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

3330 – OBJECT-ORIENTED PROGRAMMING

Laboratory 4: Classes and Objects – Part1

Due Date: Friday September 26th , 2025, at 5:00 pm

(No late submissions will be accepted for this Lab)

1 Goal

In this lab you will know what a class and an object is in general and how to declare and create them in Java.

2 Resources

- Lecture notes "Unit 2"

3 Directed Lab Work

3.1 Fundamentals of object-oriented programming

Object-oriented programming (OOP) is a programming paradigm based on the concepts of "objects", which may be characterized by a state and a behavior. Objects contain a state (data) in the fields (also known as attributes) and a behavior in the methods. In object-oriented programs, a set of objects interacts with each other to perform the program functions.

In an object-oriented program real-world (or not absolutely real) objects are modeled. For these objects, only essential characteristics should be selected. It's known as **data abstraction** — a fundamental principle of OOP.

Hiding internal state of objects and requiring all interaction to be performed through methods is known as **data encapsulation** — another fundamental principle of OOP.

Objects - Here are a lot of real-world objects around you: pets, buildings, cars, computers, planes and so on. Even a computer program may be considered as an object.

Classes - In the real world, you will often find many individual objects that have similar characteristics. These objects may belong to the same type (or class). A class describes a structure of similar objects: fields and methods.

We can even say that a class is a blueprint from which individual objects (instances) are created. Let's look at the following examples.

Example 1: This picture shows an abstract building for describing buildings as a class (type) such that:

- It may have some attributes:
  - a number of floors (an integer number);
  - an occupied area (a floating-point number, square meters);
  - a year of construction (an integer number);
- Each object (instance) of the building type has the same attributes but different values. For instance, we have two buildings (objects):
  - Building 1: the number of floors = 4, the occupied area = 2400.16, the year of construction = 1966;
  - Building 2: the number of floors = 6, the occupied area = 3200.54, the year of construction = 2001;
- It's quite difficult to find a behavior of a building. But this example shows attributes well.

Example 2: The second picture shows an abstract plane for describing all planes as a class (type) such that:

- It may have some attributes:
  - family name (a string, for example, "Airbus A320" or "Boeing 777");
  - passengers' capacity (an integer number);
  - standard speed (an integer number);
  - current coordinates (it's needed to fly);
  - and so on.
- Also, it has a behavior (a method): transfer from one geographical point to another.
- Objects of the plane class have the same attributes and a method transfer. But, the values of attributes and behavior, depending on the object state, may be different.

3.1.1 Declaring classes

In Java, when writing an object-oriented program, first, we declare classes (or find the suitable classes in libraries) and then we create instances (objects) of these classes. You have already known at least one class from the Java Class Library. It's a widely used class String. Now it's time to declare your own classes.

In terms of Java, classes are a mechanism used to group data (fields) and methods together into coherent modules. It makes your programs more structured and maintainable.

The source code of classes is located in **.java** files. Often the file contains a single class and has the name **yourClassName.java**. But it's not always so. After the compilation processes classes are located in **.class** files that contain bytecode.

To declare a class Java provides a keyword **class**. Let's declare the class Nothing consisting of nothing:

```
class Nothing {
  // there is nothing
}
```

The class doesn't contain any attributes (fields), but we can create an instance of the class.

3.1.2 Instantiating objects

To create an instance the keyword **new** is used:

```
Nothing nothing = new Nothing();
```

Perhaps, creating an instance of the class Nothing sounds like a kind of magic. But soon everything will be clear.

Pay attention a class's name starts with a capital letter, according to the naming convention. If a class has a name with multiple words, each word starts with a capital letter. For example:

MyCustomClass
FastHttpConnector
DemoApplication

Also, we can create an instance of any standard class using the keyword **new**.

```
Date now = new Date(); // it contains "now" date
```

There is no difference between instantiating your and standard classes in Java.

3.1.3 Class members

A Java class can contain fields, methods, and constructors.

- Fields are variables that store data.
- Methods are operations that the class or instances can perform.
- Constructors are special kind of methods that initialize instances of the class. They must match the class name.

Fields and methods are often considered as a class's members, constructors are not always, but they are a part of a class.

Not all Java classes have fields, constructors, and methods. Sometimes you have classes that only contain fields (data), and sometimes you have classes that only contain methods (operations). It depends on what the Java class is supposed to do.

Example. Let's consider a more complex example (Widget Factory).

There is a class for a widget manufacturing plant. The class have a method whose argument is the number of widgets that must be produced. It has another method that calculates how many days it will take to produce the number of widgets. This factory produces 10 widgets each hour. It operates two shifts of eight hours each per day.

```
/**
 * The WidgetFactory class stores data for a widget manufacturing plant.
 */
public class WidgetFactory {
   private final double WIDGETS_PER_HOUR = 10.0; // Number of widgets per hour
   private final int NUM_SHIFTS = 2;             // Number of shifts
   private final int HOURS_PER_SHIFT = 8;        // Hours per shift
   private int numWidgets;                       // To hold the number of widgets

   /**
    * The setNumWidgets method sets the number of widgets.
    */
   public void setNumWidgets(int n) {
      numWidgets = n;
   }

   /**
    * The getNumWidgets method returns the number of widgets.
    */
   public int getNumWidgets() {
      return numWidgets;
   }

   /**
    * The getDaysToProduce method returns the number of days required
    * to produce the widgets.
    */
   public double getDaysToProduce() {
      // Calculate the number of widgets produced per day.
      double widgetsPerDay = WIDGETS_PER_HOUR * NUM_SHIFTS * HOURS_PER_SHIFT;

      // Return the number of days required to produce
      // the specified number of widgets.
      return numWidgets / widgetsPerDay;
   }
}
```

To better understand the members of the class WidgetFactory read the provided code's comments.

3.1.4 Invoking instance methods

We can create an instance of a class and invoke its methods. See the following example where an instance of the class WidgetFactory named "wf" is created.

```
/**
 * This program demonstrates the WidgetFactory class.
 */
import java.util.Scanner;
public class WidgetDemo
{
   public static void main(String[] args)
   {
         int noOfWidgets;       // To hold the number of widgets

         // Create a WidgetFactory object.
         WidgetFactory wf = new WidgetFactory();

         // Create a Scanner object for keyboard input.
         Scanner keyboard = new Scanner(System.in);

         // Get the number of widgets.
         System.out.print("Enter the number of widgets: ");
         noOfWidgets = keyboard.nextInt();

         // We set the given number of widgets.
         wf.setNumWidgets(noOfWidgets);

         // Display the number of days required to
         // produce that many widgets.
         System.out.println("It will take " +
                 wf.getDaysToProduce() +
                 " days to produce " + noOfWidgets + " widgets.");
    }
}
```

In this example, if we entered the value 1500 for the number of widgets the program will display the following message:

```
Enter the number of widgets: 1500
It will take 9.375 days to produce 1500 widgets.
```

3.2 Task 1

- Use IntelliJ IDE to create a new Java project named Task1.
- Create a new Java class and name it **Cylinder**.
- In this class add the following fields:
    - radius: a double
    - height: a double
    - PI: a final double initialized with the value 3.14159
- The class should have the following methods:
    - Constructor. Accepts the radius and the height of the cylinder as two arguments.
    - **setRadius**. A mutator method for the radius field.
    - **setHeight**. A mutator method for the height field.
    - **getRadius**. An accessor method for the radius field.

- **getHeight**. An accessor method for the height field.
- **getVolume**. Returns the volume of the cylinder, which is calculated as
  volume = PI * radius * radius * height
- **getCurvedSurfaceArea**. Returns the area formed by the curved surface of the cylinder i.e. space occupied between the two parallel circular bases, which is calculated as
  curved surface area = 2 * PI * radius * height
- **get TotalSurfaceArea**. Returns the total area that the cylinder occupies, which is calculated as
  total surface area = 2 * PI * radius * (height + radius)
- Now you need to write a driver program that demonstrates the Cylinder class. Right-Click the **src** folder and choose new→Java class, name this class **CylinderDemo**.

In this CylinderDemo class (the driver program) write the main method that will ask the user for the cylinder's radius and the height creating a Cylinder object, and then reporting the cylinder's volume, curved surface area, and total surface area.

- The program output should be something similiter to the following:

```
Enter the radius of a cylinder: 3
Enter the height of a cylinder: 9
The cylinder's radius is 3.0
The cylinder's height is 9.0
The cylinder's volume is 254.46878999999996
The cylinder's curved surface area is 169.64585999999997
The cylinder's total surface area is 226.19447999999997
```

- Run your program and make sure it prints the correct output. Do not forget to add proper comments to make your program self-documenting.

3.3 Methods

A method is a sequence of statements grouped together to perform an operation. In Java, a method is always located inside a class. The relation between methods and classes will be learned further. In this Lab, you will learn how to define new methods. It is assumed you've already known how to invoke existing methods.

3.3.1 The base syntax of methods

In general case, a method has the following six components:

1. a name;
2. a set of modifiers (public, static, etc)
3. a type of the return value;
4. a list of parameters (as well known as formal parameters) in parenthesis ();
5. a list of exceptions;
6. a body containing statements to perform the operation.

Some of these components are always required and others are optional. Now, we will focus on 1, 3, 4 and 6 components.

3.3.2 Defining a simple method

Here is an example of a simple method that calculates the sum of two given numbers:

```
public static int sum(int a, int b) {
    return a + b;
}
```

The `sum` is a typical method written in Java. It returns the sum of two its parameters (arguments). The parameters are written in the parenthesis "(...)". To return the integer result the keyword `return` is written.

In general case, a returning value and parameters can have any type, including non-primitive types. Also, the method has two modifiers: `public` and `static`.

The combination of the name of a method and its parameters is called the **signature**. It doesn't include:

- the returning type
- modifiers
- names of parameters

The considered method `sum` has the signature `sum(int, int).`

3.3.3 Naming methods

There are two kinds of restrictions for the name of a method: the compiler (required) and the naming convention (optional, but desired).

The Java compiler requires that a method name can be a legal identifier. The rules for legal identifiers are the following:

- identifiers are case-sensitive;
- an identifier can include Unicode letters, digits, and two special characters ($,_);
- an identifier can't start with a digit;
- identifiers must not be a keyword.

In addition, there is a naming convention that restricts possible method names. It's optional but desired for developers.

By the convention, method names should be a verb in lowercase or multi-word name that begins with a verb in lowercase, followed by adjectives, nouns, etc. In multi-word names, the first letter of the second and the following words should be capitalized. Here are some correct examples:

```
sum
getValue
calculateNumberOfOranges
findUserByName
printArray
```

3.3.4 The type of a returning value and parameters

A method can return a single value or nothing. To declare a method returns nothing you should write the special keyword `void` as the type of a result value.

The following method prints the sum of two given numbers and returns no value.

```
public static void printSum(int a, int b) {
    System.out.println(a + b);
}
```

A method can take one or multiple parameters of the same or different types. Also, it's possible to declare a method without any parameters, but "()" are still required.

```
// The method has an int parameter
public static void method1(int a) {
    // do something
}

// The method has long and double parameters
public static void method2(long a, double b) {
    // do something
}

// The method has no any parameters and returns a value of int type
public static int method3() {
    return 3;
}
```

When you call a method with a value of a primitive type then a copy of the value is created. Inside a method, you can process this copy. If you change it, the passed argument is not changed.

```
public static void main(String[] args) {
    int val = 100; // 100

    change(val); // try to change val
    System.out.println(val);
    // it prints "100", because the method changed a copy
    // of the val not the val itself
}

// The method changes a given value
public static void change(int val) {
    val = 400; // now, the copy is 400
}
```

As you can see, the method changed a copy of the given integer value 100.

3.4 Task 2

- Using IntelliJ IDE, create a new Java project called "**Task2**" that has two classes, a **Temperature** class and a **TemperatureDemo** class.
- The Temperature class will hold a temperature in Fahrenheit and provide methods to get the temperature in Fahrenheit, Celsius, and Kelvin. The class should have the following field:
  - **fTemp** —A double that holds a Fahrenheit temperature.
- The class should also have the following methods:

- Constructor—The constructor accepts a Fahrenheit temperature (as a double) and stores it in the fTemp field.
- **setFahrenheit** —The setFahrenheit method accepts a Fahrenheit temperature (as a double) and stores it in the fTemp field.
- **getFahrenheit** —Returns the value of the fTemp field, as a Fahrenheit temperature (no conversion required).
- **getCelsius** —Returns the value of the fTemp field converted to Celsius.
- **getKelvin** —Returns the value of the fTemp field converted to Kelvin.

- Use the following formula to convert the Fahrenheit temperature to Celsius:
  `Celsius = (5/9) x (Fahrenheit - 32)`
- Use the following formula to convert the Fahrenheit temperature to Kelvin:
  `Kelvin = ((5/9) x (Fahrenheit - 32)) + 273`

- The TemperatureDemo class will demonstrate the driver program that asks the user for a Fahrenheit temperature. The program should create an instance of the Temperature class, with the value entered by the user passed to the constructor. The program should then call the object's methods to display the temperature in Celsius and Kelvin
- The program output should be something similiter to the following:

```
Enter the Fahrenheit temperature: 212
Celsius: 100.00
Kelvin: 373.00
```

- Run your program and make sure it prints the correct output. Do not forget to add proper comments to make your program self-documenting.

3.5 Task 3

- 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 **30 minutes** to complete and submit your answer, please follow the following important instructions.
  - You are to complete this quiz independently and alone. Collaborating, discussing, or sharing of information during the quiz is not permitted.
  - Use the IntelliJ Idea IDE to create a new Java project file and named it yourUMId_Lab4_Quiz. yourUMId is the first part of your UM (University of Missouri) email address before the @ sign.
  - The quiz link will be inactive on **September 26 at 5:00 PM**. Note that, it is allowed to submit only once, so please review your code carefully before submitting your answer.
  - 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_Lab4_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_Lab4, then copy the folders of projects Task1 and Task2 in your new folder yourUMId_Lab4. yourUMId is the first part of your UM (University of Missouri) email address before the @ sign Zip yourUMId_Lab4 folder into yourUMId_Lab4.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_* Lab4.zip file into Canvas on **September 26, before 5:00 PM**.
- **No submissions will be accepted after September 26, 5:00 PM**