

At this lab section, we will learn fundamentals of Object-Oriented Programming: Classes, interfaces, and inheritance.

# Fundamentals of

Object Oriented Programming

Asst. Prof. Dr. Feriştah DALKILIÇ Res. Asst. Fatih DİCLE

## PART 1 –Inheritance

**Exercise 1.** At this section, we will examine inheritance concept. Our first class is the main class *Test.java,* the second class is the super class *Point.java* and the third class is *Circle.java* that is inherited from *Point.java.*

Step – 1

Create a new Java Project. Add the java files *Test.java, Point.java* and *Circle.java.*

Test. java

**public class** Test {

**public static void** main(String[] args) {

System.***out***.println("---------------------------------------");

Point p = **new** Point(0, 0);

System.***out***.println("---------------------------------------");

Circle c = **new** Circle(2, 4);

System.***out***.println("---------------------------------------");

Circle c1 = **new** Circle();

System.***out***.println("---------------------------------------");

p.Display(); System.***out***.println();

System.***out***.println("---------------------------------------");

c.Display();

System.***out***.println("---------------------------------------");

c1.Display();

System.***out***.println("---------------------------------------");

System.***out***.println(c.Calculate(**new** Circle(9, 5)));}

}

Point. java

**public class** Point {

**private int** x;

**public int** y;

**public** Point(**int** \_x, **int \_**y)

{

x = \_x;

**this**.y = \_y;

System.***out***.println("Point constructor with parameter.");

}

**protected int** GetX()

{

**return this**.x;

}

**public void** SetX(**int** x)

{

**this**.x = x;

}

**public void** Display()

{

System.***out***.print("X = " + x + " Y = " + y);

}

}

Circle. java

**public class** Circle **extends** Point

{

**private int** r;

**public** Circle()

{

**super**(0, 0);

r = 0;

System.***out***.println("Circle constructor.");

}

**public** Circle(**int** r, **int** x)

{

**super**(x, 0);

**this**.r = r;

System.***out***.println("Circle constructor with parameter.");

}

**public double** Calculate(Circle c)

{

**return** GetX() + c.GetX() + **super**.y + c.y;

}

**public void** Display()

{

System.***out***.println("\nCircle: "); **super**.Display(); System.***out***.println(" Z = " + r);

}

}

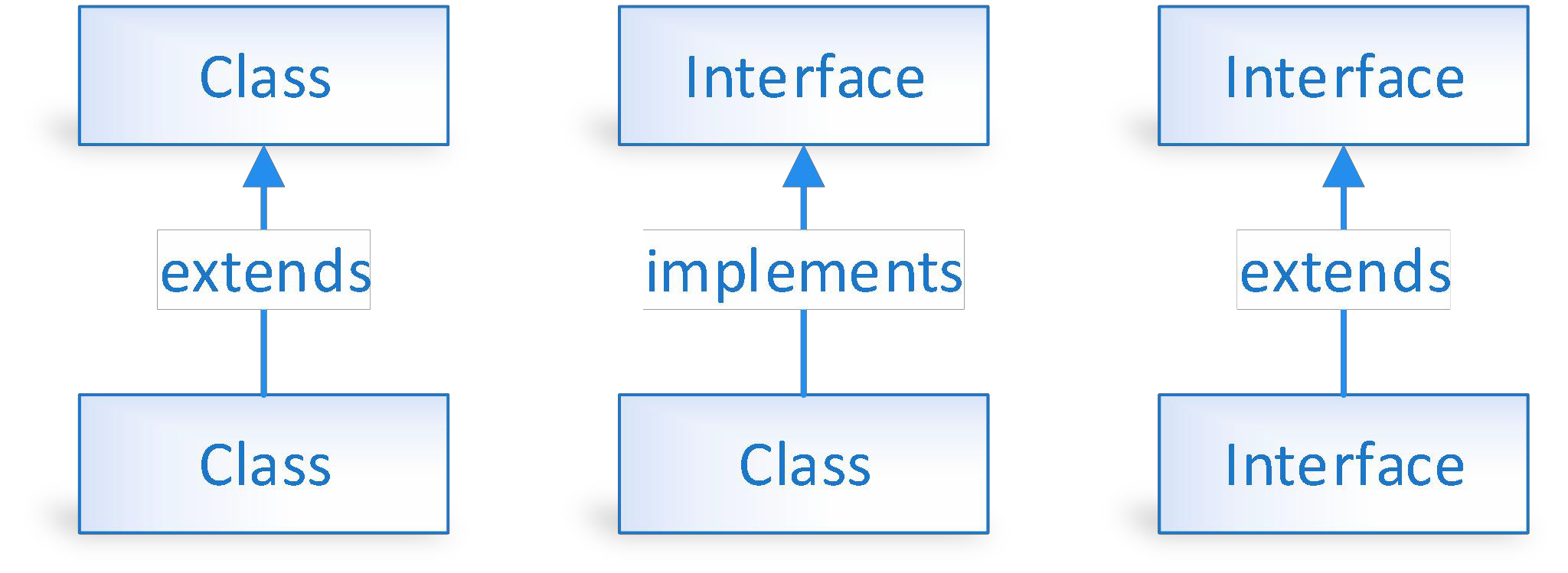
Step – 2

Paste the output of the Test.java.

|  |
| --- |
|  |

## PART 2 –Interfaces

* An interface, sometimes also called an abstract data type, defines the set of operations supported by a data structure. It tells nothing about how the data structure implements these operations.
* A data structure implementation, on the other hand, includes the internal representation of the data structure.
* There can be many implementations of a single interface. At the same time, a class can implement more than one interface.



An interface is different from a class in several ways:

* You cannot instantiate an interface.
* An interface does not contain any constructors.
* All the methods in an interface are abstract.
* An interface cannot contain instance fields. The only fields that can appear in an interface must be declared both static and final.
* An interface is not extended by a class; it is implemented by a class.
* An interface can extend multiple interfaces.

Why do we use interfaces?

* It is used to achieve total abstraction.
* Since java does not support multiple inheritance in case of class, but by using interface it can achieve multiple inheritance.
* It is also used to achieve loose coupling.
* Interfaces are used to implement abstraction. So, the question arises why use interfaces when we have abstract classes?
* The reason is, abstract classes may contain non-final variables, whereas variables in interface are final, public, and static.

**Exercise 2.** At this section, we will implement Shape interface. Circle, Rectangle, and Triangle are the three different implementations of the same interface.

Step – 1

Create a new Java Project. Add the java files *Shape.java, Circle.java*, *Rectangle.java*, and *Triangle.java.*

|  |
| --- |
| **public interface** Shape { **public** String type(); **public double** area();  **public double** perimeter();  }  **public class** Circle **implements** Shape {  **private double** radius;  **public** Circle(**double** radius) {  **this**.radius = radius;  }  **public double** area() {  **return** Math.***PI*** \* radius \* radius;  }  **public double** perimeter() {  **return** 2.0 \* Math.***PI*** \* radius;  }  @Override  **public** String type() {  **return** "Circle";  }  }  **public class** Rectangle **implements** Shape {  **private double** width;  **private double** height;  **public** Rectangle(**double** width, **double** height) {  **this**.width = width;  **this**.height = height;  }  **public double** area() {  **return** width \* height;  }  **public double** perimeter() {  **return** 2.0 \* (width + height);  }  @Override  **public** String type() {  **return** "Rectangle";  }  }  **public class** Triangle **implements** Shape {  **private double** a; **private double** b; **private double** c;  **public** Triangle(**double** a, **double** b, **double** c) {  **this**.a = a; **this**.b = b; **this**.c = c;  }  **public double** area() {  **double** s = (a + b + c) / 2.0;  **return** Math.*sqrt*(s \* (s - a) \* (s - b) \* (s - c));  }  **public double** perimeter() {  **return** a + b + c;  }  @Override  **public** String type() {  **return** "Triangle";  }  } |

Step – 2

Add *Test.java* into your project. You are expected to create instances of all 3 subclasses, store these objects in a common array with the type of base interface, and print objects’ information by iterating the array. Change the highlighted lines with your own.

|  |
| --- |
| **public class** Test {  **public static void** main(String[] args) {  //Create a circle, a rectangle, and a triangle object  //Add all objects into a single array  //print objects’ type, area and perimeter info by looping the array.  }  } |

Step – 3

Paste the output of the Test.java.

|  |
| --- |
| Your Code |
|  |

## PART 3 –Abstract Classes

There are two ways to achieve abstraction in Java,

1. Interface (100%)
2. Abstract class (0 to 100%)

An abstract class,

* must be declared with an abstract keyword.
* It can have abstract and non-abstract methods.
* It cannot be instantiated.
* It can have constructors and static methods also.
* It can have final methods which will force the subclass not to change the body of the method.

**Abstract Class vs Interface**

* A class can inherit from only one abstract class, but it can implement multiple interfaces. This is because an abstract class represents a type of object, while an interface represents a set of behaviors.
* Abstract classes can have access modifiers such as public, protected, and private for their methods and properties, while interfaces can only have public access.
* An abstract class can have member variables, while an interface cannot.

**Exercise 3.** You are given an abstract class named Course.

|  |
| --- |
| Course.java |
| **public** **abstract** **class** Course {    **public** String code;    **public** Course(String courseCode) {  code = courseCode;  System.***out***.println("Opening the course " + courseCode);  }  **public** **abstract** **void** Syllabus();  **public** **void** Learn(){  System.***out***.println("Preparing the content of " + **this**.code);  }  } |

Step – 1

Derive two different classes with names *DataStructures.java* (CME2201) and *OperatingSystems.java* (CME3205) from the class *Course*.

Step – 2

In Test.java create instances of classes *DataStructures* and *OperatingSystems*. Call the methods Syllabus() and Learn().

|  |
| --- |
| Your Code (*DataStructures.java, OperatingSystems.java, and* Test.java) |
|  |

|  |
| --- |
| Your Output |
|  |

**Submission**

* Upload your answer sheets in PDF format by naming it as studentID\_Name\_FamilyName.pdf.
* DO NOT USE any Turkish character or blank in naming the file.