Classes and Objects 

1. Overview

The learning objectives of this laboratory session are:
- Understand and properly use classes constructors
- Understand and properly use arrays

2. Classes

2.1. Constructors

When you create a new instance (a new object) of a class using the **new** keyword, a constructor for that class is called. Constructors are used to initialize the instance variables (fields) of an object. Constructors are similar to methods, but with some important differences.
Constructor name is class name. A constructors must have the same name as the class its in.
Default constructor. If you don't define a constructor for a class, a default parameterless constructor is automatically created by the compiler. The default constructor calls the default parent constructor (super()) and initializes all instance variables to default value (zero for numeric types, null for object references, and false for booleans).
Default constructor is created only if there are no constructors. If you define any constructor for your class, no default constructor is automatically created.
Differences between methods and constructors:
- There is no return type given in a constructor signature (header). The value is this object itself so there is no need to indicate a return value.
- There is no return statement in the body of the constructor.
- The first line of a constructor must either be a call on another constructor in the same class (using this), or a call on the superclass constructor (using super). If the first line is neither of these, the compiler automatically inserts a call to the parameterless super class constructor.

These differences in syntax between a constructor and method are sometimes hard to see when looking at the source. It would have been better to have had a keyword to clearly mark constructors as some languages do.
this(…) - Calls another constructor in same class. Often a constructor with few parameters will call a constructor with more parameters, giving default values for the missing parameters. Use this to call other constructors in the same class.
super(…). Use super to call a constructor in a parent class. Calling the constructor for the superclass must be the first statement in the body of a constructor. If you are satisfied with the default constructor in the superclass, there is no need to make a call to it because it will be supplied automatically. Given here for reference, as we have not dealt with inheritance so far.

Example of explicit this constructor call:

```
public class Point {
    int m_x;
    int m_y;

    //============ Constructor
    public Point(int x, int y) {
        m_x = x;
        m_y = y;
    }
```

```
    //============ Parameterless default constructor
    public Point() {
        this(0, 0);  // Calls other constructor.
    }
    . . .
}
```

3. Arrays

An array can store many similar values in memory. Each value can be accessed by specifying a subscript or index. "Array" in Java means approximately the same thing as array, matrix, or vector does in math.

Unlike math, you must declare the array and allocate a fixed amount of memory for it.

3.1. Declaring an array

An array variable is like other variables -- you must declare it, which means you must declare the type of elements that are in an array. All elements must be the same type. Write the element type name, then "[]", then the name of the array variable. The declaration allocates only enough space for a reference to an array (typically 4 bytes), but doesn't create the actual array object.

```
String[]  args;   // args is an array of Strings
int[]     scores; // scores is an array of ints
JButton[] bs;     // bs is an array of JButtons
```

No size in declaration. Unlike some languages, never put the size of the array in the declaration because an array declaration specifies only the element type and the variable name.

Allocate an array object with new. Create an array using new. This example creates an array of 100 int elements, from a[0] to a[99].

```
int[] a;           // Declare a to be an array of ints
a = new int[100];  // Allocate an array of 100 ints
```

These are often combined in one line.

```
int[] a = new int[100];  // Declare and allocate.
```

Subscripts
Subscripts are enclosed in square brackets []. x_i in mathematics is x[i] in Java, and is pronounced "x-sub-i".
Subscript ranges always start at zero because Java came largely from C++, which had a good reason for using zero (pointer arithmetic on arrays). It isn't the way that humans normally count; you'll just have to live with it.
Java always checks subscript legality to be sure the subscript is >= 0, and less than the number of elements in the array. If the subscript is outside this range, Java throws ArrayIndexOutOfBoundsException. This is far superior to the behavior of C and C++, which allow out of range references. Consequently, Java programs are far less susceptible to bugs and security flaws than C/C++ programs.

Length of an array
Each array has a constant (final) instance variable that has its length. You can find out how many elements an array can hold by writing the array name followed by **.length**. In the previous example,

`a.length` would be 100. Remember that this is the number of elements in the array, one more than the maximum subscript.

Java idiom for looping over an array

The most common use of .length is in a for loop test condition. For example, the variable i will go over the entire range of subscripts of the array a.

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

If you only need to reference the value of each of the elements, you can use the somewhat simpler Java 5 for loop, which keeps track of the index and assigns successive values to a variable (v in this example).

```
for (int v : a) {
    . . .
}
```

Example Version 1 – Adding all elements of an array

These statements create an array and put 1000 random values in it. The second loop adds all 1000 elements. It would have been better to add them in the first loop, but writing it this way allows two examples of loops.

```
int[] a;           // Declare an array of ints
a = new int[1000]; // Create an array of 1000 ints.

//... Assign random values to each element.
for (int i=0; I < a.length; i++)
{
    a[i] = (int)(Math.random() * 100000);  // Random number 0-99999
}

//... Add all values in the array.
int sum = 0;           // Start the total sum at 0.
for (int i=0; i<a.length; i++)
{
    sum = sum + a[i];  // Add the next element to the total
}
```

Example Version 2 – Adding all elements of an array in Java 5

This is the the same as above, but uses the Java 5 "for each" loop to do the sum, which frees you from using an index if you simply need to get all successive values. This kind of loop only gets the values, so it couldn't have been used to set the values in the first loop above.

```
. . .
int sum = 0;        // Start the total sum at 0.
for (int v : a)
{
    sum = sum + v;  // Add the next element to the total
}
```

Initial array element values -- zero/null/false

When an array is allocated (with new), all elements are set to an initial value. The initial value is 0 if the element type is numeric (int, float, ...), false for boolean, and null for all object types.

3.2. Array Initialization

When you declare an array, you can can also allocate a preinitialized array object in the same statement. In this case, do not give the array size because Java counts the number of values to determine the size. For example,

```
// Java 1.0 style -- shorter, but can be used ONLY IN DECLARATIONS
String[] days = {"Su", "Mo", "Tu", "We", "Th", "Fr", "Sa"};
```

Array variables are references to arrays

When you declare an array variable, Java reserves only enough memory for a reference (Java's name for an address or pointer) to an array object. References typically require only 4 bytes. When an array object is created with new, a reference is returned, and that reference can then be assigned to a variable. When you assign one array variable to another, only the reference is copied. For example,

```
int[] a = new int[] {100, 99, 98};
int[] b;
// "a" points to an array, and "b" doesn't point to anything
b = a;      // Now b referes to the SAME array as "a"
b[1] = 0;   // Also changes a[1] because a and b refer to the same array.
// Both a and b refer to same array with value {100, 0, 98}
```

3.3. Intermediate Arrays

Anonymous arrays
Java 2 added anonymous arrays, which allow you to create a new array of values anywhere in the program, not just in an initialization in a declaration.

```
// This anonymous array style can also be used in other statements.
String[] days = new String[] {"Su", "Mo", "Tu", "We", "Th", "Fr", "Sa"};
```

You can also use anonymous array syntax in other parts of the program. For example,

```
// Outside a declaration you can make this assignment.
x = new String[] {"Su", "Mo", "Tu", "We", "Th", "Fr", "Sa"};
```

You must be careful not to create these anonymous arrays in a loop or as local variables because each use of new will create another array.

Dynamic allocation

Because arrays are allocated dynamically, the initialization values may arbitrary expresssions. For example, this call creates two new arrays to pass as parameters to drawPolygon.

```
g.drawPolygon(new int[] {n, n+45, 188}, new int[] {y/2, y*2, y}, 3);
```

C-style array declarations
Java also allows you to write the square brackets after the variable name, instead of after the type. This is how you write array declarations in C, *but is not a good style for Java*. C declaration syntax can get very ugly with part of the declaration before the variable, and part after. Java has a much nicer style where all type information can be written together without the use of a variable, and there are times when only the Java notation is possible to use.

```
int[] a;   // Java style -- good
int a[];   // C style -- legal, but not Java style
```

3.4. Common array problems

Some common array programming mistakes are:

- Forgetting that array subscripts start with zero.
- Writing a.length() instead of a.length. The length() method is used with Strings, not arrays.
- Declaring an array with a size. E.g., int[100] a; instead of int[] a = new int[100];

3.5. Library methods for arrays

Static methods for manipulating arrays are available in the **java.util.Arrays** class.

Arrays.asList()	Returns a List based on the array.
Arrays.toString()	Returns a readable form of the array.
Arrays.binarySearch()	Performs binary search of a sorted array.
Arrays.equals	Compares two arrays for equality.
Arrays.fill()	Fills entire array or subrange with a value.
Arrays.sort()	Sorts an array.

In addition there is System.arrayCopy() method.

Reverse an array

This version of reverse uses two subscripts: one that starts at the left (beginning) of the array, and one that starts at the right (end) of the array. You can also use a for loop that goes to the middle of the array.

```
//========================================================= reverse
public static void reverse(int[] b)
{
   int left  = 0;          // index of leftmost element
   int right = b.length-1; // index of rightmost element

   while (left < right)
   {
      // exchange the left and right elements
      int temp = b[left];
      b[left]  = b[right];
      b[right] = temp;

      // move the bounds toward the center
      left++;
      right--;
   }
}//endmethod reverse
```

A for loop to do this would replace the above 8 statements with these. Both loops are the same speed, so the choice should be for the one which is more readable to you.

```
for (int left=0, int right=b.length-1; left<right; left++, right--) {
    // exchange the first and last
    int temp = b[left]; b[left]  = b[right]; b[right] = temp;
}
```

3.6. Multi-dimensional Arrays

All arrays in Java are really linear, one-dimensional arrays. However, you can easily build multi-dimensional arrays from these, and Java has features in the language to help you do this.

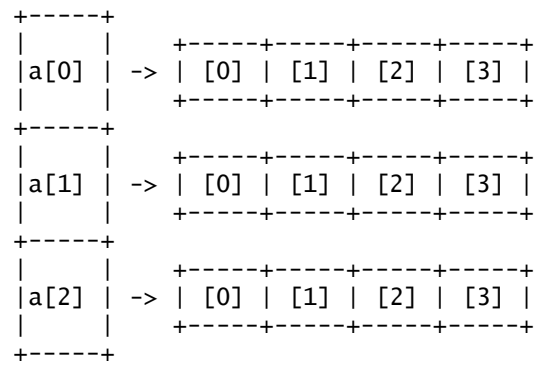
These examples all use two-dimensional arrays, but the same syntax and coding can easily be extended to arrays of any dimension. By convention two dimensional arrays have rows (horizontal) and columns (vertical). The first subscript selects the row (which is a one-dimensional array itself), and the second subscript selects the element in that row/array.

Visualizing two-dimensional arrays

Assume we have an array, a, with three rows and four columns.

```
a[0][0]       a[0][1]       a[0][2]       a[0][3]
a[1][0]       a[1][1]       a[1][2]       a[1][3]
a[2][0]       a[2][1]       a[2][2]       a[2][3]
```

Two-dimensional arrays are usually visualized as a matrix, with rows and columns. This diagram shows the array a with its corresponding subscripts.



In Java two-dimensional arrays are implemented is a one-dimensional array of one-dimensional arrays – like this.

Declaring and Allocating a two-dimensional array

Let's declare a board for playing the game of tic-tac-toe. It will have three rows (the first subscript) and three columns (the second subscript) and contain an int in each element.

```
int[][] board = new int[3][3];
```

Initial values

It's possible to assign initial values to an array when you declare it in a manner very similar to one-dimensional arrays, but with an extra level of braces. The dimension sizes are computed by the compiler from the number of values.

```
int[][] board = new int[][] {{0,0,0},{0,0,0},{0,0,0}};
```

You must assign values to an element before you use it, either with an initializer as above or assignment. Example -- drawing the tic-tac-toe board

It's often easiest to use two-dimensional arrays with nested for loops. For example, the following code draws the tic-tac-toe board in a paint method. It assumes that a cell is 10 pixels on a side, and that a positive number represents an X and a negative number represents a O.

```
for (int row=0; row<3; row++)
{
    for (int col=0; col<3; col++)
    {
        if (board[row][col] > 0) { // draw X
            g.drawLine(col*10, row*10  , col*10+8, row*10+8);
            g.drawLine(col*10, row*10+8, col*10+8, row*10  );
        }
        else if (board[row][col] < 0)
        { // draw O
            g.drawOval(col*10, row*10, 8, 8);
        }
    }
}
```

4. Lab Tasks

4.1. Study and understand the given text and examples

4.2. Define a class called Matrix which implements the matrix addition, subtraction, multiplication, and the division by a value operations.

4.3. Implement maintaining ChessBoard class. The figures and pawns are also classes. You should check for valid moves for every piece.