ECE25100: OBJECT ORIENTED PROGRAMMING

Note 5 _ Arrays

Instructor: Xiaoli Yang

Introducing Arrays

- An Array is a bounded (fixed size) collection of elements of the same type. Arrays are:
  - indexed - elements are accessed/modified according to its position (i.e., index) in the array (starting at 0).
  - fixed size - we cannot add more elements or remove some from an array once it has been created.
  - strongly typed and homogeneous - all elements are of the same type.
  - "Bounds - checked" - if we try to access outside of an array, JAVA stops us.

Declaring and Creating Array Variables

- Declaring array variables:
  - `datatype[ ] arrayRefVar;`

    Example: `double[ ] myList;`
  - `datatype arrayRefVar[ ];` // This style is allowed, but not preferred

    Example: `double myList[ ];`
- Creating arrays
  - `arrayRefVar = new datatype[arraySize];`

    Example: `myList = new double[10];`

    myList[0] references the first element in the array.

    myList[9] references the last element in the array.

Declaring and Creating Array Variables

- Declaring and Creating in one step
  - datatype[] arrayRefVar = new datatype[arraySize];

    double[] myList = new double[10];

    int[] sickDays = new int[30];

    person[] friends = new Person[50];
  - datatype arrayRefVar[] = new datatype[arraySize];

    double myList[] = new double[10];
- Newly created arrays are filled with:
  - 0 for numbered arrays
  - character 0 (i.e., the null character) for char arrays
  - false for boolean arrays
  - null for reference type (i.e., Object) arrays

Declaring and Creating Array Variables

- Declaring, creating, initializing in one step:

```
double[ ] myList = {1.9, 2.9, 3.4, 3.5};
```

This shorthand syntax must be in one statement.

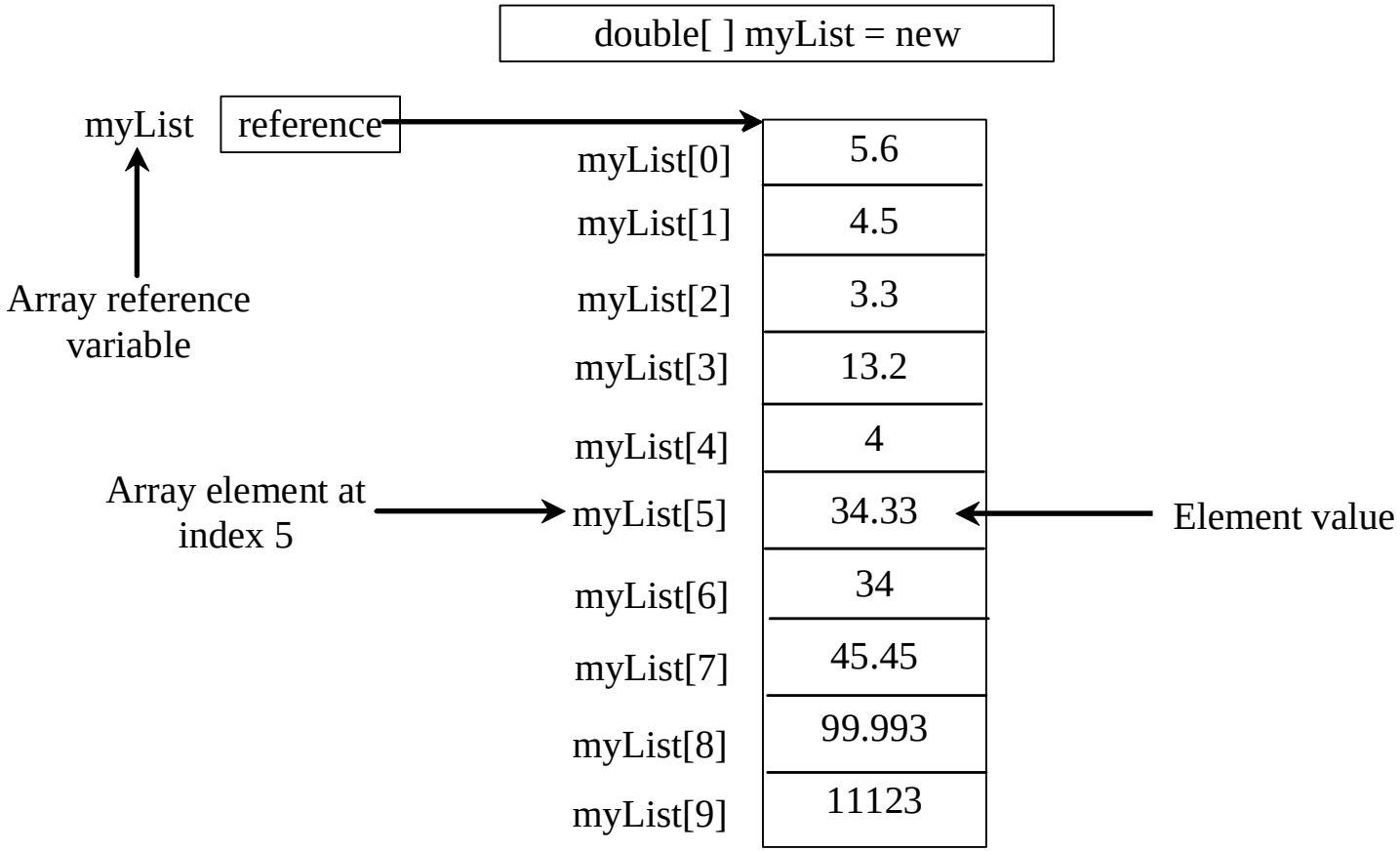
- This shorthand notation is equivalent to the following statements:

```
double[ ] myList = new double[4];
myList[0] = 1.9;
myList[1] = 2.9;
myList[2] = 3.4;
myList[3] = 3.5;
```

- Using the shorthand notation, you have to declare, create, and initialize the array all in one statement. Splitting it would cause a syntax error. For example, the following is wrong:

```
double[ ] myList;
myList = {1.9, 2.9, 3.4, 3.5};
```

Introducing Arrays (Cont.)



Introducing Arrays (Cont.)

- The array elements are accessed through the index. The array indices are 0-based, i.e., it starts from 0 to arrayRefVar.length-1. the figure on the previous slide, myList holds ten double values and the indices are from 0 to 9.
- Each element in the array is represented using the following syntax, known as an indexed variable:

```
arrayRefVar[index];
```

Introducing Arrays (Cont.)

- After an array is created, an indexed variable can be used in the same way as a regular variable. For example, the following code adds the value in myList[0] and myList[1] to myList[2].

```
myList[2] = myList[0] + myList[1];
```

- Once an array is created, its size is fixed. It cannot be changed. You can find its size (the length of an array) using

```
arrayRefVar.length
```

Example: myList.length returns 10

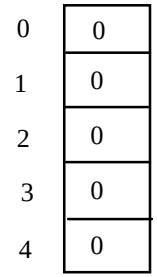
animation

Trace Program with Arrays

Declare array variable values, create an array, and assign its reference to values

```
public class Test {
  public static void main(String[] args) {
    int[] values = new int[5];
    for (int i = 1; i < 5; i++) {
      values[i] = i + values[i-1];
    }
    values[0] = values[1] + values[4];
  }
}
```

After the array is created



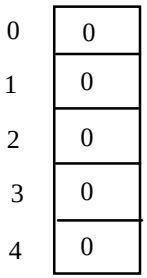
animation

Trace Program with Arrays

i becomes 1

```
public class Test {
  public static void main(String[] args) {
    int[] values = new int[5];
    for (int i = 1; i < 5; i++) {
      values[i] = i + values[i-1];
    }
    values[0] = values[1] + values[4];
  }
}
```

After the array is created



animation

Trace Program with Arrays

i (=1) is less than 5

```
public class Test {
  public static void main(String[] args) {
    int[] values = new int[5];
    for (int i = 1; i < 5; i++) {
      values[i] = i + values[i-1];
    }
    values[0] = values[1] + values[4];
  }
}
```

After the array is created

0	0
1	0
2	0
3	0
4	0

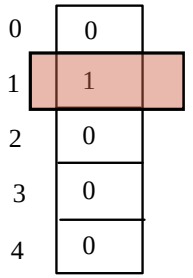
animation

Trace Program with Arrays

After this line is executed, value[1] is 1

```
public class Test {
  public static void main(Strin      rgs) {
    int[] values = new int[5];
    for (int i = 1; i < 5; i++) {
      values[i] = i + values[i-1];
    }
    values[0] = values[1] + values[4];
  }
}
```

After the first iteration



animation

Trace Program with Arrays

After i++, i becomes 2

```
public class Test {
  public static void main(String[] args) {
    int[] values = new int[5];
    for (int i = 1; i < 5; i++) {
      values[i] = i + values[i-1];
    }
    values[0] = values[1] + values[4];
  }
}
```

After the first iteration

0	0
1	1
2	0
3	0
4	0

animation

Trace Program with Arrays

```
public class Test {
  public static void main(String[] args) {
    int[] values = new int[5];
    for (int i = 1; i < 5; i++) {
      values[i] = i + values[i-1];
    }
    values[0] = values[1] + values[4];
  }
}
```

i (= 2) is less than 5

After the first iteration

0	0
1	1
2	0
3	0
4	0

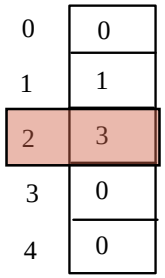
animation

Trace Program with Arrays

After this line is executed,
values[2] is 3 (2 + 1)

```
public class Test {
  public static void main(Str      args) {
    int[] values = new int[5];
    for (int i = 1; i < 5; i++) {
      values[i] = i + values[i-1];
    }
    values[0] = values[1] + values[4];
  }
}
```

After the second iteration



animation

Trace Program with Arrays

After this, i becomes 3.

```
public class Test {
  public static void main(String[] args) {
    int[] values = new int[5];
    for (int i = 1; i < 5; i++) {
      values[i] = i + values[i-1];
    }
    values[0] = values[1] + values[4];
  }
}
```

After the second iteration

0	0
1	1
2	3
3	0
4	0

animation

Trace Program with Arrays

i (=3) is still less than 5.

```
public class Test {
  public static void main(String[] args) {
    int[] values = new int[5];
    for (int i = 1; i < 5; i++) {
      values[i] = i + values[i-1];
    }
    values[0] = values[1] + values[4];
  }
}
```

After the second iteration

0	0
1	1
2	3
3	0
4	0

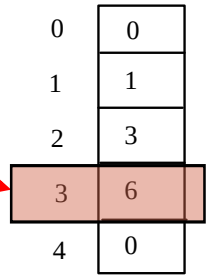
animation

Trace Program with Arrays

```
public class Test {
  public static void main(String[] ar      {
    int[] values = new int[5];
    for (int i = 1; i < 5; i++) {
      values[i] = i + values[i-1];
    }
    values[0] = values[1] + values[4];
  }
}
```

After this line, values[3] becomes 6 (3 + 3)

After the third iteration



animation

Trace Program with Arrays

```
public class Test {
  public static void main(String[] args) {
    int[] values = new int[5];
    for (int i = 1; i < 5; i++) {
      values[i] = i + values[i-1];
    }
    values[0] = values[1] + values[4];
  }
}
```

After this, i becomes 4

After the third iteration

0	0
1	1
2	3
3	6
4	0

animation

Trace Program with Arrays

i (=4) is still less than 5

```
public class Test {
  public static void main(String[] args) {
    int[] values = new int[5];
    for (int i = 1; i < 5; i++) {
      values[i] = i + values[i-1];
    }
    values[0] = values[1] + values[4];
  }
}
```

After the third iteration

0	0
1	1
2	3
3	6
4	0

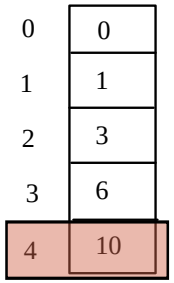
animation

Trace Program with Arrays

After this, values[4] becomes 10 (4 + 6)

```
public class Test {
  public static void main(S    g[] args) {
    int[] values = new int[5
    for (int i = 1; i < 5; i++) {
      values[i] = i + values[i-1];
    }
    values[0] = values[1] + values[4];
  }
}
```

After the fourth iteration



animation

Trace Program with Arrays

After i++, i becomes 5

```
public class Test {
  public static void main(String[] args) {
    int[] values = new int[5];
    for (int i = 1; i < 5; i++) {
      values[i] = i + values[i-1];
    }
    values[0] = values[1] + values[4];
  }
}
```

After the fourth iteration

0	0
1	1
2	3
3	6
4	10

animation

Trace Program with Arrays

i (=5) < 5 is false. Exit the loop

```
public class Test {
  public static void main(String[] args) {
    int[] values = new int[5];
    for (int i = 1; i < 5; i++) {
      values[i] = i + values[i-1];
    }
    values[0] = values[1] + values[4];
  }
}
```

After the fourth iteration

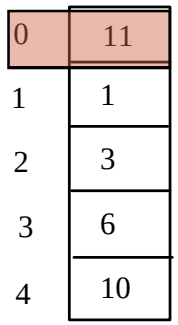
0	0
1	1
2	3
3	6
4	10

animation

Trace Program with Arrays

After this line, values[0] is 11 (1 + 10)

```
public class Test {
  public static void main(S      [] args) {
    int[] values = new int
    for (int i = 1; i < 5; i      ) {
      values[i] = i + va      es[i-1];
    }
    values[0] = values[1] + values[4];
  }
}
```



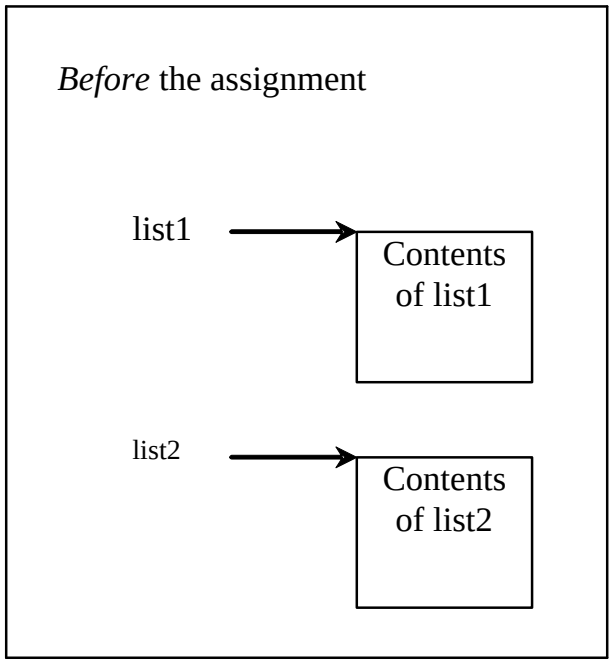
Case Study: Assigning Grades

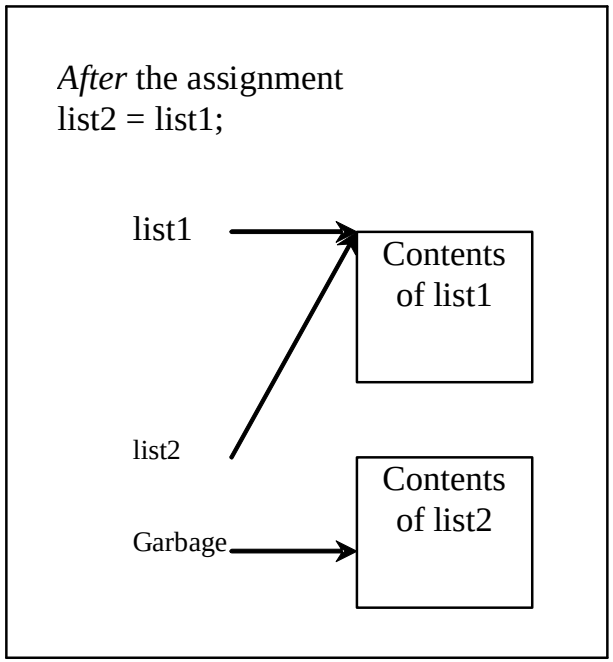
- Objective: read 20 student scores (int), get the best score, and then assign grades based on the following scheme:
  - Grade is A if score is >= best–10;
  - Grade is B if score is >= best–20;
  - Grade is C if score is >= best–30;
  - Grade is D if score is >= best–40;
  - Grade is F otherwise.

Copying Arrays

- Often, in a program, you need to duplicate an array or a part of an array. In such cases you could attempt to use the assignment statement (=), as follows:

```
list2 = list1;
```





Copying Arrays

- Using a loop:

```
int[] sourceArray = {2, 3, 1, 5, 10};
int[] targetArray = new int[sourceArray.length];
for (int i = 0; i < sourceArrays.length; i++)
targetArray[i] = sourceArray[i];
```

- The arraycopy Utility

```
arraycopy(sourceArray, src_pos, targetArray, tar_pos, length);
```

Example:

```
System.arraycopy(sourceArray, 0, targetArray, 0, sourceArray.length);
```

Passing Arrays to Methods

```
public static void printArray(int[] array) {
  for (int i = 0; i < array.length; i++) {
    System.out.print(array[i] + " ");
  }
}
```

```
//Invoke the method
int[] list = {3, 1, 2, 6, 4, 2};
printArray(list);
```

```
//Invoke the method
printArray(new int[ ]{3, 1, 2, 6, 4, 2});
```

Anonymous array

The statement

```
printArray(new int[]{3, 1, 2, 6, 4, 2});
```

creates an array using the following syntax:

```
new dataType[]{literal0, literal1, ..., literalk};
```

There is no explicit reference variable for the array. Such array is called an anonymous array.

Pass By Value and By Reference

- Java uses pass by value to pass parameters to a method. There are important differences between passing a value of variables of primitive data types and passing arrays.
- For a parameter of a primitive type value, the actual value is passed. Changing the value of the local parameter inside the method does not affect the value of the variable outside the method.
- For a parameter of an array type, the value of the parameter contains a reference to an array; this reference is passed to the method. Any changes to the array that occur inside the method body will affect the original array that was passed as the argument.

Simple Example

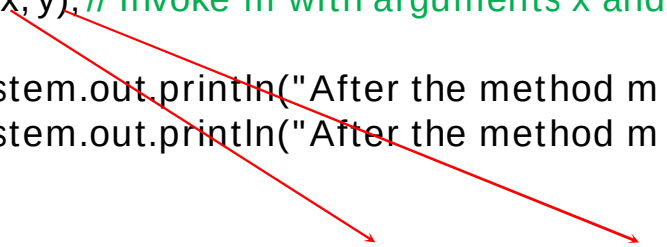
```
public class Test {
  public static void main(String[] args) {
    int x = 1; // x represents an int value
    int[ ] y = new int[10]; // y represents an array of int values
    y[0]=1;

    System.out.println("Before the method m is called, x is " + x);
    System.out.println("Before the method m is called, y[0] is " + y[0]);

    m(x, y); // Invoke m with arguments x and y

    System.out.println("After the method m is called, x is " + x);
    System.out.println("After the method m is called, y[0] is " + y[0]);
  }

  public static void m(int number, int[ ] numbers) {
    number = 2222; // Assign a new value to number
    numbers[0] = 5555; // Assign a new value to numbers[0]
  }
}
```



Question: What are the outputs from the above source code?

Two-dimensional Arrays

```
// Declare array ref var
dataType[ ][ ] refVar;

// Create array and assign its reference to variable
refVar = new dataType[10][10];

// Combine declaration and creation in one statement
dataType[ ][ ] refVar = new dataType[10][10];

// Alternative syntax
dataType refVar[ ][ ] = new dataType[10][10];
```

Declaring Variables of Two-dimensional Arrays and Creating Two-dimensional Arrays

- Declaring two-dimensional array

```
int[ ][ ] matrix = new int[10][10];
double[ ][ ] x;
```

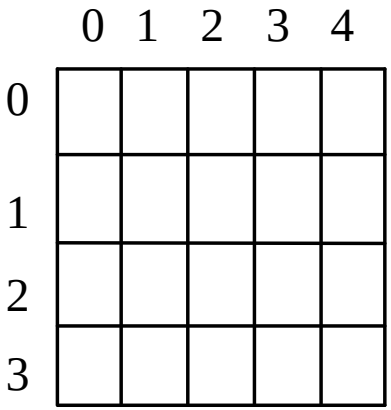
or

```
int matrix[ ][ ] = new int[10][10];
matrix[0][0] = 3;
```

- Initializing two-dimensional array

```
for (int i = 0; i < matrix.length; i++)
for (int j = 0; j < matrix[i].length; j++)
    matrix[i][j] = (int)(Math.random() * 1000);
```

Two-dimensional Array Illustration



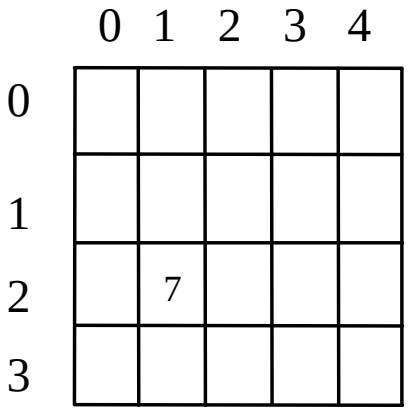

```
matrix = new int[4][5];
```

matrix.length?

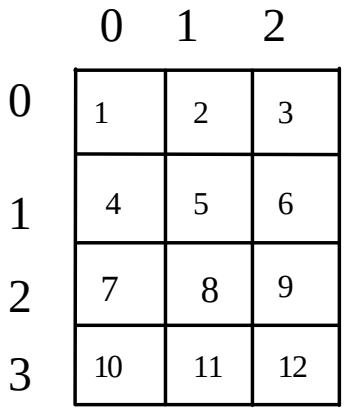
4

matrix[0].length?

5




```
matrix[2][1] = 7;
```




```
int[][] array = {
  {1, 2, 3},
  {4, 5, 6},
  {7, 8, 9},
  {10, 11, 12}
};
```

array.length?

4

array[0].length?

3

Declaring, Creating, and Initializing Using Shorthand Notations

- You can also use an array initializer to declare, create and initialize a two-dimensional array. For example,

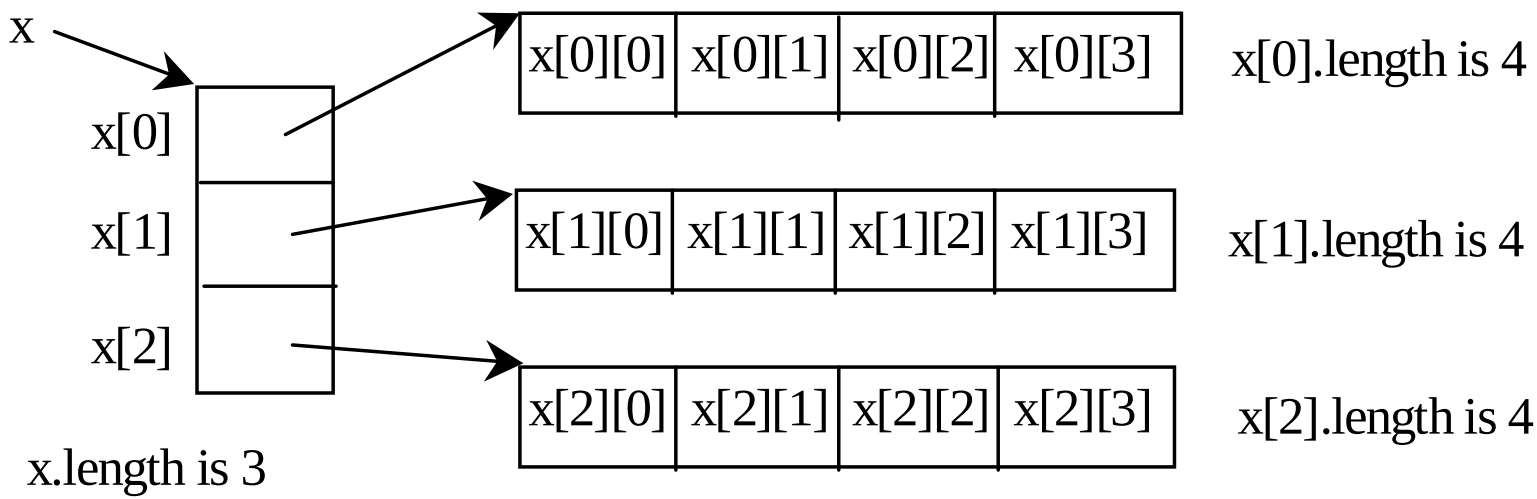
```
int[ ][ ] array = {
  {1, 2, 3},
  {4, 5, 6},
  {7, 8, 9},
  {10, 11, 12}
};
```

Same as

```
int[ ][ ] array = new int[4][3];
array[0][0] = 1; array[0][1] = 2; array[0][2] = 3;
array[1][0] = 4; array[1][1] = 5; array[1][2] = 6;
array[2][0] = 7; array[2][1] = 8; array[2][2] = 9;
array[3][0] = 10; array[3][1] = 11; array[3][2] = 12;
```

Lengths of Two-dimensional Arrays

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



Lengths of Two-dimensional Arrays, cont.

```
int[ ][ ] array = {
  {1, 2, 3},
  {4, 5, 6},
  {7, 8, 9},
  {10, 11, 12}
};
```

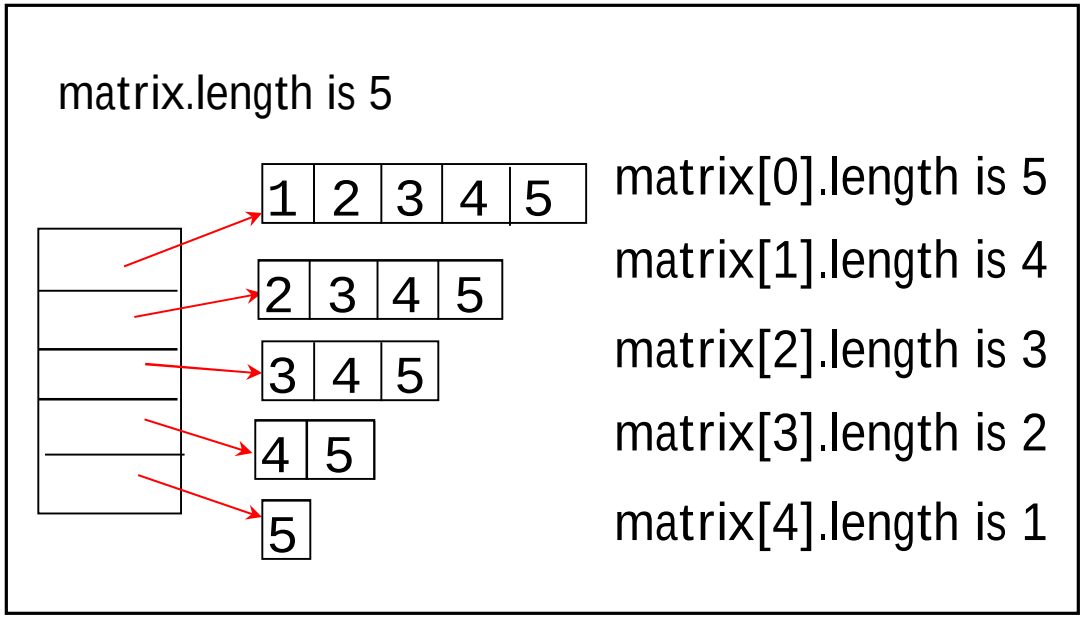
```
array.length
array[0].length
array[1].length
array[2].length
array[3].length
```

`array[4].length` ArrayIndexOutOfBoundsException

Ragged Arrays

- Each row in a two-dimensional array is itself an array. So, the rows can have different lengths. Such an array is known as a ragged array. For example,

```
int[][] matrix = {
  {1, 2, 3, 4, 5},
  {2, 3, 4, 5},
  {3, 4, 5},
  {4, 5},
  {5}
};
```



Multidimensional Arrays

- Occasionally, you will need to represent n-dimensional data structures. In Java, you can create n-dimensional arrays for any integer n.
- The way to declare two-dimensional array variables and create two-dimensional arrays can be generalized to declare n-dimensional array variables and create n-dimensional arrays for n >= 3.
- For example, the following syntax declares a three-dimensional array variable scores, creates an array, and assigns its reference to scores.

```
double[ ][ ][ ] scores = new double[10][5][2];
```

Examples

- Example :

  CalculatorTester.java

- Example: Write a method dayOfWeek(int dayNumber) which takes a day number and returns a String indicating the day of the week (assume that Sunday is day 1). For example, dayOfWeek(3) should return "Tuesday". DayConverterTester.java

Team/League Case Study

Example 6:

1. Consider a League of Teams where different teams play against each other on separate occasions. Each Team should maintain their name, as well as the number of wins, losses and ties that they have had. Make a class called Team with the appropriate get/set/constructor and toString() methods. Write methods called totalPoints() and gamesPlayed() that return the total number of points the team has and the number of games the team has played, respectively. A team gets 2 points for every win and 1 point for every tie.

Team/League Case Study (Cont.)

2. Now let us implement the League class. A League also has a name as well as an array of at most 8 teams. Also, it should keep track of the number of teams that are currently in the league. Again, we'll make appropriate get/set/constructor and toString() methods.
3. in order to add teams to the League, we need an addTeam(Team aTeam) method. We'll also make a showTeams() method that will print out the teams so we can see if things are working.

Team/League Case Study (Cont.)

4. Now all that remains, is to make the functionality for recording wins/losses and ties. To do this, we can make two methods. The first `recordWinAndLoss` method will specify the team that won and the team that lost and will record the appropriate points as well as update the number of games played. The second `recordTie` method will take two teams that tied and update the points and games played as necessary.

/*** Note the use of the get and set methods to make the changes within the Team objects. This is necessary, since we cannot modify the instance variables directly (they are private). Also, we should not even know about how the Team is represented, only how to use the get and set methods as needed. ***/

Team/League Case Study (Cont.)

5. You should now notice something tedious. We would have to make 8 variables if we want to record losses and ties among all teams. Instead, we should make two more recording methods that take the team names as arguments. The methods will then search the league to find the teams with those names and update those teams as needed. To do this, we'll need a method that takes a String as an argument and returns the Team with that name: `teamWithName` method

Team/League Case Study (Cont.)

6. Now we can make two new record methods that take String parameters. Recall that we can use the same method name since JAVA distinguishes between different methods according to their arguments (recall that this is called overloading): `recordWinAndLoss` and `recordTie` methods.
7. Lastly, we should make some interesting methods. We will make a method called `totalGamesPlayed()` which returns the total number of games played in the league. Also, we'll make methods `firstPlaceTeam()` and `lastPlaceTeam()` which returns the team with the most and least points, respectively. If there is a tie pertaining to the total number of points, then we'll chose one of them arbitrarily.
8. Test the program.

References

- Carleton University COMP 1005 and 1006:
  http://www.scs.carleton.ca/~lanthier/teaching/COMP1405/Notes/
  http://www.scs.carleton.ca/~lanthier/teaching/COMP1406/Notes/
- Armstrong Atlantic State University
  http://www.cs.armstrong.edu/liang/intro10e/
- Oracle Java tutorial:
  http://docs.oracle.com/javase/tutorial/
- MIT opencourseware
  https://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-092-introduction-to-programming-in-java-january-iap-2010/