UNIVERSITY OF MISSOURI-COLUMBIA
COLLEGE OF ENGINEERING
DEPARTMENT OF ELECTRICAL ENGINEERING AND COMPUTER SCIENCE

3330 – OBJECT-ORIENTED PROGRAMMING

Laboratory 8: Classes and Objects, and Arrays

Due Date: Friday October 31st, 2025, at 5:00 pm

1 Goal

In Lab3, we introduced the concepts of a class and an object in general, and how to declare and create them in Java. In this lab, we will do more hands-on practice in object-oriented programming because it's one of the most important concepts in Java.

Arrays are used to store multiple values in a single variable, instead of declaring separate variables for each value. In this lab, we will do hands-on practice in the declaration, instantiation, and initialization of one or multiple dimension arrays.

2 Resources

- Lecture notes "Unit 4", "Unit5", "Unit6", BoyNames.txt file, and GirlNames.txt file.

3 Directed Lab Work

3.1 Methods overloading

If methods have the same name, but a different number or type of parameters, they are overloaded. It means you can invoke different methods by the same name by passing different arguments.

As an example, let's consider some overloaded method from the standard class Math:

```
public static int abs(int a) { return (a < 0) ? -a : a; }
public static float abs(float a) { return (a <= 0.0F) ? 0.0F - a : a; }
```

These methods have the same name but different type of the argument. They are overloaded.

Note that, it's impossible to declare more than one method with the same name and parameters (number and types), even with different return types. The return type is not considered for overloading because it's not a part of the signature. Here are three methods print for printing different values:

```
public static void print(String stringToPrint) {
    System.out.println(stringToPrint);
}
public static void print(String stringToPrint, int times) {
    for (int i = 0; i < times; i++) {
        System.out.println(stringToPrint);
    }
}
public static void print(int val) {
    System.out.println(val);
}
```

The first method prints an input string, the second one prints an input string a given number of times and the last one prints an integer value. These methods are overloaded.

Let's invoke these methods:

```
print("some string");
print("another string", 2);
print(5);
```

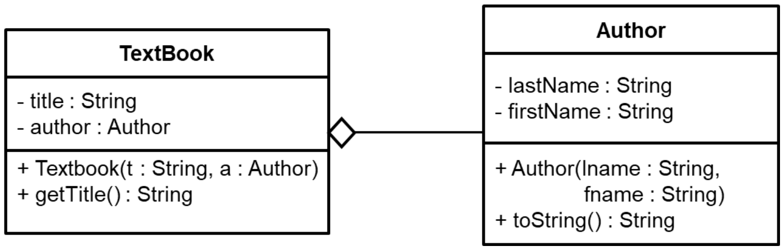
As you can see, it's possible to call any of these methods by the same name passing suitable arguments. The code outputs:

```
some string
another string
another string
5
```

3.2 Aggregation in UML Diagrams

In an object-oriented application it is common for objects of different classes to collaborate. This simply means that objects interact with each other. Sometimes one object will need the services of another object to fulfill its responsibilities. For example, let’s say an object needs to read a number from the keyboard and then format the number to appear as a dollar amount. The object might use the services of a Scanner object to read the number from the keyboard, and then use the services of a DecimalFormat object to format the number. In this example, the object is collaborating with objects created from classes in the Java API. The objects that you create from your own classes can also collaborate with each other.

If one object is to collaborate with another object, then it must know something about the other object’s class methods and how to call them. For example, suppose we were to write a class named Textbook that has the fields **title** and **author**. **title** of type String, however **author** is another class that has two String fields firstName and lastName. In this case we say that the class Textbook needs the class author to define one of its field and we denote this relationship by the aggregation relationships as depicted in the following UML diagram.



3.3 Task 1

- Use IntelliJ IDE to create a new Java project named Task1.
- Create a new Java class and name it **FileSave,** in which write the following:
  - **`Constructor`**: The class’s constructor should take the name of a file as an argument.
  - A method **`save(String line)`** : This method should open the file defined by the constructor, save the string value of **`line`** at the end of the file, and then close the file.

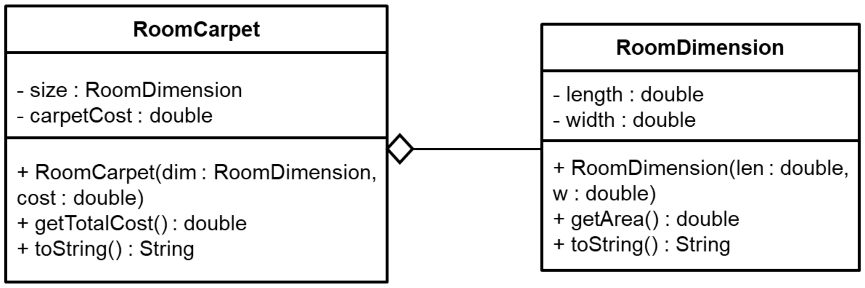
- Create a new Java class and name it **FileDisplay,** in which write the following:
  - **`Constructor`**: The class's constructor should take the name of a file as an argument.
  - A method **`display()`** : This method should check if the file defined by the constructor exits or not, if the file does not exist, it displays the message "The file does not exists" and then exit the program, however if the file exists it displays the file contents line by line into the screen, and then close the file.
  - A method **`display(int n)`** : This method is similar to **`display()`** but it displays the first **`n`** number of lines of the file's contents only. If the file contains less than **`n`** lines, it should display the file's entire contents.
  - A method **`display(int from, int to)`** : This method is similar to **`display()`** but it displays the file's contents starting from line number **`from`** to the line number **`to`**. If the file contains less than **`to`** lines, it should display the file's entire contents. Note that, **`from`** is always less than **`to.`**
- Create another class called FilesDemo, in its main method
  - create an instance of the class FileSave to create new file named "lines.txt"
  - print the following lines into the file:

```
1-Lorem ipsum dolor sit amet
2-Consectetuer adipiscing elit
3-Sed diam nonummy nibh euismod tincidunt
4-Ut wisi enim ad minim veniam
5-Quis nostrud exerci tation ullamcorper
6-Suscipit lobortis nisl ut aliquip ex ea commodo consequat
7-Duis autem vel eum iriure dolor in hendrerit
8-Vel illum dolore eu feugiat nulla facilisis at vero eros
```

  - create an instance of the class FileDisplay to open the "lines.txt"
  - using this instance invoke the method **`display()`**
  - using this instance invoke the method **`display(3)`** and **`display(10)`**.
  - using this instance invoke the method **`display(3, 5)`**.
  - using this instance invoke the method **`display(3, 10)`**.
- You need to throw appropriate exceptions in any methods in the above classes when handling files..
- Run your program and make sure it prints the correct output.
- Don't forget to add comments.

3.4 Task 2

- Consider the following UML class diagram, it shows the class designs and the relationships between the classes.
- Using IntelliJ IDE, create a new Java project called "**Task2**" that has three classes, a **RoomDimension** class, **RoomCarpet** class and a **CarpetCalculator** class.



- First, you should create the class named RoomDimension as depicted in the figure. The `getArea()` method returns the area of the room. (The area of the room is the room's length multiplied by the room's width.)
- The `toString()` of this class should return a string showing the length, width, and the area of the room. For example: "Length: 10.0 Width: 8.0 Area: 80.0"
- Next you should create a RoomCarpet class as depicted in the figure. Note that, the class has a RoomDimension object as a field. `getTotslCost()` method calculates the price, by multiplying the area of the floor (width times length) by the price per square foot of carpet. For example, the area of floor that is 12 feet long and 10 feet wide is 120 square feet. To cover that floor with carpet that costs $8 per square foot would cost $960. (12 * 10 * 8 = 960.)
- The `toString()` of this class should return a string showing the size dimension and the total carpet cost.
- Once you have written these classes, use them in an application in the **CarpetCalculator** class that asks the user to enter the dimensions of a room and the price per square foot of the desired carpeting. The application should display the total cost of the carpet.
  - Sample Run:

```
Enter the price per square foot: 8
Enter the length of the room: 12
Now enter the width of the room: 10
Room dimensions:
Length: 12.0  Width: 10.0  Area: 120.0
Carpet cost: $960.00
```

- Run your program and make sure it prints the correct output.
- Don't forget to add comments.

3.5 Arrays

3.5.1 Introduction to arrays

An array is an ordered collection of elements of the same type. It can store only a fixed number of elements. The length of an array is established when the array is created. Arrays allow you to group and processing similar data together.

indexes	0	1	2	3	4
elements	10.8	14.3	13.5	12.1	9.7

Some features:

- an array is a reference type;
- all elements are stored in the memory sequentially;
- each element of an array is accessed by its numerical index, the first element has the **index 0**;
- the last element is accessed by the index equal to **array size - 1**;
- it's possible to create an array of any types;

- the maximum size of an array is limited by the **Integer**.**MAX_VALUE**;

3.5.2 Declaration, instantiation, initialization

To create an array filled with useful elements we should:

- declare a variable of an array type (**declaration**);
- create an instance of the array object (**instantiation**);
- initialize the array by some values (**initialization**);

To declare an array we must use two special characters `[]` after the name of the type of elements in the array:

```
int[] array; // declaration;
```

or after the name of an array variable:

```
int array[]; // declaration;
```

3.5.3 Creating an array with the specified elements

Java provides several ways to create an array. The simplest way to instantiate and initialize an array is to enumerate all its elements:

```
int[] numbers = { 1, 2, 3, 4 }; // an array of 1, 2, 3, 4;
```

But, this way to create an array has some disadvantages: we need to know values and number of elements at the moment of compilation.

Another way is to initialize an array using variables

```
int a = ..., b = ..., c = ...;
int[] numbers = { a, b, c };
```

In this case, we should have all elements at the moment of the array creation.

3.5.4 Creating an array using the keyword "new"

The most general way to create an instance (object) of an array is to use the special keyword new and specify the necessary number of elements:

```
int n = ...; // n is a length of an array
int[] numbers = new int[n];
```

Now, the array has n elements. Each element is equal to 0 (the default value). Next, we should make explicit initialization of elements.

It's possible to separate declaration and instantiation in two lines:

```
int[] numbers;
numbers = new int[n];
```

Also, we can write the keyword new and enumerate all elements of an array:

```
float[] floatNumbers; // declaration
floatNumbers = new float[] { 1.02f, 0.03f, 4f } // instantiation and initialization;
```

3.5.5 The length of an array

To obtain the length of an existing array we should write arrayName.length. It returns the length of the array. Here is an example:

```
int[] array = { 1, 2, 3, 4 }; // an array of numbers
int length = array.length; // number of elements of the array
System.out.println(length); // 4
```

3.5.6 Accessing elements

The values of elements of an array can be changed. To set (get) a value to (from) array the index is used.

Set the value by the index:

```
array[index] = val;
```

Get the value by the index:

```
val = array[index];
```

Indexes of an array have numbers from **`0`** to **`length - 1`** inclusive. Let's see an example.

```
int numbers[] = new int[3]; // numbers: [0, 0, 0]
numbers[0] = 1; // numbers: [1, 0, 0]
numbers[1] = 2; // numbers: [1, 2, 0]
numbers[2] = numbers[0] + numbers[1]; // numbers: [1, 2, 3]
```

Some explanations:

- in the first line, the integer array named numbers with the size 3 is created. By default, all elements are equal to 0;
- in the second line, the value "1" is assigned to the first element of the array by its index (do not forget, the first element has the index 0);
- in the third line, the value "2" is assigned to the second element of the array by its index (numbers[1] - is the second element);
- in the last line, the sum of first two elements is assigned to the third element by its index.

Note, if we try to access a non-existing element by an index then a runtime exception happens. For instance, let's try to get fourth element (with index 3) of the considered array numbers.

```
int elem = numbers[3];
```

The program throws **ArrayIndexOutOfBoundsException**. Be careful with indexes when accessing elements of an array.

3.5.7 The utility class Arrays

The class provides a lot of useful methods for processing arrays.

- convert array to string using **Arrays.toString(...)** and then print it:

```
byte[] famousNumbers = { 0, 1, 2, 4, 8, 16, 32, 64 };
System.out.println(Arrays.toString(famousNumbers)); //it prints [0, 1, 2, 4, 8, 16, 32, 64]
```

- sorting a whole array or a part of it:

```
long[] bigNumbers = {20000000000L,40000000000L,10000000000L,30000000000L}; //it's unsorted
Arrays.sort(bigNumbers); // sorting whole array
// it prints [10000000000, 20000000000, 30000000000, 40000000000]
System.out.println(Arrays.toString(bigNumbers));
```

- comparing arrays: two arrays are equal if they contain the same elements in the same order:

```
int[] numbers1 = { 1, 2, 5, 8 };
int[] numbers2 = { 1, 2, 5 };
int[] numbers3 = { 1, 2, 5, 8 };
System.out.println(Arrays.equals(numbers1, numbers2)); // it prints "false"
System.out.println(Arrays.equals(numbers1, numbers3)); // it prints "true"
```

- filling a whole array or a part of it by some values:

```
int size = 10;
char[] characters = new char[size];
//It takes an array, start index, end index (exclusive) and the value for filling the array
Arrays.fill(characters, 0, size / 2, 'A');
Arrays.fill(characters, size / 2, size, 'B');
System.out.println(Arrays.toString(characters)); //it prints [A, A, A, A, A, B, B, B, B, B]
```

3.5.8 Processing arrays using loops

Often, it's needed to perform some kind of algorithms on the elements of an array. For instances: sort them, find the maximum element, print only positive numbers, reverse the order, calculate the arithmetic average of numbers and so on.

A convenient way to process an array is to iterate over the array using a loop. The property length of an array can help us to avoid **ArrayIndexOutOfBoundsException**.

Example 1. Filling an array with the squares of indexes.

```
int n = 10; // the size of an array
int[] squares = new int[n]; // creating an array with the specified size
System.out.println(Arrays.toString(squares)); // [0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
/* iterating over the array */
for (int i = 0; i < squares.length; i++) {
    squares[i] = i * i; // set the value by the element index
}
System.out.println(Arrays.toString(squares)); // [0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

In the code above an array with the size 10 is created (filled with 0). Then each element of the array is set to the square of the element's index. Then the array is output to the standard output.

3.5.9 Reading an array from the standard input

Using a loop we can read all elements of an array from the standard input. For example, the input consists of two lines. The first line contains the length of an array, the second line - all elements of the array.

```
5
101 102 504 302 881
```

```
import java.util.Scanner;
import java.util.Arrays;
public class ReadingArrayExample {
    public static void main(String args[]) {
        Scanner scanner = new Scanner(System.in);
        int len = scanner.nextInt(); // reading a length
        int[] array = new int[len];  // creating an array with the specified length

        for (int i = 0; i < len; i++) {
            array[i] = scanner.nextInt(); // read the next number of the array
        }
        System.out.println(Arrays.toString(array)); // output the array
    }
}
```

The program outputs:

```
[101, 102, 504, 302, 881]
```

3.5.10 Iterating over arrays using the for-each loop

Since Java 5 there is a special kind of the for loop called for-each. It is a special kind of the for-loop that iterates arrays and collections without using indexes.

Let's write a code for calculating the number of 'a' letter in the given character array. To iterate over the array we'll use for-each loop.

```
char[] characters = { 'a', 'b', 'c', 'a', 'b', 'c', 'a' };
int counter = 0;
for (char ch : characters) {
    if (ch == 'a') {
        counter++;
    }
}
System.out.println(counter); // it outputs "3"
```

3.6 Task 3

- Use IntelliJ IDE to create a new Java project named Task3.
- Create a new Java class and name it **SequenceOperations.** In this class, write the following static methods:
  - `getTotal`: This method should accept an array of double numbers as its argument and return the total of the values in the array.
  - `getAverage`: This method should accept an array of double numbers as its argument and return the average of the values in the array.
  - `getHighest`: This method should accept an array of double numbers as its argument and return the highest value in the array.
  - `getReverse`: This method should accept an array of double numbers as its argument and return an array of these values in the reverse order. For example, if the given array is 3.0, 2.4, 6.0 then it should return an array of 6.0, 2.4, 3.0.
- Create another class called **Task3Demo.** The main method of this class creates a double array of the following values 3.0, 2.4, 6.0, 2.0, 4.0, 5.1, 7.2. Then it should use the above static methods to display on the screen the following information:
  - The total number of the values in the array.
  - The average number of the values in the array.
  - The highest number in the array
  - The array values in the reverse order.
- Sample output:

```
Processing the array: [3.0, 2.4, 6.0, 2.0, 4.0, 5.1, 7.2]
Total: 29.7
Average: 4.24
Highest value: 7.2
Array Reverse: [7.2, 5.1, 4.0, 2.0, 6.0, 2.4, 3.0]
```

- Don’t forget to add comments.

3.7 Task 4

KidsPopularName
- kidsNames : String[] - fileName : String
+ KidsPopularName(String fileName) - getNumNames(): int - fillNames(): void + isPopularName(name: String): boolean

- Use IntelliJ IDE to create a new Java project named Task4.
- Create a new Java class called KidsPopularName as described in the following UML diagram.
- In this class:
  - o The **constructor** takes the file name as an argument, initialize the filename field and initialize the **kidsNames** array by the items in this file. The constructor uses the helper method **`getNumNames()`** to determine the size (total number of items) of the file, and the method **`fillNames()`** to read the names from the file line by line and assign these names to the array.
  - o **`isPopularName`**: This method should accept a name of type String as its argument, search for the name within the array and return **true** if the name exists in the **kidsNames** array or **false** otherwise. You may need to use the String class method **`equalsIgnoreCase()`**.
- Create another class called **NameSearch.** The main method of this class:
  - o Creates two instance of the class KidsPopularName, called boyNames and girlNames. The constructor's argument of boyNames is "BoyNames.txt" and the constructor's argument of boyNames is "GirlNames.txt". These two files are given in Canvas with this lab link.
  - o Ask the user to enter a kid name, using the **isPopularName**() method determine whether this name is boy popular name and/or girl popular name.
  - o If the name boy popular name it displays, it is one of the most popular boy's names.
  - o If the name girl popular name it displays, it is one of the most popular girl's names.
  - o If the name is not boy or girl popular name, it displays, it is not one of the most popular kid's names.
  - o You need to use try-catch blocks and/or throw proper exceptions in the above classes when handling files.
- Don't forget to add comments.

3.8 Task 5

- In this task you will be writing the quiz part of this lab. Use the Quizzes tab on Canvas to complete and submit answer for this quiz.
- The quiz includes one programming question to evaluate your understanding of this lab material. You will be given **45 minutes** to complete and submit your answer, please follow the following important instructions.
  - o You are to complete this quiz independently and alone. Collaborating, discussing, or sharing of information during the quiz is not permitted.
  - o Use the IntelliJ Idea IDE to create a new Java project file and named it yourUMId_Lab8_Quiz. yourUMId is the first part of your UM (University of Missouri) email address before the @ sign.
  - o The quiz link will be inactive on **October 31 at 5:00 PM**. Note that, it is allowed to submit only once, so please review your code carefully before submitting your answer.
  - o To start, click Final Assessment link. Once you click "Begin Assessment," you will have 30 minutes to complete this quiz.

- Read the question and start to write your codes using the opened project file in the IntelliJ IDE.
- Once you finished writing your program, export your project into yourUMId_Lab8_Quiz.zip file.
- To submit your answers, click the Browse button, navigate the file explorer to your exported zip file, click Upload button, and then click Submit for Grading button.

4 Hand In

- Create a new folder and name it yourUMId_Lab8, then copy the folders of projects from Task1 to Task4 in your new folder yourUMId_Lab8. yourUMId is the first part of your UM (University of Missouri) email address before the @ sign Zip yourUMId_Lab8 folder into yourUMId_Lab8.zip.
- **Please note that the naming conventions, the types, and the submitted file structure are very important. The improper file structures, types or names will cause losing marks.**
- Submit your *yourUMId*_ Lab8.zip file into Canvas on **October 31st, before 5:00 PM**.
- **No submissions will be accepted after October 31st, 5:00 PM.**