Sukkur Institute of Business Administration University
Department of Computer Science
BS – II (CS/SE/AI) Spring 2024

Object Oriented Programming

Lab # 03: To become familiar with Java Math Class and Arrays (One-Dimensional and Multi-Dimensional and Uneven Arrays) in Java

Instructor: Engr. Zainab Umair Kamangar

Lab Report Rubrics (Add the points in each column, then add across the bottom row to find the total score)					**Total Marks**
S.No	**Criterion**	**0.5**	**0.25**	**0.125**	
1	Accuracy	☐ Desired output	☐ Minor mistakes	☐ Critical mistakes	
2	Timing	☐ Submitted within the given time	☐ 1 day late	☐ More than 1 day late	

Submission Profile

Name: Submission date (dd/mm/yy):
Enrollment ID: Receiving authority name and signature:
Comments:

_____
Instructor Signature

Note: Submit this lab hand-out in the next lab with attached solved activities and exercises

Objectives

After performing this lab, students will be able to understand,

- Some differences of type conversion rules as compared to C++
- Automatic Type Promotion in Expressions
- Comments in Java
- Java Math Class
- Arrays (One Dimension and Multi Dimension)
- Uneven Array (Jagged Array)

Type Conversion Rules:

Main differences as Compared to C++:

Aspect	C++	Java
Integer Conversions	Allows implicit narrowing conversions	Generally, avoids implicit narrowing conversions
Implicit Casting	More implicit casting possibilities	Stricter, relies more on explicit casting
Operator Overloading	Allows operator overloading for custom types	Does not allow operator overloading for primitive types
User Control	Offers more flexibility and control over conversions	Offers explicit casting with less flexibility

Example differences:

1. **Narrowing conversion:**

   C++

   int x = 256;

   char c = x; // In C++, this silently truncates x to fit in a char (potentially overflows), while Java would trigger an error.

2. **Implicit casting:**

   C++

   float f = 3.14;

   int sum = f + 1; // In C++, f is implicitly cast to an int without warning, while Java would require explicit casting.

Automatic type promotion in Expressions

When evaluating expressions involving multiple primitive data types, Java automatically promotes smaller data types to larger data types to ensure accurate results and prevent data loss.

- This happens without explicit casting, making code more concise and readable.

Rules:

1. **byte, short, and char** are always promoted to **int**.
2. If any operand is **long**, the expression is promoted to **long**.
3. If any operand is **float**, the expression is promoted to **float**.
4. If any operand is **double**, the expression is promoted to **double**.

Example 1: byte and short promoted to int

```
byte b = 10;
short s = 20;
int result = b + s; // b and s are promoted to int before addition
System.out.println(result); // Output: 30
```

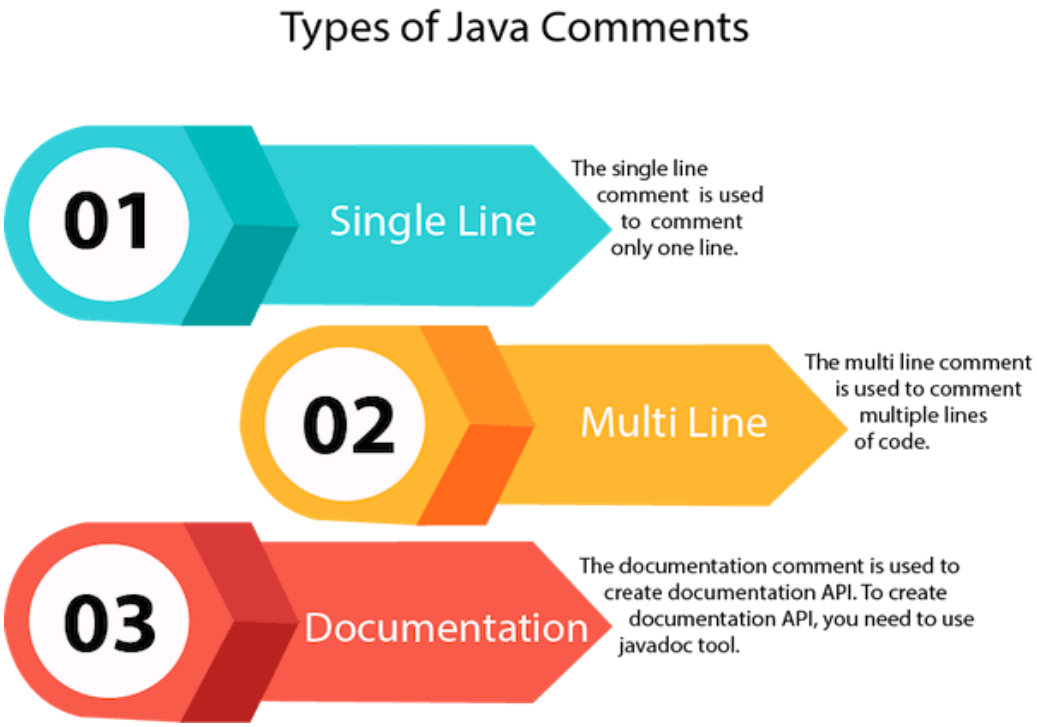
Example 2: long promotion

```
int x = 100;
long y = 200L;
long sum = x + y; // x is promoted to long before addition
System.out.println(sum); // Output: 300
```

Example 3: float promotion

```
int a = 5;
float b = 2.5f;
float result2 = a / b; // a is promoted to float before division
System.out.println(result2); // Output: 2.0
```

Java Comments



Single line	Multi-line	Documentation comment
//This is single line comment	/* This is a multi-line comment */	/** * *We can use various tags to depict the *parameter *or heading or author name *We can also use HTML tags * */

Java Documentation Comments

```
import java.io.*;

/**
* Add Two Numbers!
* The AddNum program implements an application that
* simply adds two given integer numbers and Prints
* the output on the screen.
* <p>
* <b>Note:</b> Giving proper comments in your program makes it more
* user friendly and it is assumed as a high quality code.
*
* @author  Zainab Umair
```

```
* @version 21.0
* @since   2024-01-29
*/

public class AddNum {
  /**
  * This method is used to add two integers. This is
  * a the simplest form of a class method, just to
  * show the usage of various javadoc Tags.
  * @param numA This is the first paramter to addNum method
  * @param numB  This is the second parameter to addNum method
  * @return int This returns sum of numA and numB.
  */

  public int addNum(int numA, int numB) {
    return numA + numB;
  }

  /**
  * This is the main method which makes use of addNum method.
  * @param args Unused.
  * @exception IOException On input error.
  * @see IOException
  */

  public static void main(String args[]) throws IOException {
    AddNum obj = new AddNum();
    int sum = obj.addNum(10, 20);

    System.out.println("Sum of 10 and 20 is :" + sum);
  }
}
```

The javadoc tool recognizes the following tags:

https://www.tutorialspoint.com/java/java_documentation.htm

https://www.javatpoint.com/java-comments

The javadoc Tags

Tag	Description	Syntax
@author	Adds the author of a class.	@author name-text
{@code}	Displays text in code font without interpreting the text as HTML	{@code text}

	markup or nested javadoc tags.	
{@docRoot}	Represents the relative path to the generated document's root directory from any generated page.	{@docRoot}
@deprecated	Adds a comment indicating that this API should no longer be used.	@deprecated deprecatedtext
@exception	Adds a **Throws** subheading to the generated documentation, with the classname and description text.	@exception class-name description
{@inheritDoc}	Inherits a comment from the **nearest** inheritable class or implementable interface.	Inherits a comment from the immediate surperclass.
{@link}	Inserts an in-line link with the visible text label that points to the documentation for the specified package, class, or member name of a referenced class.	{@link package.class#member label}
{@linkplain}	Identical to {@link}, except the link's label is displayed in plain text than code font.	{@linkplain package.class#member label}
@param	Adds a parameter with the specified parameter-name followed by the specified description to the "Parameters" section.	@param parameter-name description
@return	Adds a "Returns" section with the description text.	@return description
@see	Adds a "See Also" heading with a link or text entry that points to reference.	@see reference
@serial	Used in the doc comment for a default serializable field.	@serial field-description | include | exclude
@serialData	Documents the data written by the writeObject( ) or writeExternal( ) methods.	@serialData data-description
@serialField	Documents an ObjectStreamField	@serialField field-name field-

	component.	type field-description
@since	Adds a "Since" heading with the specified since-text to the generated documentation.	@since release
@throws	The @throws and @exception tags are synonyms.	@throws class-name description
{@value}	When {@value} is used in the doc comment of a static field, it displays the value of that constant.	{@value package.class#field}
@version	Adds a "Version" subheading with the specified version-text to the generated docs when the -version option is used.	@version version-text

JAVA Math Class

https://docs.oracle.com/javase/8/docs/api/java/lang/Math.html

The Java programming language supports basic arithmetic with its arithmetic operators: +, -, *, /, and %. The Math class provides methods and constants for doing more advanced mathematical computation.

The Math is located in the java.lang package, and not in the java.math package. Thus, the fully qualified class name of the Math class is java.lang.Math

The methods in the Math class are all static, so you call them directly from the class, like this:

Math.cos(angle);

***Note:** Using the static import language feature, you don't have to write Math in front of every math function: import static java.lang.Math;

This allows you to invoke the Math class methods by their simple names.

For example: cos(angle);

Constants

The Math class includes two constants:

- Math.E, which is the base of natural logarithms, and
- Math.PI, which is the ratio of the circumference of a circle to its diameter.

Basic Math Methods

The Math class includes more than 40 static methods. They can be categorized as *trigonometric methods*, *exponent methods*, and *service methods*. Service methods include the rounding, min, max, absolute, and random methods.

Trigonometric Methods

The Math class contains the following methods.

Method	*Description*
`sin(radians)`	Returns the trigonometric sine of an angle in radians.
`cos(radians)`	Returns the trigonometric cosine of an angle in radians.
`tan(radians)`	Returns the trigonometric tangent of an angle in radians.
`toRadians(degree)`	Returns the angle in radians for the angle in degree.
`toDegree(radians)`	Returns the angle in degrees for the angle in radians.
`asin(a)`	Returns the angle in radians for the inverse of sine.
`acos(a)`	Returns the angle in radians for the inverse of cosine.
`atan(a)`	Returns the angle in radians for the inverse of tangent.

The parameter for sin, cos, and tan is an angle in radians. The return value for asin, acos, and atan is a degree in radians in the range between -pi/2 and pi/2.

- One degree is equal to pi/180 in radians,
- 90 degrees is equal to pi/2 in radians
- 30 degrees is equal to pi/6 in radians.

Exponent Methods

There are five methods related to exponents in the Math class.

Method	*Description*
`exp(x)`	Returns e raised to power of x (e^x).
`log(x)`	Returns the natural logarithm of x ($\ln(x) = \log_e(x)$).
`log10(x)`	Returns the base 10 logarithm of x ($\log_{10}(x)$).
`pow(a, b)`	Returns a raised to the power of b (a^b).
`sqrt(x)`	Returns the square root of x ($\sqrt{x}$) for x >= 0.

The Rounding Methods

The Math class contains five rounding methods

Method	*Description*
`ceil(x)`	x is rounded up to its nearest integer. This integer is returned as a double value.
`floor(x)`	x is rounded down to its nearest integer. This integer is returned as a double value.
`rint(x)`	x is rounded up to its nearest integer. If x is equally close to two integers, the even one is returned as a double value.
`round(x)`	Returns (int)Math.floor(x + 0.5) if x is a float and returns (long)Math.floor(x + 0.5) if x is a double.

The Service Methods

The **min**, **max**, and **abs** Methods

The **min** and **max** methods return the minimum and maximum numbers of two numbers (**int**, **long**, **float**, or **double**).

For example, **max(4.4, 5.0)** returns **5.0**, and **min(3, 2)** returns **2**.

The **abs** method returns the absolute value of the number (**int**, **long**, **float**, or **double**).

This method generates a random **double** value greater than or equal to 0.0 and less than 1.0 (**0 <= Math.random() < 1.0**). You can use it to write a simple expression to generate random numbers in any range.

Arrays in Java

Making an array in a Java program involves three distinct steps:

- Declare the array name.
- Create the array.
- Initialize the array values.

We refer to an array element by putting its index in square brackets after the array name.

To use an array in a program, you must declare a variable to reference the array and specify the array's *element type*.

Syntax:

elementType[] arrayRefVar;

The **elementType** can be any data type, and all elements in the array will have the same data type.

Say suppose :

int[] arrayRef;

Unlike declarations for primitive data type variables, the declaration of an array variable does not allocate any space in memory for the array. It creates only a storage location for the reference to an array.

If a variable does not contain a reference to an array, the value of the variable is **null**. You cannot assign elements to an array unless it has already been created. After an array variable is declared, you can create an array by using the **new** operator and assign its reference to the variable with the following **syntax**:

arrayRefVar = new elementType[arraySize];

// integer array

int[] arr;

arr = new int[10];

Java has a shorthand notation, known as the *array initializer*, which combines the declaration, creation, and initialization of an array in one statement using the following **syntax**:

elementType[] arrayRefVar = {value0, value1, ..., valuek};

int[] arr= {10,20,30,40,50,60,70,80,90,100};

Arrays Class

Arrays class which is in java.util.Arrays package, is a provision by Java that provides you several methods through which arrays can be manipulated. This class also lets you perform sorting and searching operations on an array.

https://www.geeksforgeeks.org/array-class-in-java/

https://docs.oracle.com/javase/8/docs/api/java/util/Arrays.html

Multi-dimensional Arrays:

A multidimensional array is an array of arrays. Each element of a multidimensional array is an array itself. For example,

```
int[][] a = new int[3][4];
```

Here, we have created a multidimensional array named a. It is a 2-dimensional array, that can hold a maximum of 12 elements,

	Column 1	Column 2	Column 3	Column 4
Row 1	a[0][0]	a[0][1]	a[0][2]	a[0][3]
Row 2	a[1][0]	a[1][1]	a[1][2]	a[1][3]
Row 3	a[2][0]	a[2][1]	a[2][2]	a[2][3]

Here is how we can initialize a 2-dimensional array in Java.

```
int[][] a = {
    {1, 2, 3},
    {4, 5, 6, 9},
    {7},
};
```

Example: 2-dimensional Array

```
class MultidimensionalArray {
  public static void main(String[] args) {
    // create a 2d array
    int[][] a = {
      {1, 2, 3},
      {4, 5, 6, 9},
      {7},
    };
```

```
        // calculate the length of each row
        System.out.println("Length of row 1: " + a[0].length);
        System.out.println("Length of row 2: " + a[1].length);
        System.out.println("Length of row 3: " + a[2].length);
    }
}
```

Output:

```
Length of row 1: 3
Length of row 2: 4
Length of row 3: 1
```

In the above example, we are creating a multidimensional array named a. Since each component of a multidimensional array is also an array (a[0], a[1] and a[2] are also arrays). Here, we are using the length attribute to calculate the length of each row.

Example: Print all elements of 2d array Using Loop

```
class MultidimensionalArray {
    public static void main(String[] args) {
        int[][] a = {
            {1, -2, 3},
            {-4, -5, 6, 9},
            {7},
        };
        for (int i = 0; i < a.length; ++i) {
            for(int j = 0; j < a[i].length; ++j) {
                System.out.println(a[i][j]);
            }
        }
    }
}
```

Output:

```
1
-2
3
-4
-5
6
9
7
```

We can also use the for...each loop to access elements of the multidimensional array. For example,

```
class MultidimensionalArray {
   public static void main(String[] args) {

      // create a 2d array
      int[][] a = {
         {1, -2, 3},
         {-4, -5, 6, 9},
         {7},
      };

      // first for...each loop access the individual array
      // inside the 2d array
      for (int[] innerArray: a) {
         // second for...each loop access each element inside the row
         for(int data: innerArray) {
            System.out.println(data);
         }
      }
   }
}
```

Output:

```
1
-2
3
-4
-5
6
9
7
```

In the above example, we are have created a 2d array named a. We then used for loop and for...each loop to access each element of the array.

Remember, Java uses zero-based indexing, that is, indexing of arrays in Java starts with 0 and not 1.

Let's take another example of the multidimensional array. This time we will be creating a 3-dimensional array. For example,

String[][][] data = new String[3][4][2];

Here, data is a 3d array that can hold a maximum of 24 (3*4*2) elements of type String.

Let's see how we can use a 3d array in Java. We can initialize a 3d array similar to the 2d array. For example,

```
// test is a 3d array
int[][][] test = {
      {
       {1, -2, 3},
       {2, 3, 4}
      },
      {
       {-4, -5, 6, 9},
       {1},
       {2, 3}
      }
};
```

Basically, a 3d array is an array of 2d arrays. The rows of a 3d array can also vary in length just like in a 2d array.

Example: 3-dimensional Array

```
class ThreeArray {
   public static void main(String[] args) {
      // create a 3d array
      int[][][] test = {
         {
          {1, -2, 3},
          {2, 3, 4}
         },
         {
          {-4, -5, 6, 9},
          {1},
          {2, 3}
         }
      };
      // for..each loop to iterate through elements of 3d array
      for (int[][] array2D: test) {
         for (int[] array1D: array2D) {
            for(int item: array1D) {
```

```
            System.out.println(item);
          }
        }
      }
    }
}
```

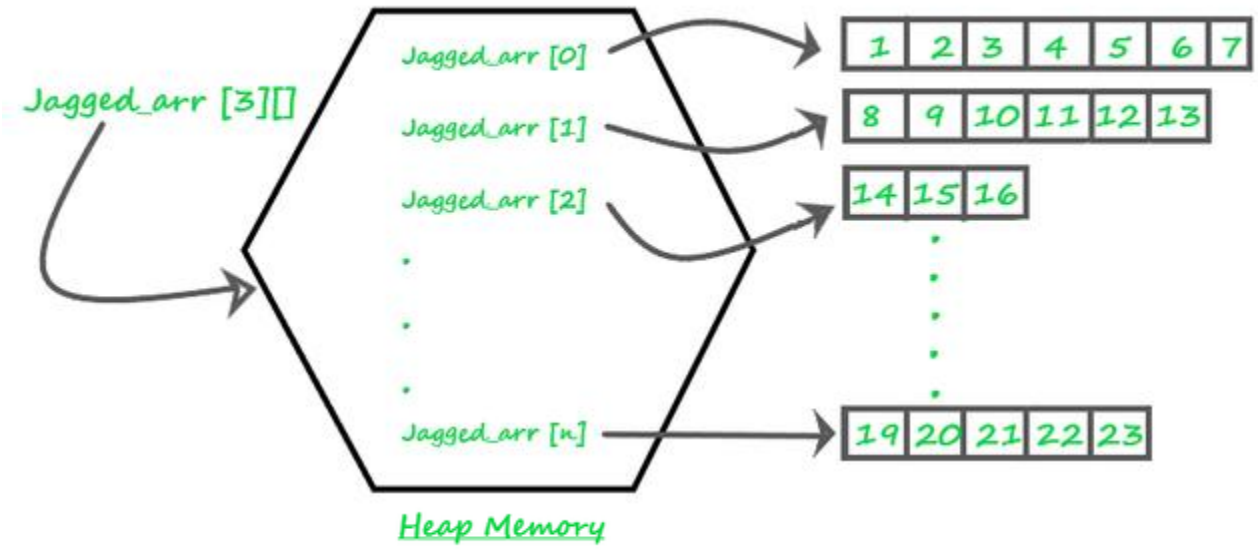
Output:

```
1
-2
3
2
3
4
-4
-5
6
9
1
2
3
```

Uneven Array (Jagged Array)

A jagged array is an array of arrays such that member arrays can be of different sizes, i.e., we can create a 2-D array but with a variable number of columns in each row. These types of arrays are also known as Jagged arrays.

Pictorial representation of Jagged array in Memory:



Declaration and Initialization of Jagged array :

Syntax: data_type array_name[][] = new data_type[n][]; //n: no. of rows
array_name[] = new data_type[n1] //n1= no. of columns in row-1
array_name[] = new data_type[n2] //n2= no. of columns in row-2
array_name[] = new data_type[n3] //n3= no. of columns in row-3
.
.
.
array_name[] = new data_type[nk] //nk=no. of columns in row-n

Alternative, ways to Initialize a Jagged array :

```
int arr_name[][] = new int[][]  {
        new int[] {10, 20, 30 ,40},
        new int[] {50, 60, 70, 80, 90, 100},
        new int[] {110, 120}
          };
```

OR

```
int[][] arr_name = {
    new int[] {10, 20, 30 ,40},
    new int[] {50, 60, 70, 80, 90, 100},
    new int[] {110, 120}
      };
```

OR

```
int[][] arr_name = {
    {10, 20, 30 ,40},
    {50, 60, 70, 80, 90, 100},
    {110, 120}
      };
```

Following are Java programs to demonstrate the above concept.

Example:1

```
// Program to demonstrate 2-D jagged array in Java
class Main {
  public static void main(String[] args)
  {
    // Declaring 2-D array with 2 rows
    int arr[][] = new int[2][];

    // Making the above array Jagged

    // First row has 3 columns
    arr[0] = new int[3];

    // Second row has 2 columns
    arr[1] = new int[2];

    // Initializing array
    int count = 0;
    for (int i = 0; i < arr.length; i++)
      for (int j = 0; j < arr[i].length; j++)
        arr[i][j] = count++;

    // Displaying the values of 2D Jagged array
    System.out.println("Contents of 2D Jagged Array");
    for (int i = 0; i < arr.length; i++) {
      for (int j = 0; j < arr[i].length; j++)
        System.out.print(arr[i][j] + " ");
      System.out.println();
    }
  }
}
```

Output

```
Contents of 2D Jagged Array
0 1 2
3 4
```

Example: 2

Following is another example where i'th row has i columns, i.e., the first row has 1 element, the second row has two elements and so on.

```
// Another Java program to demonstrate 2-D jagged
// array such that first row has 1 element, second
// row has two elements and so on.
class Main {
   public static void main(String[] args)
   {
      int r = 5;
       // Declaring 2-D array with 5 rows
      int arr[][] = new int[r][];

      // Creating a 2D array such that first row
      // has 1 element, second row has two
      // elements and so on.
      for (int i = 0; i < arr.length; i++)
         arr[i] = new int[i + 1];
      // Initializing array
      int count = 0;
      for (int i = 0; i < arr.length; i++)
         for (int j = 0; j < arr[i].length; j++)
            arr[i][j] = count++;
      // Displaying the values of 2D Jagged array
      System.out.println("Contents of 2D Jagged Array");
      for (int i = 0; i < arr.length; i++) {
         for (int j = 0; j < arr[i].length; j++)
            System.out.print(arr[i][j] + " ");
         System.out.println();
      }
   }
}
```

Output

```
Contents of 2D Jagged Array
0
1 2
3 4 5
6 7 8 9
10 11 12 13 14
```

Example: 3

```
import java.util.Scanner;

public class JaggedArray {
   public static void main(String[] args) {
      Scanner scan = new Scanner(System.in);
      System.out.print("Enter the number of sub-arrays: ");
      int numberOfArrays = scan.nextInt();

      // Declare the jagged array
      int[][] jaggedArray = new int[numberOfArrays][];

      // Allocate memory to each sub-array
      for (int i = 0; i < numberOfArrays; i++) {
         System.out.print("Enter the size of sub-array " + (i + 1) + ": ");
         int sizeOfSubArray = scan.nextInt();
         jaggedArray[i] = new int[sizeOfSubArray];
      }

      // Initialize the elements of each sub-array
      for (int i = 0; i < numberOfArrays; i++) {
         System.out.println("Enter the elements of sub-array " + (i + 1) + ":");
         for (int j = 0; j < jaggedArray[i].length; j++) {
            jaggedArray[i][j] = scan.nextInt();
         }
      }

      // Print the elements of the jagged array
      System.out.println("The jagged array is:");
      for (int i = 0; i < numberOfArrays; i++) {
         for (int j = 0; j < jaggedArray[i].length; j++) {
            System.out.print(jaggedArray[i][j] + " ");
         }
         System.out.println();
      }

      scan.close();
   }
}
```

Jagged arrays have the following advantages in Java:

- ***Dynamic allocation:*** Jagged arrays allow you to allocate memory dynamically, meaning that you can specify the size of each sub-array at runtime, rather than at compile-time.
- ***Space utilization:*** Jagged arrays can save memory when the size of each sub-array is not equal. In a rectangular array, all sub-arrays must have the same size, even if some of them have unused elements. With a jagged array, you can allocate just the amount of memory that you need for each sub-array.
- ***Flexibility:*** Jagged arrays can be useful when you need to store arrays of different lengths or when the number of elements in each sub-array is not known in advance.
- ***Improved performance***: Jagged arrays can be faster than rectangular arrays for certain operations, such as accessing elements or iterating over sub-arrays because the memory layout is more compact.

It's important to note that jagged arrays also have some ***disadvantages***, such as ***increased complexity*** in the code and a potentially ***less readable codebase***, so they should be used carefully and appropriately.

Exercises

Question 1: (Marks average)
Write a Java program that creates an array of student's marks. You are required to calculate the average marks of the students. The array length should be 20.

Question 2: (Transpose matrix)
Write a program that takes elements of 2D matrix using an array and generates output as the transpose of the matrix.

Question: 3 (Matrix Addition)
Write a program that takes elements of two 3x3 matrices using an array and generates output as the addition of the matrix.

Question: 4 (Matrix Multiplication)
Write a program that takes elements of two 3x3 matrices using an array and generates output as the multiplication of the matrix.

Question: 5 (Row Average)
Write a program that takes elements of 2D matrix using an array and generates output as the average of each row of the matrix.

Question: 6 (Length of sub-array in Jagged array)
Write a program that takes elements of a jagged array from user and print the length of each sub-array.

Question: 7 (Sort each sub-array in Jagged array)
Write a program that takes elements of a jagged array from user and print the elements of each sub-array of a jagged array in ascending order.

Question: 8 (Quadratic Formula)
Write a program that solves quadratic equations of the form: $ax^2 + bx + c = 0$. Values of a, b, and c can be taken as input from the user.

Question 9: (Round up)
Write a Java program to round up the result of the variable division.

Question 10: (Degrees and Radians)
Write a Java program that converts degrees into radians and radians into degrees.