Republic of the Philippines
City of Olongapo
GORDON COLLEGE
Olongapo City Sports Complex, Donor St., East Tapinac, Olongapo City
www.gordoncollege.edu.ph

Computer Programming 2

Module 4: Arrays, Strings and Console Input

Name (LN,FN,MN): _________________________________ Program/Yr/Block: ___________

I. Introduction

In this module, we will consider a fundamental construct known as the array. An array stores a sequence of values that are all of the same type. We want not just to store values but also to be able to quickly access each individual value. The method that we use to refer to individual values in an array is to number and then index them—if we have n values, we think of them as being numbered from 0 to n−1.

We will also cover Strings in this module which are widely used in computer programming. Strings in Java are objects rather than primitive types. Moreover, we will also cover the use of the Scanner class which is simple text scanner which can parse primitive types and strings using regular expressions (advanced concept, you'll learn more about regular expressions later).

II. Learning Objectives

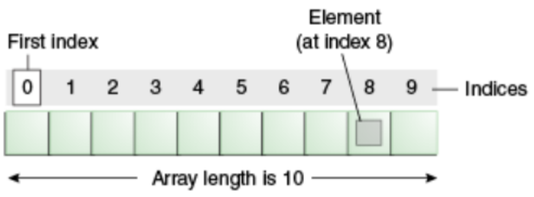
After completing this module, you should be able design and write programs that use the following concepts:

1. Arrays (one-dimensional/two-dimensional arrays)
2. Strings (String methods)
3. Console input using the Scanner class

III. Topics and Key Concepts

A. Arrays

An array is a container object that holds a fixed number of values of a single type. The length of an array is established when the array is created. After creation, its length is **fixed.**



An array of 10 elements.

Each item in an array is called an element, and each element is accessed by its numerical index. As shown in the preceding illustration, numbering begins with 0. The 9th element, for example, would therefore be accessed at index 8.

The following program, ArrayDemo, creates an array of integers, puts some values in the array, and prints each value to standard output.

```
class ArrayDemo {
    public static void main(String[] args) {
        // declares an array of integers
        int[] anArray;

        // allocates memory for 10 integers
        anArray = new int[10];

        // initialize first element
        anArray[0] = 100;
        // initialize second element
        anArray[1] = 200;
        // and so forth
        anArray[2] = 300;
        anArray[3] = 400;
        anArray[4] = 500;
        anArray[5] = 600;
        anArray[6] = 700;
        anArray[7] = 800;
        anArray[8] = 900;
        anArray[9] = 1000;

        System.out.println("Element at index 0: "
                           + anArray[0]);
        System.out.println("Element at index 1: "
                           + anArray[1]);
        System.out.println("Element at index 2: "
                           + anArray[2]);
        System.out.println("Element at index 3: "
                           + anArray[3]);
        System.out.println("Element at index 4: "
                           + anArray[4]);
        System.out.println("Element at index 5: "
                           + anArray[5]);
        System.out.println("Element at index 6: "
                           + anArray[6]);
        System.out.println("Element at index 7: "
                           + anArray[7]);
        System.out.println("Element at index 8: "
                           + anArray[8]);
        System.out.println("Element at index 9: "
                           + anArray[9]);
    }
}
```

The output from this program is:

```
Element at index 0: 100
Element at index 1: 200
Element at index 2: 300
Element at index 3: 400
Element at index 4: 500
Element at index 5: 600
Element at index 6: 700
Element at index 7: 800
Element at index 8: 900
Element at index 9: 1000
```

In a real-world programming situation, you would probably use one of the supported looping constructs to iterate through each element of the array, rather than write each line individually as in the preceding example. However, the example clearly illustrates the array syntax.

A.1 Declaring a Variable to Refer to an Array

The preceding program declares an array (named anArray) with the following line of code:

```
// declares an array of integers
int[] anArray;
```

Like declarations for variables of other types, an array declaration has two components:

1. the array's type
2. array's name

An array's type is written as type[], where type is the data type of the contained elements; the brackets are special symbols indicating that this variable holds an array. The size of the array is not part of its type (which is why the brackets are empty).

An array's name can be anything you want, provided that it follows the rules and conventions. As with variables of other types, the declaration does not actually create an array; it simply tells the compiler that this variable will hold an array of the specified type.

Similarly, you can declare arrays of other types:

```
byte[] anArrayOfBytes;
short[] anArrayOfShorts;
long[] anArrayOfLongs;
float[] anArrayOfFloats;
double[] anArrayOfDoubles;
boolean[] anArrayOfBooleans;
char[] anArrayOfChars;
String[] anArrayOfStrings;
```

You can also place the brackets after the array's name:

```
// this form is discouraged
float anArrayOfFloats[];
```

However, convention discourages this form; the brackets identify the array type and should appear with the type designation.

A.2 Creating, Initializing, and Accessing an Array

One way to create an array is with the new operator. The next statement in the ArrayDemo program allocates an array with enough memory for 10 integer elements and assigns the array to the anArray variable.

```
// create an array of integers
anArray = new int[10];
```

If this statement is missing, then the compiler prints an error like the following, and compilation fails:

```
ArrayDemo.java:4: Variable anArray may not have been initialized.
```

The next few lines assign values to each element of the array:

```
anArray[0] = 100; // initialize first element
anArray[1] = 200; // initialize second element
anArray[2] = 300; // and so forth
```

Each array element is accessed by its numerical index:

```
System.out.println("Element 1 at index 0: " + anArray[0]);
System.out.println("Element 2 at index 1: " + anArray[1]);
System.out.println("Element 3 at index 2: " + anArray[2]);
```

Alternatively, you can use the shortcut syntax to create and initialize an array:

```
int[] anArray = {
    100, 200, 300,
    400, 500, 600,
    700, 800, 900, 1000
};
```

Here the length of the array is determined by the number of values provided between braces and separated by commas.

For economy in code, we often take advantage of Java's default array initialization convention as shown in the example code below:

```
double[] a = new double[n];
```

The default initial value is 0 for all numeric primitive types and false for type boolean.

A.3 Programming with Arrays

We need consider a number of important characteristics of programming with arrays:

- **Data Structure.** An array is the first data structure that you will use in programming. A data structure is an arrangement of data that enables efficient processing by a program.

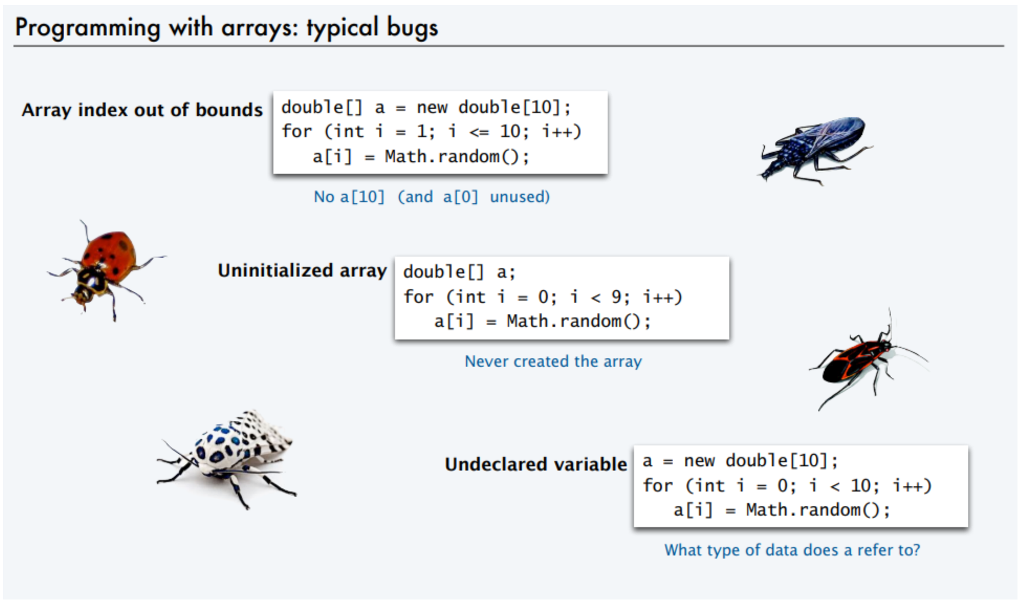
- **Zero-based indexing.** We always refer to the first element of an array a[] as a[0], the second as a[1], and so forth. It might seem more natural to you to refer to the first element as a[1], the second value as a[2], and so forth, but starting the indexing with 0 has some advantages and has emerged as the convention used in most modern programming languages.

- **Array length.** Once we create an array, its length is fixed. You can refer to the length of an anArray[] in your program with the code anArray.length.

```
System.out.println(anArray.length);
```

- **Memory representation.** When you use new to create an array, Java reserves space in memory for it (and initializes the values). This process is called memory allocation.

- **Bounds checking.** When programming with arrays, you must be careful. It is your responsibility to use legal indices when accessing an array element.



Programming with arrays: typical bugs

Array index out of bounds

```
double[] a = new double[10];
for (int i = 1; i <= 10; i++)
   a[i] = Math.random();
```

No a[10] (and a[0] unused)

Uninitialized array

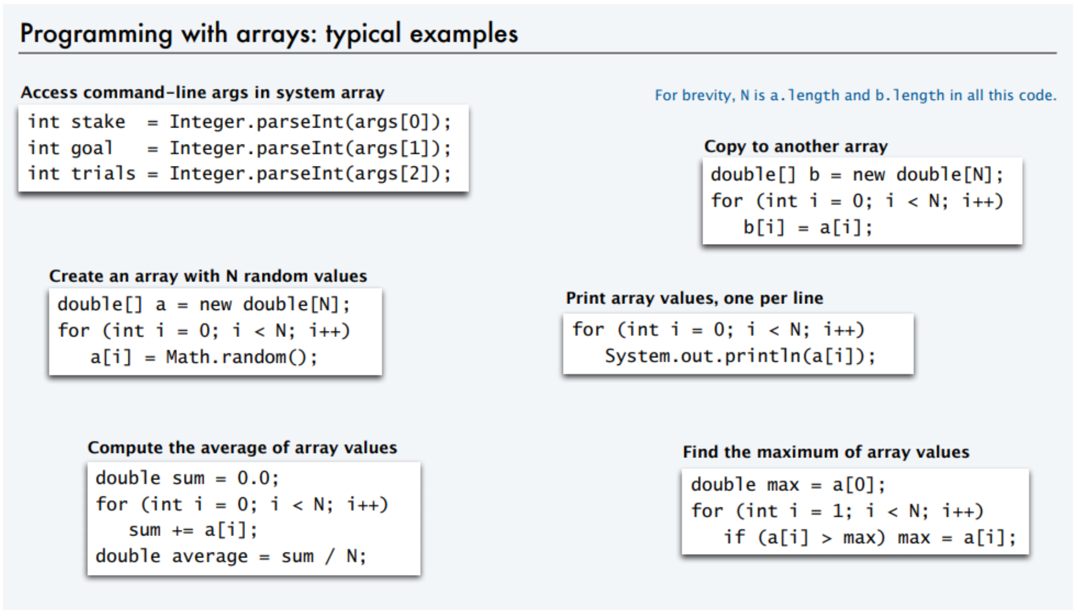
```
double[] a;
for (int i = 0; i < 9; i++)
   a[i] = Math.random();
```

Never created the array

Undeclared variable

```
a = new double[10];
for (int i = 0; i < 10; i++)
   a[i] = Math.random();
```

What type of data does a refer to?



Programming with arrays: typical examples

For brevity, N is a.length and b.length in all this code.

Access command-line args in system array

```
int stake  = Integer.parseInt(args[0]);
int goal   = Integer.parseInt(args[1]);
int trials = Integer.parseInt(args[2]);
```

Copy to another array

```
double[] b = new double[N];
for (int i = 0; i < N; i++)
   b[i] = a[i];
```

Create an array with N random values

```
double[] a = new double[N];
for (int i = 0; i < N; i++)
   a[i] = Math.random();
```

Print array values, one per line

```
for (int i = 0; i < N; i++)
   System.out.println(a[i]);
```

Compute the average of array values

```
double sum = 0.0;
for (int i = 0; i < N; i++)
   sum += a[i];
double average = sum / N;
```

Find the maximum of array values

```
double max = a[0];
for (int i = 1; i < N; i++)
   if (a[i] > max) max = a[i];
```

The Deck.java code below contains the full code for creating and shuffling a deck of cards.

```
public class Deck {
    public static void main(String[] args) {
        String[] SUITS = {
            "Clubs", "Diamonds", "Hearts", "Spades"
        };

        String[] RANKS = {
            "2", "3", "4", "5", "6", "7", "8", "9", "10",
            "Jack", "Queen", "King", "Ace"
        };

        // initialize deck
        int n = SUITS.length * RANKS.length;
        String[] deck = new String[n];
        for (int i = 0; i < RANKS.length; i++) {
            for (int j = 0; j < SUITS.length; j++) {
                deck[SUITS.length*i + j] = RANKS[i] + " of " + SUITS[j];
            }
        }

        // shuffle
        for (int i = 0; i < n; i++) {
            int r = i + (int) (Math.random() * (n-i));
            String temp = deck[r];
            deck[r] = deck[i];
            deck[i] = temp;
        }

        // print shuffled deck
        for (int i = 0; i < n; i++) {
            System.out.println(deck[i]);
        }
    }

}
```

Sample run:

```
/******************************************************************************
 *  Compilation:  javac Deck.java
 *  Execution:    java Deck
 *
 *  Deal 52 cards uniformly at random.
 *
 *  % java Deck
 *  Ace of Clubs
 *  8 of Diamonds
 *  5 of Diamonds
 *  ...
 *  8 of Hearts
 *
 ******************************************************************************/
```

The Sample.java program takes two command-line arguments m and n and produces a random sample of m of the integers from 0 to n-1.

```java
public class Sample {
    public static void main(String[] args) {
        int m = Integer.parseInt(args[0]);    // choose this many elements
        int n = Integer.parseInt(args[1]);    // from 0, 1, ..., n-1

        // create permutation 0, 1, ..., n-1
        int[] perm = new int[n];
        for (int i = 0; i < n; i++)
            perm[i] = i;

        // create random sample in perm[0], perm[1], ..., perm[m-1]
        for (int i = 0; i < m; i++)  {

            // random integer between i and n-1
            int r = i + (int) (Math.random() * (n-i));

            // swap elements at indices i and r
            int t = perm[r];
            perm[r] = perm[i];
            perm[i] = t;
        }

        // print results
        for (int i = 0; i < m; i++)
            System.out.print(perm[i] + " ");
        System.out.println();
    }
}
```

Sample run:

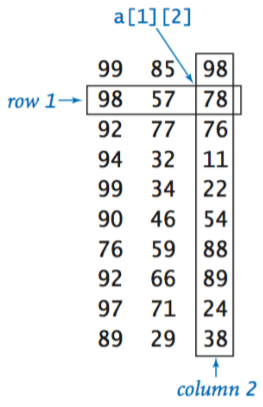
```
/******************************************************************************
 *  Compilation:  javac Sample.java
 *  Execution:    java Sample m n
 *
 *  This program takes two command-line arguments m and n and produces
 *  a random sample of m of the integers from 0 to n-1.
 *
 *  % java Sample 6 49
 *  10 20 0 46 40 6
 *
 *  % java Sample 10 1000
 *  656 488 298 534 811 97 813 156 424 109
 *
 ******************************************************************************/
```

A.4 Multi-dimensional Arrays

You can also declare an array of arrays (also known as a multidimensional array) by using two or more sets of brackets, such as `String[][]` names. Each element, therefore, must be accessed by a corresponding number of index values.

In the Java programming language, a multidimensional array is an array whose components are themselves arrays. This is unlike arrays in C or Fortran. A consequence of this is that the rows are allowed to vary in length (ragged arrays), as shown in the following MultiDimArrayDemo program:

```
class MultiDimArrayDemo {
    public static void main(String[] args) {
        String[][] names = {
            {"Mr. ", "Mrs. ", "Ms. "},
            {"Smith", "Jones"}
        };
        // Mr. Smith
        System.out.println(names[0][0] + names[1][0]);
        // Ms. Jones
        System.out.println(names[0][2] + names[1][1]);
    }
}
```



The output from this program is:

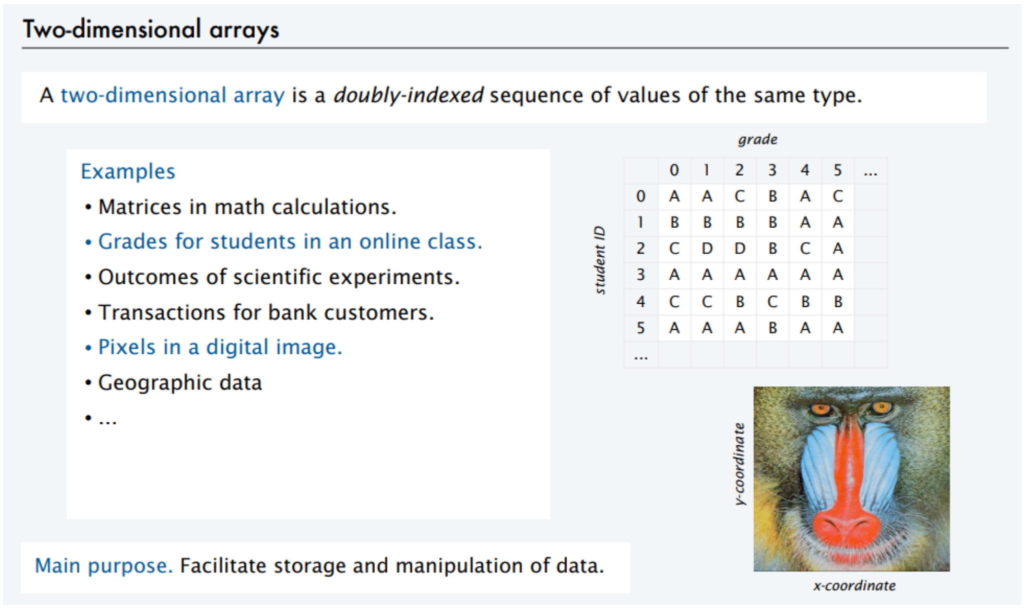
```
Mr. Smith
Ms. Jones
```

In many applications, a natural way to organize information is to use a table of numbers organized in a rectangle and to refer to rows and columns in the table. The mathematical abstraction corresponding to such tables is a matrix; the corresponding Java construct is a **two-dimensional array.**

To refer to the element in row i and column j of a two-dimensional array a[][], we use the notation a[i][j]; to declare a two-dimensional array, we add another pair of brackets; to create the array, we specify the number of rows followed by the number of columns after the type name (both within brackets), as follows:

```
double[][] a = new double[m][n];
```

We refer to such an array as an m-by-n array. By convention, the first dimension is the number of rows and the second dimension is the number of columns.



Two-dimensional arrays

A two-dimensional array is a *doubly-indexed* sequence of values of the same type.

Examples
- Matrices in math calculations.
- Grades for students in an online class.
- Outcomes of scientific experiments.
- Transactions for bank customers.
- Pixels in a digital image.
- Geographic data
- ...

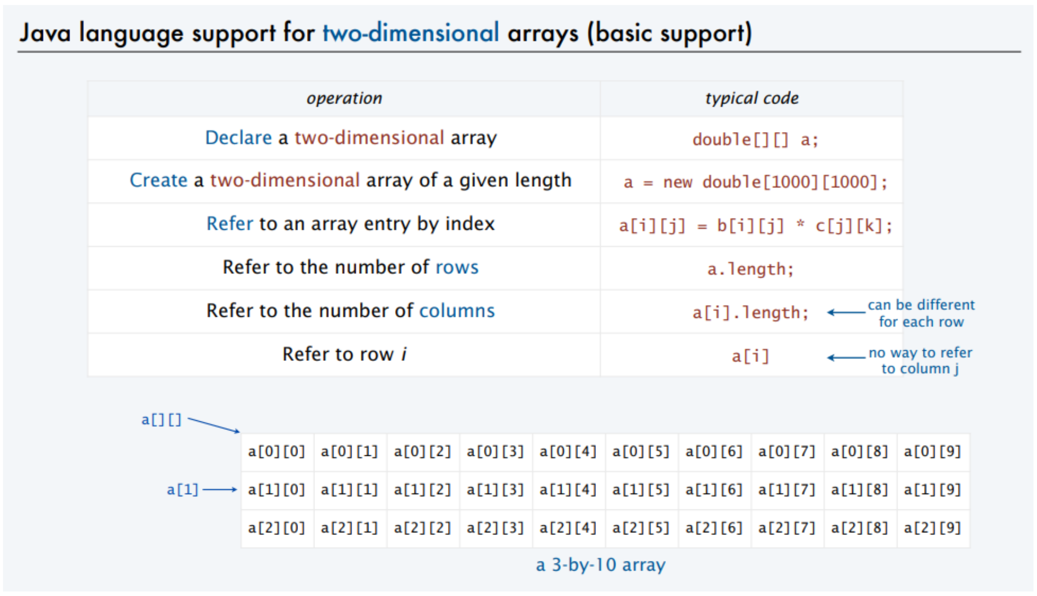
grade

student ID	0	1	2	3	4	5	...
0	A	A	C	B	A	C	
1	B	B	B	B	A	A	
2	C	D	D	B	C	A	
3	A	A	A	A	A	A	
4	C	C	B	C	B	B	
5	A	A	A	B	A	A	
...							

Main purpose. Facilitate storage and manipulation of data.

As with one-dimensional arrays, Java initializes all entries in arrays of numbers to 0 and in arrays of Booleans to false. Default initialization of two-dimensional arrays is useful because it masks more code than for one-dimensional arrays. To access each of the elements in a two-dimensional array, we need nested loops:

```
double[][] a;
a = new double[m][n];
for (int i = 0; i < m; i++)
   for (int j = 0; j < n; j++)
      a[i][j] = 0;
```

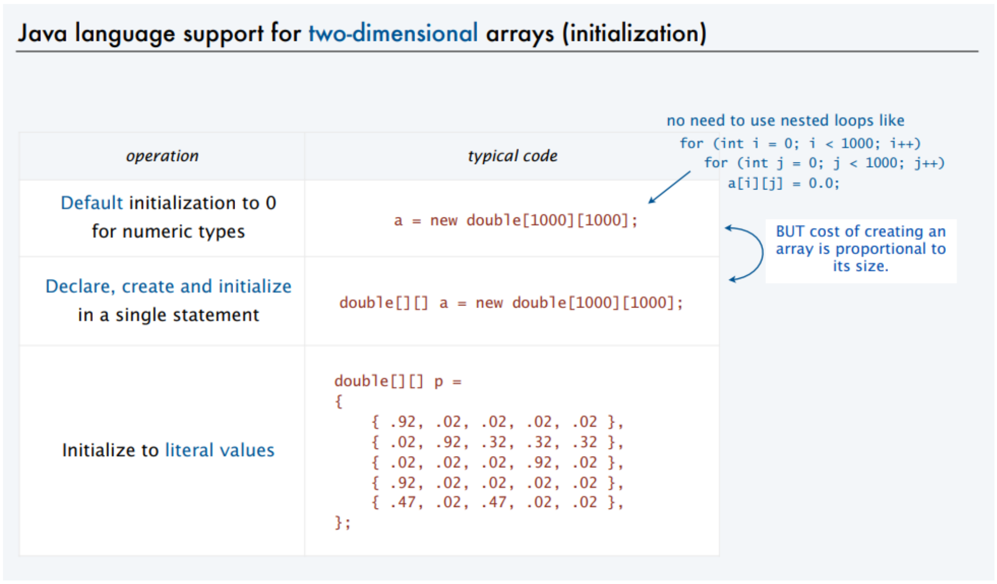


Java language support for two-dimensional arrays (basic support)

operation	typical code	
Declare a two-dimensional array	`double[][] a;`	
Create a two-dimensional array of a given length	`a = new double[1000][1000];`	
Refer to an array entry by index	`a[i][j] = b[i][j] * c[j][k];`	
Refer to the number of rows	`a.length;`	
Refer to the number of columns	`a[i].length;`	can be different for each row
Refer to row *i*	`a[i]`	no way to refer to column j

a[0][0]	a[0][1]	a[0][2]	a[0][3]	a[0][4]	a[0][5]	a[0][6]	a[0][7]	a[0][8]	a[0][9]
a[1][0]	a[1][1]	a[1][2]	a[1][3]	a[1][4]	a[1][5]	a[1][6]	a[1][7]	a[1][8]	a[1][9]
a[2][0]	a[2][1]	a[2][2]	a[2][3]	a[2][4]	a[2][5]	a[2][6]	a[2][7]	a[2][8]	a[2][9]

a 3-by-10 array



Java language support for two-dimensional arrays (initialization)

operation	*typical code*
Default initialization to 0 for numeric types	`a = new double[1000][1000];`
Declare, create and initialize in a single statement	`double[][] a = new double[1000][1000];`
Initialize to literal values	`double[][] p = { { .92, .02, .02, .02, .02 }, { .02, .92, .32, .32, .32 }, { .02, .02, .02, .92, .02 }, { .92, .02, .02, .02, .02 }, { .47, .02, .47, .02, .02 }, };`

no need to use nested loops like

```
for (int i = 0; i < 1000; i++)
   for (int j = 0; j < 1000; j++)
      a[i][j] = 0.0;
```

BUT cost of creating an array is proportional to its size.

The same notation extends to arrays that have any number of dimensions. For instance, we can declare and initialize a three-dimensional array with the code:

```
double[][][] a = new double[n][n][n];
```

B. Strings

Strings, which are widely used in Java programming, are a sequence of characters. In the Java programming language, strings are objects.

The Java platform provides the String class to create and manipulate strings.

B.1 Creating Strings

The most direct way to create a string is to write:

```
String greeting = "Hello world!";
```

In this case, "Hello world!" is a string literal—a series of characters in your code that is enclosed in double quotes. Whenever it encounters a string literal in your code, the compiler creates a String object with its value—in this case, Hello world!.

As with any other object, you can create String objects by using the new keyword and a constructor. The String class has thirteen constructors that allow you to provide the initial value of the string using different sources, such as an array of characters:

```
char[] helloArray = { 'h', 'e', 'l', 'l', 'o', '.' };
String helloString = new String(helloArray);
System.out.println(helloString);
```

The last line of this code snippet displays hello.

The String class is immutable, so that once it is created a String object cannot be changed. The String class has a number of methods that appear to modify strings. Since strings are immutable, what these methods really do is create and return a new string that contains the result of the operation.

B.2 String Length

Methods used to obtain information about an object are known as accessor methods. One accessor method that you can use with strings is the length() method, which returns the number of characters contained in the string object. After the following two lines of code have been executed, len equals 17:

```
String palindrome = "Dot saw I was Tod";
int len = palindrome.length();
```

A palindrome is a word or sentence that is symmetric—it is spelled the same forward and backward, ignoring case and punctuation. Here is a short and inefficient program to reverse a palindrome string. It invokes the String method charAt(i), which returns the ith character in the string, counting from 0.

```
public class StringDemo {
    public static void main(String[] args) {
        String palindrome = "Dot saw I was Tod";
        int len = palindrome.length();
        char[] tempCharArray = new char[len];
        char[] charArray = new char[len];

        // put original string in an
        // array of chars
        for (int i = 0; i < len; i++) {
            tempCharArray[i] =
                palindrome.charAt(i);
        }

        // reverse array of chars
        for (int j = 0; j < len; j++) {
            charArray[j] =
                tempCharArray[len - 1 - j];
        }

        String reversePalindrome =
            new String(charArray);
        System.out.println(reversePalindrome);
    }
}
```

Running the program produces this output:

```
doT saw I was toD
```

To accomplish the string reversal, the program had to convert the string to an array of characters (first for loop), reverse the array into a second array (second for loop), and then convert back to a string.

The String class includes a method, getChars(), to convert a string, or a portion of a string, into an array of characters so we could replace the first for loop in the program above with:

```
palindrome.getChars(0, len, tempCharArray, 0);
```

B.3 Concatenating Strings

The String class includes a method for concatenating two strings:

```
string1.concat(string2);
```

This returns a new string that is string1 with string2 added to it at the end.

You can also use the concat() method with string literals, as in:

```
"My name is ".concat("Rumplestiltskin");
```

Strings are more commonly concatenated with the + operator, as in

```
"Hello," + " world" + "!"
```

which results in

```
"Hello, world!"
```

The + operator is widely used in print statements. For example:

```
String string1 = "saw I was ";
System.out.println("Dot " + string1 + "Tod");
```

which prints

```
Dot saw I was Tod
```

Such a concatenation can be a mixture of any objects. For each object that is not a String, its toString() method is called to convert it to a String.

The Java programming language does not permit literal strings to span lines in source files, so you must use the + concatenation operator at the end of each line in a multi-line string. For example:

```
String quote =
    "Now is the time for all good " +
    "men to come to the aid of their country.";
```

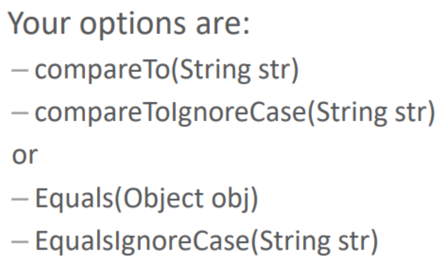
Breaking strings between lines using the + concatenation operator is, once again, very common in print statements.

B.4 Comparing Strings

You should never compare the value of Strings by using the == method (equals equals) that you use with primitive data types.

This is because the == operator only compares the contents of stack memory therefore only compares the String references not the String values (heap memory).

When comparing Strings there are different methods that you can use. The two ways of doing this are the compareTo() method and the equals() method.

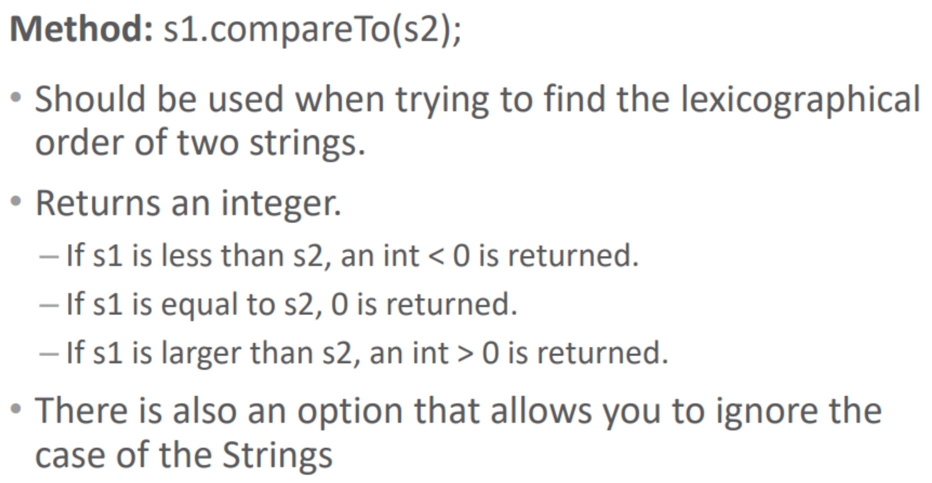


Your options are:

- compareTo(String str)
- compareToIgnoreCase(String str)

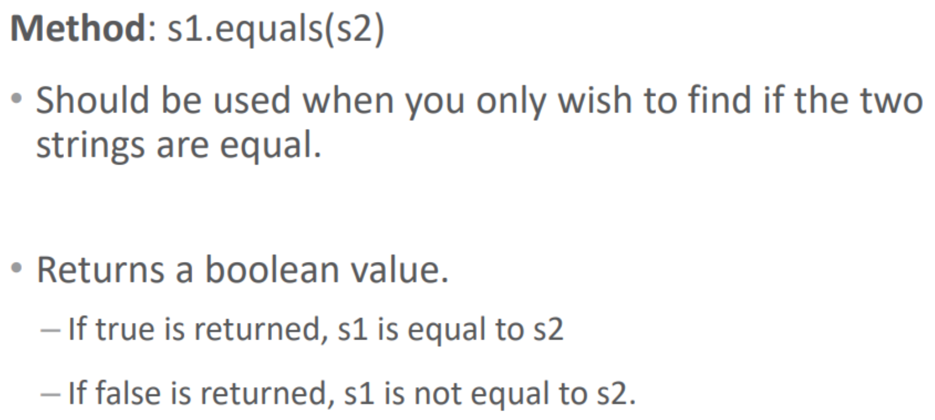
or

- Equals(Object obj)
- EqualsIgnoreCase(String str)



Method: s1.compareTo(s2);

- Should be used when trying to find the lexicographical order of two strings.
- Returns an integer.
  - If s1 is less than s2, an int < 0 is returned.
  - If s1 is equal to s2, 0 is returned.
  - If s1 is larger than s2, an int > 0 is returned.
- There is also an option that allows you to ignore the case of the Strings



Method: s1.equals(s2)

- Should be used when you only wish to find if the two strings are equal.
- Returns a boolean value.
  - If true is returned, s1 is equal to s2
  - If false is returned, s1 is not equal to s2.

B.5 Other String Methods

`public class String`

	`String(String s)`	*create a string with the same value as* `s`
	`String(char[] a)`	*create a string that represents the same sequence of characters as in* `a[]`
`int`	`length()`	*number of characters*
`char`	`charAt(int i)`	*the character at index* `i`
`String`	`substring(int i, int j)`	*characters at indices* `i` *through* (`j-1`)
`boolean`	`contains(String substring)`	*does this string contain* `substring`*?*
`boolean`	`startsWith(String prefix)`	*does this string start with* `prefix`*?*
`boolean`	`endsWith(String postfix)`	*does this string end with* `postfix`*?*
`int`	`indexOf(String pattern)`	*index of first occurrence of* `pattern`
`int`	`indexOf(String pattern, int i)`	*index of first occurrence of* `pattern` *after* `i`
`String`	`concat(String t)`	*this string, with* `t` *appended*
`int`	`compareTo(String t)`	*string comparison*
`String`	`toLowerCase()`	*this string, with lowercase letters*
`String`	`toUpperCase()`	*this string, with uppercase letters*
`String`	`replace(String a, String b)`	*this string, with* `a`*s replaced by* `b`*s*
`String`	`trim()`	*this string, with leading and trailing whitespace removed*
`boolean`	`matches(String regexp)`	*is this string matched by the regular expression?*
`String[]`	`split(String delimiter)`	*strings between occurrences of* `delimiter`
`boolean`	`equals(Object t)`	*is this string's value the same as* `t`*'s?*
`int`	`hashCode()`	*an integer hash code*

To see code examples, go to this link.

B.6 Special Characters

Because strings must be written within quotes, Java will misunderstand this string, and generate an error:

```java
String txt = "We are the so-called "Vikings" from the north.";
```

The solution to avoid this problem, is to use the backslash escape character.

The backslash (\) escape character turns special characters into string characters:

Escape character	Result	Description
\'	'	Single quote
\"	"	Double quote
\\	\	Backslash

The sequence \" inserts a double quote in a string:

```java
String txt = "We are the so-called \"Vikings\" from the north.";
```

C. The Scanner Class

The Scanner class is a simple text scanner which can parse primitive types and strings using regular expressions.

To read input that the user has entered to the console, use the Java class Scanner. You will have to use an import statement to access the class java.util.Scanner.

```
import java.util.Scanner;
```

To initialize a Scanner, write:

```
Scanner in = new Scanner(System.in);
```

InputVariables.java uses the Scanner class to accept input from the console.

```
import java.util.Scanner;
public class InputVariables {
        public static void main(String[] args) {
                Scanner in = new Scanner(System.in);

                boolean boolVal;
                byte byteVal;
                char charVal;
                short shortVal;
                int intVal;
                long longVal;
                float floatVal;
                double doubleVal;

                System.out.print("Please enter a boolean value: ");
                boolVal = in.nextBoolean();

                System.out.print("Please enter a byte value: ");
                byteVal = in.nextByte();

                System.out.print("Please enter a char value: ");
                charVal = in.next().charAt(0);

                System.out.print("Please enter a short value: ");
                shortVal = in.nextShort();

                System.out.print("Please enter an int value: ");
                intVal = in.nextInt();

                System.out.print("Please enter a long value: ");
                longVal = in.nextLong();

                System.out.print("Please enter a float value: ");
                floatVal = in.nextFloat();

                System.out.print("Please enter a double value: ");
                doubleVal = in.nextDouble();

                in.close();

                System.out.println("boolean value: " + boolVal);
                System.out.println("byte value   : " + byteVal);
                System.out.println("char value   : " + charVal);
                System.out.println("short value  : " + shortVal);
                System.out.println("int value    : " + intVal);
                System.out.println("long value   : " + longVal);
                System.out.print("double value : " + floatVal);
                System.out.print("double value : " + doubleVal);
        }//end method main

}//end class InputVariables
```

IV. Teaching and Learning Materials Resources

- PC Computer | Laptop | Android Phone
- Gordon College LAMP
- Google Meet
- Facebook Messenger

V. Learning Tasks

A. Explore (60 points)

You are tasked to create/edit, compile and run the following sample codes that were used in this module:

1. ArrayDemo.java

Your code (provide screenshot)

Sample Run: your terminal window (provide screenshot)

2. Deck.java

Your code (provide screenshot)

Sample Run: your terminal window (provide screenshot)

3. Sample.java

Your code (provide screenshot)

Sample Run: your terminal window (provide screenshot)

4. MultiDimArrayDemo.java

Your code (provide screenshot)

Sample Run: your terminal window (provide screenshot)

5. StringDemo.java

Your code (provide screenshot)

Sample Run: your terminal window (provide screenshot)

6. InputVariables.java

Your code (provide screenshot)

Sample Run: your terminal window (provide screenshot)

B. Explain (40 points)

1. Consider the following string:

```
String hannah = "Did Hannah see bees? Hannah did.";
```

Question: What is the value displayed by the expression hannah.length()?

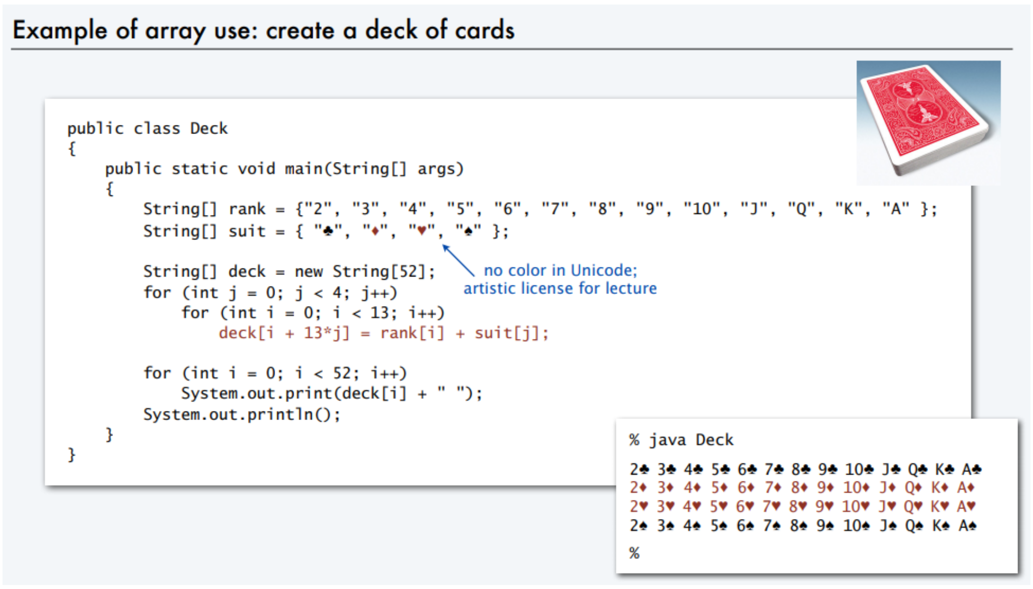
Question: What is the value returned by the method call hannah.charAt(12)?

2. How long is the string returned by the following expression? What is the string?

```
"Was it a car or a cat I saw?".substring(9, 12)
```

3. What does the following code print?

```
public class PQarray1
{
    public static void main(String[] args)
    {
        int[] a = new int[6];
        int[] b = new int[a.length];

        b = a;
        for (int i = 1; i < b.length; i++)
            b[i] = i;

        for (int i = 0; i < a.length; i++)
            System.out.print(a[i] + " ");
        System.out.println();

        for (int i = 0; i < b.length; i++)
            System.out.print(b[i] + " ");
        System.out.println();
    }
}
```



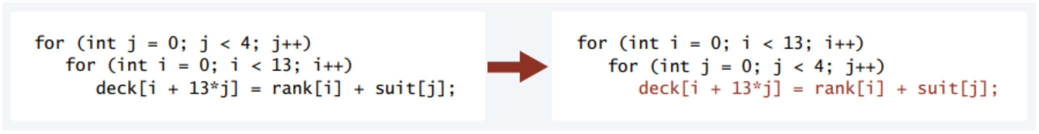
Example of array use: create a deck of cards

```java
public class Deck
{
    public static void main(String[] args)
    {
        String[] rank = {"2", "3", "4", "5", "6", "7", "8", "9", "10", "J", "Q", "K", "A" };
        String[] suit = { "♣", "♦", "♥", "♠" };

        String[] deck = new String[52];
        for (int j = 0; j < 4; j++)
            for (int i = 0; i < 13; i++)
                deck[i + 13*j] = rank[i] + suit[j];

        for (int i = 0; i < 52; i++)
            System.out.print(deck[i] + " ");
        System.out.println();
    }
}
```

no color in Unicode;
artistic license for lecture

```
% java Deck
2♣ 3♣ 4♣ 5♣ 6♣ 7♣ 8♣ 9♣ 10♣ J♣ Q♣ K♣ A♣
2♦ 3♦ 4♦ 5♦ 6♦ 7♦ 8♦ 9♦ 10♦ J♦ Q♦ K♦ A♦
2♥ 3♥ 4♥ 5♥ 6♥ 7♥ 8♥ 9♥ 10♥ J♥ Q♥ K♥ A♥
2♠ 3♠ 4♠ 5♠ 6♠ 7♠ 8♠ 9♠ 10♠ J♠ Q♠ K♠ A♠
%
```

4. What happens if the order of the for loops in Deck.java is switched?



```java
for (int j = 0; j < 4; j++)
    for (int i = 0; i < 13; i++)
        deck[i + 13*j] = rank[i] + suit[j];
```

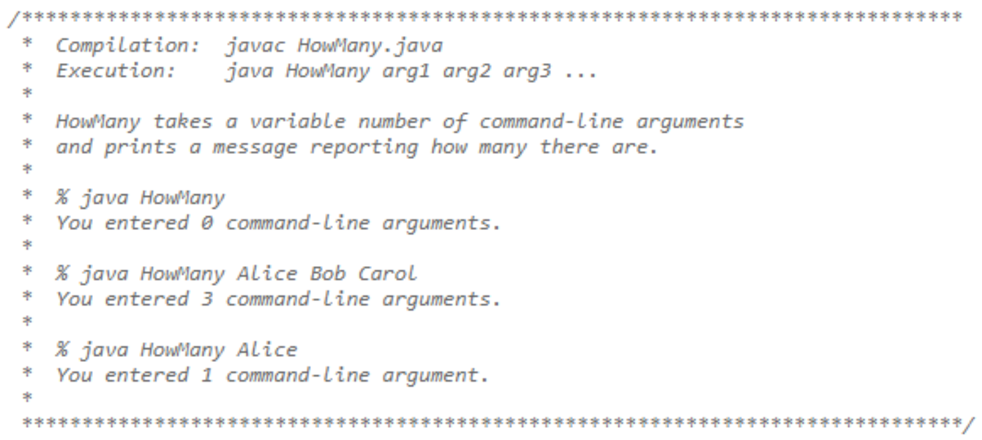
```java
for (int i = 0; i < 13; i++)
    for (int j = 0; j < 4; j++)
        deck[i + 13*j] = rank[i] + suit[j];
```

C. Engage (50 points)

1. Perform the following steps:
    a. Create a class called UniqueNums.
    b. Create a 5-element integer array called numbers.
    c. Create the following 4 variables:
        - integer value called num, initialized to 0 that stores the users input.
        - Integer value called numValues, initialized to 0 that store the number of valid entries.
        - Boolean value called valid, initialized to true that flags whether the number is unique or not.
        - Scanner object called in.
    d. Use a while loop to ask the user for values that will fill the array.

e. Use a nested do while loop to ask the user for a valid unique value.
f. Use a nested for loop to check that the entered value does not already exist in the numbers array.
g. If a valid number has been entered add it to the numbers array. Use the numValues variable for the array index.
h. Increment the numValues variable.
i. When the array has been filled with unique values close the Scanner object.
j. Use an enhanced for loop to display the contents of the numbers array to the console.

2. Write a program StrRev.java that reverses the order of values in a one-dimensional string array. (Additional points if will not create another array to hold the result)
3. Write a program HowMany.java that takes a variable or any number of command-line arguments and prints how many there are.
   Sample run:



```
/******************************************************************************
 *  Compilation:  javac HowMany.java
 *  Execution:    java HowMany arg1 arg2 arg3 ...
 *
 *  HowMany takes a variable number of command-line arguments
 *  and prints a message reporting how many there are.
 *
 *  % java HowMany
 *  You entered 0 command-line arguments.
 *
 *  % java HowMany Alice Bob Carol
 *  You entered 3 command-line arguments.
 *
 *  % java HowMany Alice
 *  You entered 1 command-line argument.
 *
 ******************************************************************************/
```

VI. References

- Oracle. nd. "Oracle Java Documentation". https://docs.oracle.com/javase/tutorial/
- Oracle Academy. 2020. "Java Programming Instructor Resources". https://academy.oracle.com
- Sedgewick Robert, Princeton University, Wayne, Kevin. 2017. "Introduction to Programming in Java: An Interdisciplinary Approach, 2nd Edition". https://introcs.cs.princeton.edu/java/home/