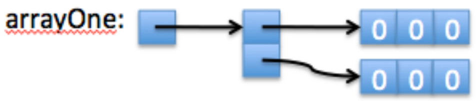
COMP132 – Computer Science II
Fall 2015

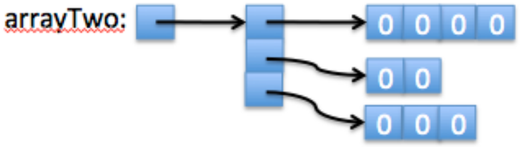
Homework #2
2D Arrays and Static Fields/Methods
Solutions

1. Draw a picture of the array that would be created by each of the following array declarations. Be sure to include the array variable and draw all references as arrows.

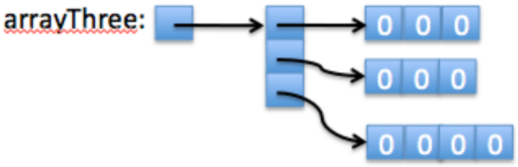
```
a. int[][] arrayOne = new int[2][3];
```



```
b. int[][] arrayTwo = new int[3][];
   arrayTwo[0] = new int[4];
   arrayTwo[1] = new int[2];
   arrayTwo[2] = new int[3];
```



```
c. int[][] arrayThree = new int[3][3];
   arrayThree[2] = new int[4];
```



2. Consider the following declaration of a ragged array:

```
int[][] myArray = new int[5][];
myArray[0] = new int[5];
   // omitted code that creates rows 1…3
myArray[4] = new int[3];
```

Given the above declaration, write java statements that will perform each of the following tasks. You should assume that rows 1…3 are sufficiently long such that all of the array indices needed are valid.

a. Print the length of the 2nd row of `myArray`. Recall that array indices begin with 0, so the 2nd row of the array is at index 1.

```
System.out.println(myArray[1].length);
```

b. Set the element in the 3rd row, 4th column of `myArray` to be 7

```
myArray[2][3] = 7;
```

c. Set the element in the last column of the last row of `myArray` to be twice the element in the first column of the first row.

```
myArray[4][2] = 2*myArray[0][0];
```

d. Set every element of `myArray` to be twice the sum of its row and column indices.

```
for (int row=0; row < myArray.length; row++) {
  for (int col=0; col<myArray[row].length; col++) {
    myArray[row][col] = 2 * (row + col);
  }
}
```

e. Swap the 2nd and 3rd rows of `myArray`. Solve this problem using the fact that a 2d array is really a 1d array of reference to the 1d arrays for the rows. Do not create any new arrays or copy the integers contained in any of the arrays. You may however find it useful to create temporary references to existing arrays.

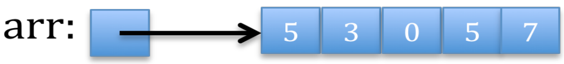
```
int[] tmp = myArray[1];
myArray[1] = myArray[2];
myArray[2] = tmp;
```

3. Give java statements that will create a triangular ragged array with 100 rows. Note: In a triangular array the first row has 1 column, the 2nd row has 2 columns and so on. Hint: Use a `for` loop!

```
public static void main(String[] args) {
   int[][] triangle = new int[100][];
      for (int i=0; i<100; i++) {
         triangle[i] = new int[i+1];
      }
}
```

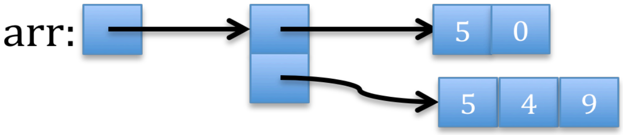
4. For each of the following snippets of code, draw a picture that shows the shape and contents of the array `arr` after the code is executed.

a.
```
int[] arr = new int[5];
arr[1] = 3;
arr[arr[1]] = 5;
arr[0] = arr[1+2];
arr[2] = arr[4];
arr[4] = 7;
```



b.
```
int[][] arr = new int[2][2];
    arr[0][0] = 5;
    arr[1][0] = 7;

    int[] arr2 = new int[3];
    arr2[0] = arr[0][0];
    arr[1] = arr2;
    arr2[2] = 9;
    arr[1][1] = 4;
```



5. This question refers to the `Candidate` class and the `Student` class from the `132SampleCode` project. Each question describes a new method to be added to the indicated class. For each method, indicate whether that method should be a `static` method or an instance method. Give a brief justification for your answer.

a. A method in the `Candidate` class that accepts a party name as a `String` parameter and determines if the `Candidate` is a member of that party.

This method must be an instance method because it needs access to the `Candidate`'s `party` field to check if it indicates the same party as the parameter.

b. A method in the `Candidate` class that accepts a party name as a `String` parameter and determines if that party name is an accepted party name as defined by one of the class constants.

This method can be a `static` method. It does not need to access any instance fields. It only needs access to its parameter and the class constants, which are also `static`.

c. A method in the `Student` class that changes the `nextID` field to the value specified by a parameter.

This method can be a `static` method. It does not need to access any instance fields. It only needs to access its parameter and set the value of the `nextID` field, which is a `static` field.

d. An accessor method in the `Student` class that returns the student's id number.

This method must be an instance method. It needs to access the `Student`'s `idNumber` field, which is an instance field.

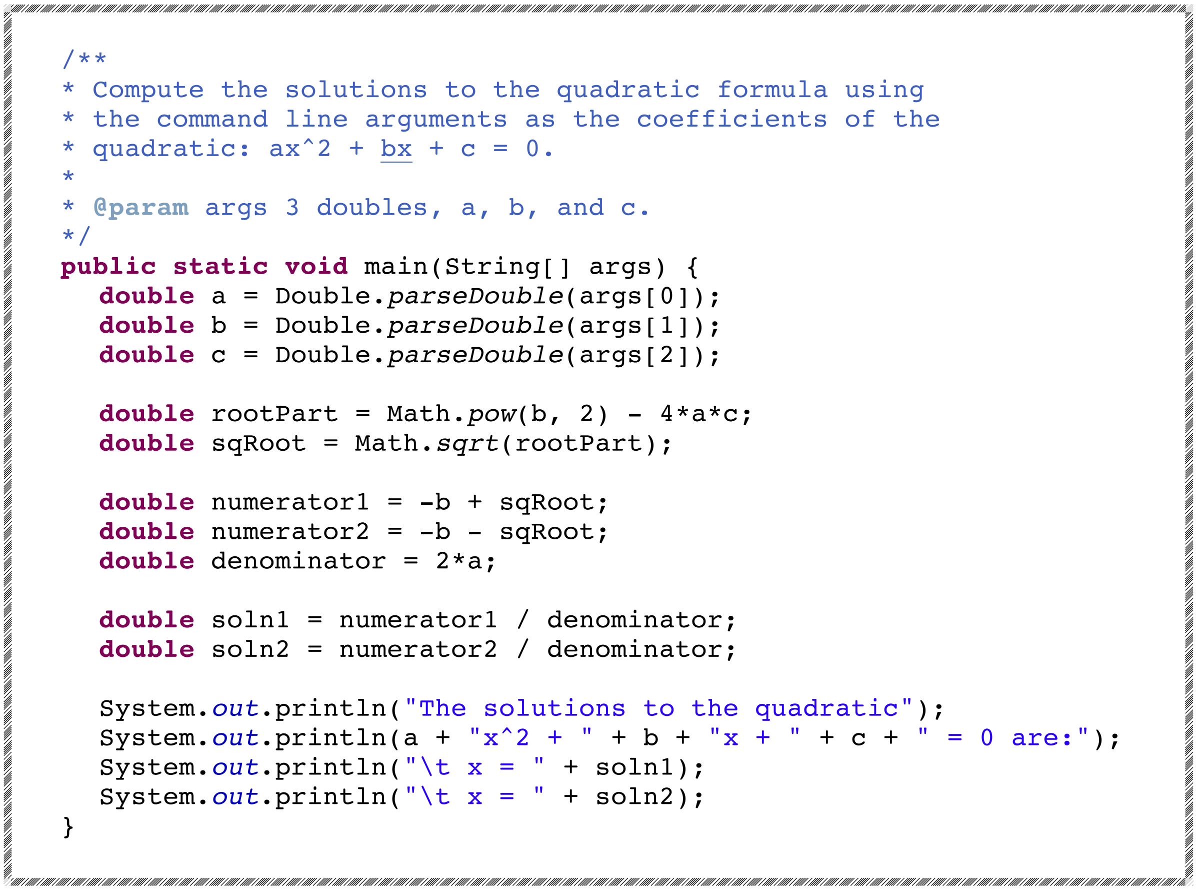
6. Write a `main` method that computes the solution(s) to a given quadratic equation. You can get a refresher on quadratic equations and their solution at:

http://www.purplemath.com/modules/quadform.htm

Your `main` method must accept three `double` values as command line parameters (see the `JavaStaticMethods` class in the `123SampleCode` project). These three values represent the coefficients (a, b, c) in the quadratic formula ($ax^2 + bx + c = 0$) to be solved. You may assume that the coefficients given yield at least one real valued solution. Your `main` method must have a JavaDoc comment, and must compute the solution(s) to the quadratic and print them to the screen.

Bonus: Modify your program so that it can also handle coefficients that result in imaginary/complex valued solutions. (+1)

NOTE: See the course How-To document to learn how to provide command line arguments to programs that are run within eclipse. This will allow you to run and test your solution from within eclipse.



```java
/**
* Compute the solutions to the quadratic formula using
* the command line arguments as the coefficients of the
* quadratic: ax^2 + bx + c = 0.
*
* @param args 3 doubles, a, b, and c.
*/
public static void main(String[] args) {
   double a = Double.parseDouble(args[0]);
   double b = Double.parseDouble(args[1]);
   double c = Double.parseDouble(args[2]);

   double rootPart = Math.pow(b, 2) - 4*a*c;
   double sqRoot = Math.sqrt(rootPart);

   double numerator1 = -b + sqRoot;
   double numerator2 = -b - sqRoot;
   double denominator = 2*a;

   double soln1 = numerator1 / denominator;
   double soln2 = numerator2 / denominator;

   System.out.println("The solutions to the quadratic");
   System.out.println(a + "x^2 + " + b + "x + " + c + " = 0 are:");
   System.out.println("\t x = " + soln1);
   System.out.println("\t x = " + soln2);
}
```

7. Consider the following class definition, which includes one static field and one instance field.

```
public class Foo {

    private static int x = 1;
    private int y;

    public Foo(int y) {
        this.y = y;
    }

    public void bar() {
        x++;
        y=x;
    }

    public void baz(int z) {
        x = x + z;
        y = y + z;
    }

    public String toString() {
        return "x= " + x + " y= " + y;
    }
}
```

Give the output that would be produced by each of the following code snippets.

a.
```
Foo f1 = new Foo(2);
Foo f2 = new Foo(3);

f1.bar();
System.out.println("f1:  " + f1);
System.out.println("f2:  " + f2);

f2.bar();
System.out.println("f1:  " + f1);
System.out.println("f2:  " + f2);
```

```
f1: x= 2 y= 2
f2: x= 2 y= 3
f1: x= 3 y= 2
f2: x= 3 y= 3
```

Explanation: `x` is a `static` field. This means that there is only one field `x` and it is associated with the class `Foo`. So all instances of `Foo` (e.g. `f1` and `f2`) share the field `x`. `y` is an instance field so each object of type `Foo` has its own field `y`. When `bar()` is called it increases the value of the (shared) `static` field `x`.

b.
```
Foo f1 = new Foo(2);
Foo f2 = new Foo(3);

f1.baz(3);
System.out.println("f1: " + f1);
System.out.println("f2: " + f2);

f2.baz(5);
System.out.println("f1: " + f1);
System.out.println("f2: " + f2);
```

```
f1: x= 4 y= 5
f2: x= 4 y= 3
f1: x= 9 y= 5
f2: x= 9 y= 8
```

Explanation: The key points here are the same as in part a. That is that `x` is a `static` field and thus there is only one `x`, and that `y` is an instance field and both `f1` and `f2` have their own independent copy of `y`. When `baz()` is invoked it modifies the value of the `static` (shared) field `x` and also the `y` field of the object on which it was invoked (i.e. f2).