**0. (0 points) NAME\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ PERIOD \_\_\_\_\_\_\_**

* Write your name on the test. If you can't remember your name, it will cost you a point.
* This test is closed book. No computer, no friends, no Internet. Just you, your brain, your pen(cil), and the paper.
* Calculators are not permitted. Not even the really big calculators in the classroom.
* Where possible, **SHOW YOUR WORK!** (It is really hard to give partial credit if you don't!)
* Do not write programs that worry about checking for errors. You may assume that all arguments to a function will be in the function's domain.
* Sometimes, if you cannot remember something important, you might find it someplace else on the test.
* Don't panic. It's just a test.

**1. (5 points; each subproblem 1 point apiece)** What will the outputs be from the following pieces of code? (Hint: Show your work. It not only makes it easier for the teacher to understand what you are doing, but it might help you follow what you are doing, too!)

A. int[] favNum = {2, 3, 5, 7, 1, 6, 4};

System.out.println(favNum[favNum[favNum[4]]]);

**7**

B. final int ROW = 5;

final int COL = 5;

int[][] identity = new int[ROW][COL];

for(int i = 0; i < ROW; i++){

for(int j = 0; j < COL; j++){

if(i == j)

identity[i][j] = 1;

else

identity[i][j] = 0;

}

}

Write the contents of the entire array.

**{ { 1, 0, 0, 0, 0},**

**{ 0, 1, 0, 0, 0},**

**{ 0, 0, 1, 0, 0},**

**{ 0, 0, 0, 1, 0},**

**{ 0, 0, 0, 0, 1} }**

C. public int thing(int x) {  
      if (x == 0) {  
         return 1;  
      } else {  
         return x \* thing(x-1);  
      }  
   }  
   System.out.println(thing(4));

**24**

D. double[] fred = {6.0, 6.6, 2.2, 1.5};  
   fred[1] /= fred[2];  
   fred[0] \*= fred[3];  
   System.out.print(fred[0] / fred[1] / fred[3]);

**2.0**

E. ArrayList<Integer> what = new ArrayList<Integer>();  
   what.add(3);  
   what.add(7);  
   what.add(2);   
   for (int i = 0; i < 4; i++)  
   {  
      if (i % 2 == 0) {  
         what.add(0, i);  
      } else {  
         what.set(i, (what.get(i) \* 2));  
      }  
   }  
   what.remove(3);  
   System.out.println(what);

**[2, 0, 6, 2]**

**2. (5 points)**A basic operation involving matrices is *scalar multiplication* (multiplying all elements of a matrix by some number). Write a method, *sMult*, that takes a 2-D array (a matrix) of integers and an integer which will multiply the array.

If your program reads as follows:

public static void main(String[] args) {

int[][] matrix = { {3, 4, 5, 9}, {2, 0, 1, 10}, {6, 8, 7, 11} }; sMult(matrix, 2);

}

after your call to  sMult your array should contain:

6 8 10 18

4 0 2 20

12 16 14 22

Write sMult. Hint: use nested for loops (i.e., one for loop inside of another for loop).

**SOLUTION**

**public** **static** **void** sMult(**int**[][] a, **int** scalar) {

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

**for** (**int** j = 0; j < a[0].length; j++) {

a[i][j] \*= scalar;

}

}

}

**3. (5 points)**A regular array has a fixed length, so what do you do when you have filled your array to its max capacity? Write a method, *arrayReSize*, that takes a filled array of integers as an input and returns an array with the same contents, but twice as big with the second half of the elements initialized to 0.

(BTW, this is what happens behind the scenes of an ArrayList. When you initialize an ArrayList, it really makes an array with a bunch of helpful methods – one of which monitors how big your array is and automatically expands it when it gets full)

If your program reads as follows:

public static void main(String[] args) {

**int**[] array = {1, 2, 3, 4};

array = *arrayReSize*(array);

}

after your call to  arrayReSize your array should contain:

1 2 3 4 0 0 0 0

Write arrayReSize. Hint: .length will be extremely useful

**SOLUTION**

**public** **static** **int**[] arrayReSize(**int**[] array) {

// Note that everything is initialized to zero

// when a new array of primitive number data

// is created.

**int**[] newArray = **new** **int**[array.length \* 2];

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

newArray[i] = array[i];

}

**return** newArray;

}

**4. (5 points)**Write a method findMin that takes an ArrayList<Integer> and returns the smallest integer in the ArrayList. Note that this method does not alter the array that is passed.

Example:

ArrayList<Integer> a = new ArrayList<Integer>();

ArrayList<Integer> b = new ArrayList<Integer>();

a.add(2);

a.add(1);

a.add(5);

a.add(5);

a.add(7); // a now contains [2, 1, 5, 5, 7]

b.add(1);

b.add(2);

b.add(3);

b.add(-5);

b.add(4); // b now contains [1, 2, 3, -5, 4]

System.out.println(findMin(a)); // prints 1

System.out.println(findMin(b)); // prints -5

Write findMin by filling in the blanks. Here are some ArrayList methods:

 add(item): adds item to the end of an ArrayList

 set(position, item): changes the element at position of the ArrayList to item

 get(position): returns the element at position of the ArrayList

 remove(position): returns the element at position of the ArrayList and then deletes it from the ArrayList

 size(): returns the number of elements of the ArrayList, not the ArrayList's capacity

**SOLUTION**

**public** **static** **int** findMin(ArrayList<Integer> alist) {

**int** holder = alist.get(0); //holds the smallest # in array

**for** (**int** i = 1; i < alist.size(); i++) {

**if** (alist.get(i) < holder) {

holder = alist.get(i);

}

}

**return** holder;

}