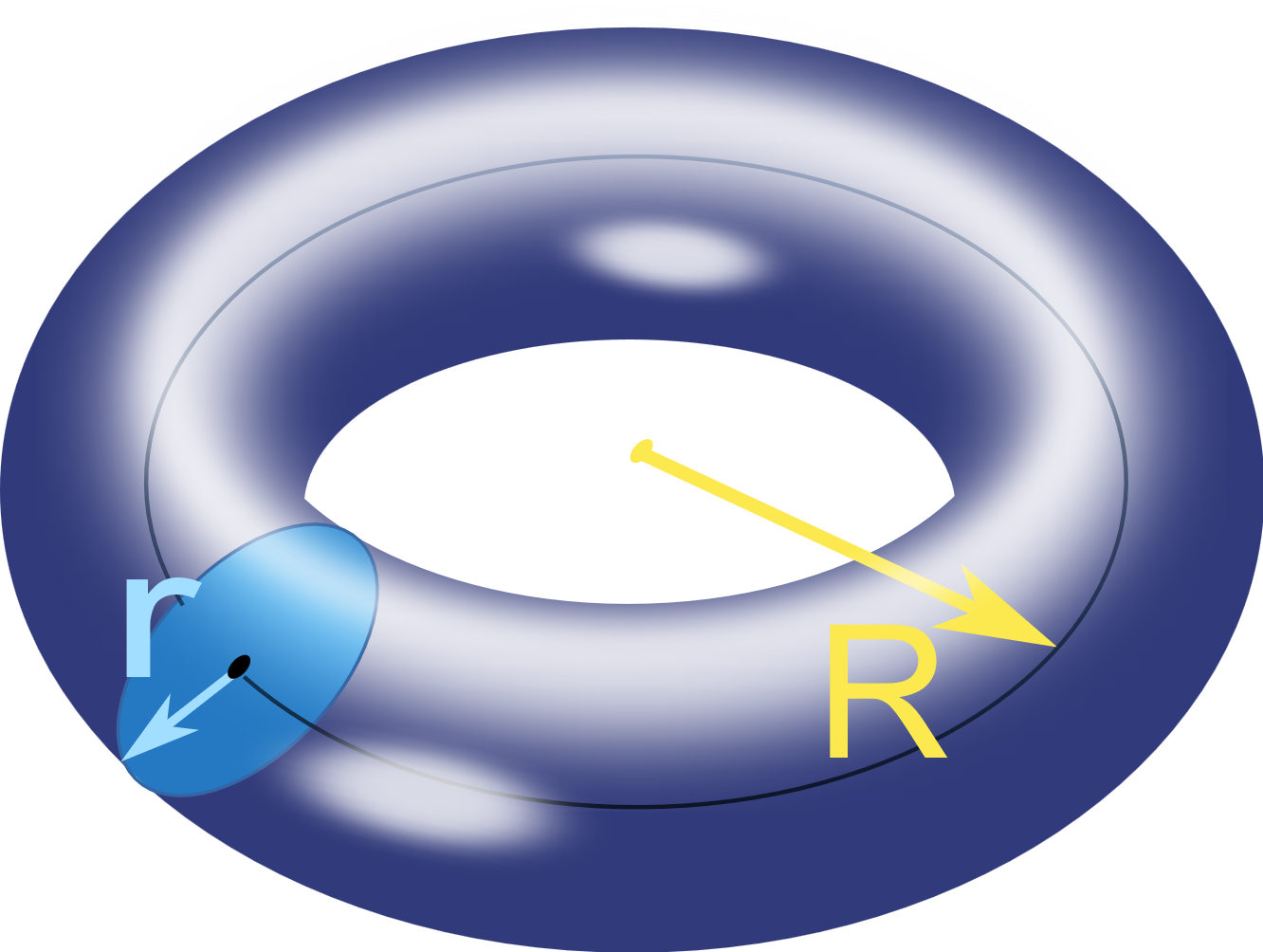
g.drawLine(xs+w, ys+i\*gap, xs+i\*gap, ys+w);

}

**break**;

}

**(10) 7. Design a class named Torus to represent a torus identified with the length of the larger circle and length of revolving edge as shown:**



**Surface Area = 4 × *π*2 × R × r**

**Volume = 2 × *π*2 × R × r2**

<https://www.mathsisfun.com/geometry/torus.html>

**Draw the UML diagram for the class. Implement the class.**