Tutorial # 6 Arrays

SOFE 2710U Object Oriented Programming and Design

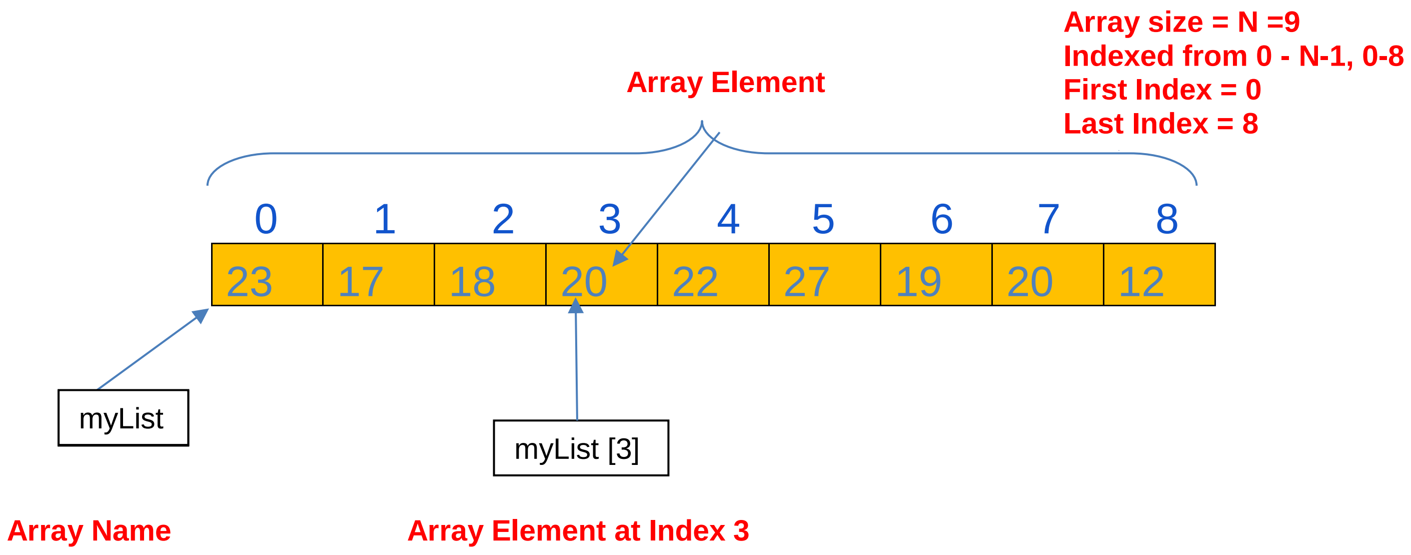
FALL 2020

Objective:

In this tutorial, you will learn the basic concepts of arrays. This tutorial covers one-dimensional arrays, arrays of objects, variable length parameter lists and Two-dimensional arrays.

Declaring and Using Arrays

Arrays are objects that can be used to organize huge amount of data and hence, it is an ordered list of values.



Declare myList array as Follows:

```
int [ ] myList= new int [9];
```

Arrays of Objects

Some of important facts about Array of objects are as follows:

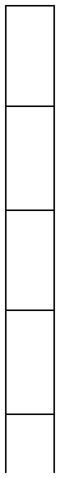
- The array elements can be object reference.
- An array of objects holds null references.
- Individual instantiation of objects stored within the array.

  Exampe:

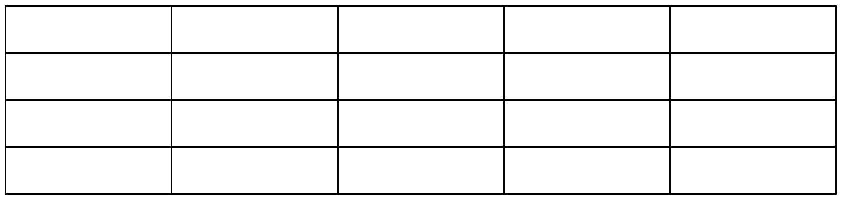
```
String [ ] cars = new String [ 7 ]
```

Two-Dimension Arrays

A two-dimension array can be considered a table of array elements with rows and columns or so-called arrays of arrays.



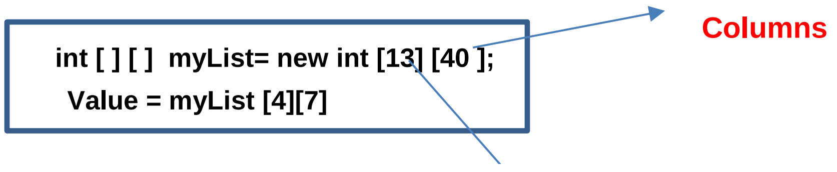
One-dimension Array



Two- dimension Array

Declare myList as Two-dimension array

The declaration of the two-dimension array can be done by specifying each size of each dimension individually and an array element is referenced using two index values as follows:



```
int [ ] [ ]  myList= new int [13] [40 ];
 Value = myList [4][7]
```

Problems:

1. Write a program that grades arithmetic quizzes as follows:

a) Ask the user how many questions are in the quiz.
b) Ask the user to enter the key (that is, the correct answers). There should be one answer for each question in the quiz, and each answer should be an integer. They can be entered on a single line, e.g., 34 7 13 100 81 3 9 10 321 12 might be the key for a 10-question quiz. You will need to store the key in an array.
c) Ask the user to enter the answers for the quiz to be graded. As for the key, these can be entered on a single line. Again, there needs to be one for each question. Note that these answers do not need to be stored; each answer can simply be compared to the key as it is entered.
d) When the user has entered all of the answers to be graded, print the number correct and the percent correct.

When this works, add a loop so that the user can grade any number of quizzes with a single key. After the results have been printed for each quiz, ask "Grade another quiz? (y/n)."

2. Write a program that prompts the user for an integer, then asks the user to enter that many values. Store these values in an array and print the array. Then reverse the array elements so that the first element becomes the last element, the second element becomes the second to last element, and so on, with the old last element now first. Do not just reverse the order in which they are printed; actually, change the way they are stored in the array. Do not create a second array; just rearrange the elements within the array you have. (Hint: Swap elements that need to change places.) When the elements have been reversed, print the array again.

3. In this exercise you will complete a class that implements a shopping cart as an array of items. The file *Item.java* contains the definition of a class named *Item* that models an item one would purchase. An item has a name, price, and quantity (the quantity purchased). The file *ShoppingCart.java* implements the shopping cart as an array of Item objects.

I. Complete the *ShoppingCart* class by doing the following:
   a. Declare an instance variable *cart* to be an array of Items and instantiate *cart* in the constructor to be an array holding *capacity* Items.
   b. Fill in the code for the *increaseSize* method. Your code should be similar to that in Listing 7.8 of the text but instead of doubling the size just increase it by 3 elements.
   c. Fill in the code for the *addToCart* method. This method should add the item to the cart and update the *totalPrice* instance variable (note this variable takes into account the quantity).
   d. Compile your class.

II. Write a program that simulates shopping. The program should have a loop that continues as long as the user wants to shop. Each time through the loop read in the name, price, and quantity of the item the user wants to add to the cart. After adding an item to the cart, the cart contents should be printed. After the loop print a "Please pay ..." message with the total price of the items in the cart.

```
// ***************************************************************
//   Item.java
//
//   Represents an item in a shopping cart.
// ***************************************************************

import java.text.NumberFormat;

public class Item
{
    private String name;
    private double price;
    private int quantity;

    //  --------------------------------------------------- --
    //  Create a new item with the given attributes.
    //  --------------------------------------------------- --
    public Item (String itemName, double itemPrice, int numPurchased)
    {
      name = itemName;
      price = itemPrice;
      quantity = numPurchased;
     }

     //  --------------------------------------------------- --
     //    Return a string with the information about the item
     //  --------------------------------------------------- --
     public String toString ()
     {
      NumberFormat fmt = NumberFormat.getCurrencyInstance();

      return (name + "\t" + fmt.format(price) + "\t" + quantity + "\t"
            + fmt.format(price*quantity));
     }

     //  ---------------------------------------------
     //    Returns the unit price of the item
     //  ---------------------------------------------

      public double getPrice()
     {
      return price;
     }

     //  ---------------------------------------------
//         Returns the name of the item
//       ---------------------------------------------
     public String getName()
     {
      return name;
     }

     //  ---------------------------------------------
     //    Returns the quantity of the item
     //  ---------------------------------------------
    public int getQuantity()
     {
      return quantity; }
     }
}


// *************************************************************
//  ShoppingCart.java
//
//  Represents a shopping cart as an array of items
// *************************************************************

import java.text.NumberFormat;

public class ShoppingCart
{
    private int itemCount;       // total number of items in the cart
    private double totalPrice;   // total price of items in the cart
    private int capacity;        // current cart capacity


     //  ---------------------------------------------------
//       Creates an empty shopping cart with a capacity of 5 items.
//       ---------------------------------------------------
     public ShoppingCart()
     {
      capacity = 5;
      itemCount = 0;
      totalPrice = 0.0;
     }
     // ----------------------------------------------------
     // Adds an item to the shopping cart.
     // ----------------------------------------------------
    public void addToCart(String itemName, double price, int quantity)
     {
     }

     // ----------------------------------------------------
     // Returns the contents of the cart together with
     // summary information.


     // ----------------------------------------------------
     public String toString()
     {
      NumberFormat fmt = NumberFormat.getCurrencyInstance();

      String contents = "\nShopping Cart\n";
      contents += "\nItem\t\tUnit Price\tQuantity\tTotal\n";

      for (int i = 0; i < itemCount; i++) contents
          += cart[i].toString() + "\n";

      contents += "\nTotal Price: " + fmt.format(totalPrice);
      contents += "\n";

      return contents;
     }

     // ----------------------------------------------------
//       Increases the capacity of the shopping cart by 3
//      ----------------------------------------------------
    private void increaseSize()
     {
     }
}
```

4. The file *Parameters.java* contains a program to test the variable length method *average* from Section 7.5 of the text. Note that *average* must be a static method since it is called from the static method *main*.

a) Compile and run the program. You must use the -source 1.5 option in your compile command.

b) Add a call to find the average of a single integer, say 13. Print the result of the call.

c) Add a call with an empty parameter list and print the result. Is the behavior what you expected?

d) Add an interactive part to the program. Ask the user to enter a sequence of at most 20 nonnegative integers. Your program should have a loop that reads the integers into an array and stops when a negative is entered (the negative number should not be stored). Invoke the average method to find the average of the integers in the array (send the array as the parameter). Does this work?

e) Add a method *minimum* that takes a variable number of integer parameters and returns the minimum of the parameters. Invoke your method on each of the parameter lists used for the average function.

```
//*******************************************************
//  Parameters.java
//
//  Illustrates the concept of a variable parameter list.
//*******************************************************

import java.util.Scanner;

public class Parameters

{
    //----------------------------------------------
//       Calls the average and minimum methods with
//       different numbers of parameters.
    //----------------------------------------------
    public static void main(String[] args)
    {
        double mean1, mean2;

        mean1 = average(42, 69, 37);
        mean2 = average(35, 43, 93, 23, 40, 21, 75);

        System.out.println ("mean1 = " + mean1);
        System.out.println ("mean2 = " + mean2);
    }

    //----------------------------------------------
    //  Returns the average of its parameters.
    //----------------------------------------------
    public static double average (int ... list)
    {
        double result = 0.0;

        if (list.length != 0)
            {
                int sum = 0;
                for (int num: list)
                    sum += num;
                result = (double)sum / list.length;
            }

        return result;
    }
}
```

One interesting application of two-dimensional arrays is *magic squares.* A magic square is a square matrix in which the sum of every row, every column, and both diagonals is the same. Magic squares have been studied for many years, and there are some particularly famous magic squares. In this exercise you will write code to determine whether a square is magic.

File *Square.java* contains the shell for a class that represents a square matrix. It contains headers for a constructor that gives the size of the square and methods to read values into the square, print the square, find the sum of a given row, find the sum of a given column, find the sum of the main (or other) diagonal, and determine whether the square is magic. The read method is given for you; you will need to write the others. Note that the read method takes a Scanner object as a parameter.

File *SquareTest.java* contains the shell for a program that reads input for squares from a file named *magicData* and tells whether each is a magic square. Following the comments, fill in the remaining code. Note that the main method reads the size of a square, then after constructing the square of that size, it calls the *readSquare* method to read the square in. The readSquare method must be sent the Scanner object as a parameter.

5. You should find that the first, second, and third squares in the input are magic, and that the rest (fourth through seventh) are not. Note that the -1 at the bottom tells the test program to stop reading.

```
// ************************************************************
// Square.java
//
// Define a Square class with methods to create and read in
// info for a square matrix and to compute the sum of a row,
// a col, either diagonal, and whether it is magic.
//
// ************************************************************

import java.util.Scanner;

public class Square
{
    int[][] square;
```

```
//--------------------------------------
//create new square of given size
//--------------------------------------
public Square(int size)
{

}

//--------------------------------------
//return the sum of the values in the given
row //--------------------------------------
public int sumRow(int row)
{

}

//--------------------------------------
//return the sum of the values in the given
column //--------------------------------------
public int sumCol(int col)
{

}

//-----------------------------------


//return the sum of the values in the main
diagonal //--------------------------------------
public int sumMainDiag()
{

}

//--------------------------------------
//return the sum of the values in the other ("reverse")
diagonal //--------------------------------------
public int sumOtherDiag()
{

}

//--------------------------------------
//return true if the square is magic (all rows, cols, and diags have
//same sum), false otherwise
//--------------------------------------
public boolean magic()
{

}

//--------------------------------------
//read info into the square from the input stream associated with the
//Scanner parameter
//--------------------------------------
public void readSquare(Scanner scan)
{
  for (int row = 0; row < square.length; row++)
    for (int col = 0; col < square.length; col ++)
      square[row][col] = scan.nextInt();
}
```

```
    //----------------------------------------
    //print the contents of the square, neatly
    formatted //----------------------------------------
    public void printSquare()

    {

    }

}


// ***************************************************************
// SquareTest.java
//
// Uses the Square class to read in square data and tell if
// each square is magic.
//
// ***************************************************************

import java.util.Scanner;

public class SquareTest
{
    public static void main(String[] args) throws
    IOException {
      Scanner scan = new Scanner(new File("magicData"));

      int count = 1;                  //count which square we're on
      int size = scan.nextInt();      //size of next square

      //Expecting -1 at bottom of input file
      while (size != -1)
          {
            //create a new Square of the given size

            //call its read method to read the values of the square

            System.out.println("\n******** Square " + count + "
            ********"); //print the square

            //print the sums of its rows

            //print the sums of its columns

            //print the sum of the main diagonal

            //print the sum of the other diagonal

            //determine and print whether it is a magic square
            //get size of next square
            size = scan.nextInt();
          }

    }
}
```

Magic Data

```
3
8    1    6
3    5    7
4    9    2
7
30     39        48        1      10       19      28
38     47         7        9      18       27      29
46      6         8       17      26       35      37
5      14        16       25      34       36      45
13     15        24       33      42       44       4
21     23        32       41      43        3      12
22     31        40       49       2       11      20
4
48      9         6        39
27     18        21        36
15     30        33        24
12     45        42         3
3
6    2    7
1    5    3
2    9    4
4
3    16          2      13
6      9         7      12
10     5        11       8
15     4        14       1
5
17     24        15        8       1
23      5        16       14       7
4       6        22       13      20
10     12         3       21      19
11     18         9        2      25
7
30     39        48        1      10       28      19
38     47         7        9      18       29      27
46      6         8       17      26       37      35
5      14        16       25      34       45      36
13     15        24       33      42        4      44
21     23        32       41      43       12       3
22     31        40       49      2        20      11
-1
```