Jordan University
Department of Computer Engineering

Object-Oriented Problem Solving: CPE 342

Lab-1 Tutorial: Getting Started + Fundamentals

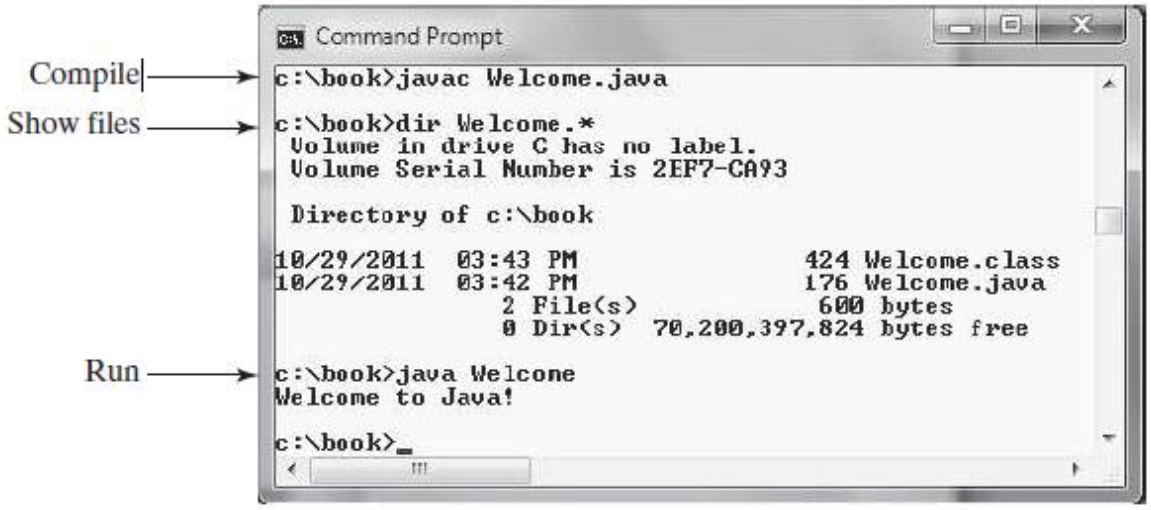
Eng. Asma Abdel Karim

Tutorial contents:

❖ Part I: Compiling and Running without an IDE (1.8 p.16&17).

1. Create your Java source file using a text editor (e.g. Windows Notepad).

2. Write commands in the command prompt to compile and run your program.



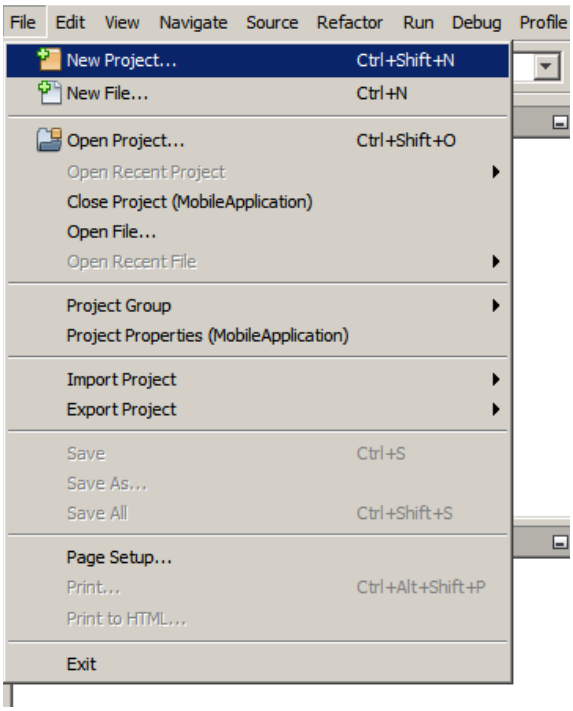
❖ Part II: Developing Java Programs Using NetBeans.

Note: similar content is available in the textbook section 1.11

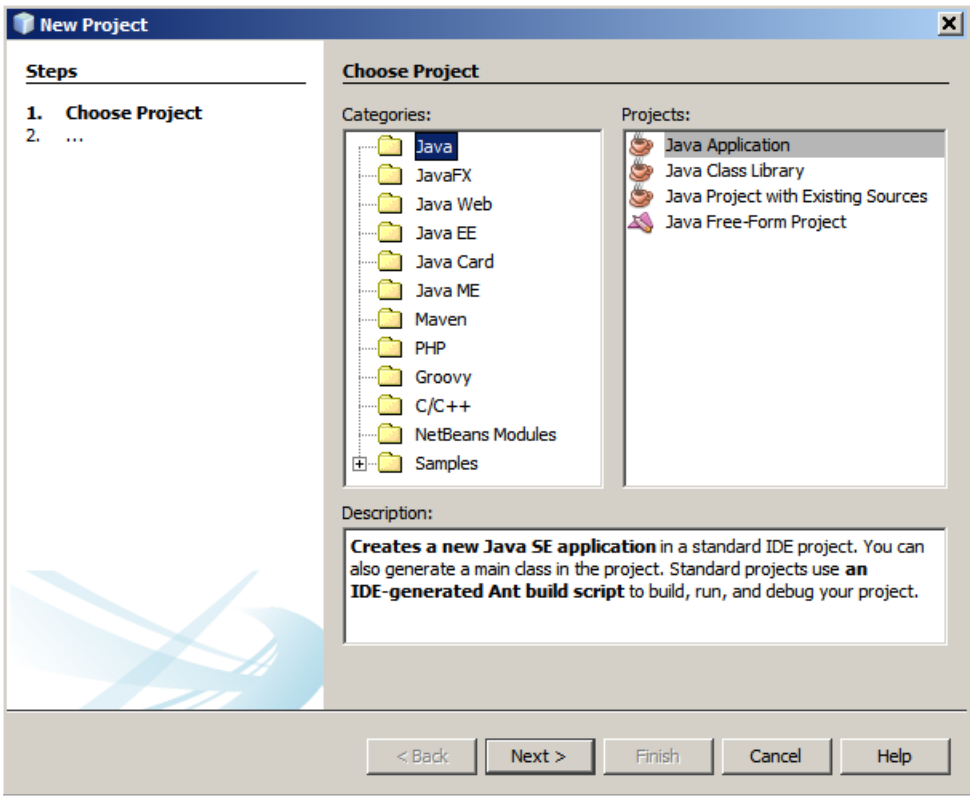
NetBeans is an Integrated Development Environment (IDE) for developing primarily with Java. It also can be used for development with other languages, in particular: PHP, C/C++, and HTML5.

1. Setting up the project.

To create a project using *NetBeans*, first start the *IDE*. Then, choose *File* > *New Project* as shown in the figure:



In the *New Project* wizard, expand the *Java* category and select *Java Application* as shown in the figure below, then click *Next*.



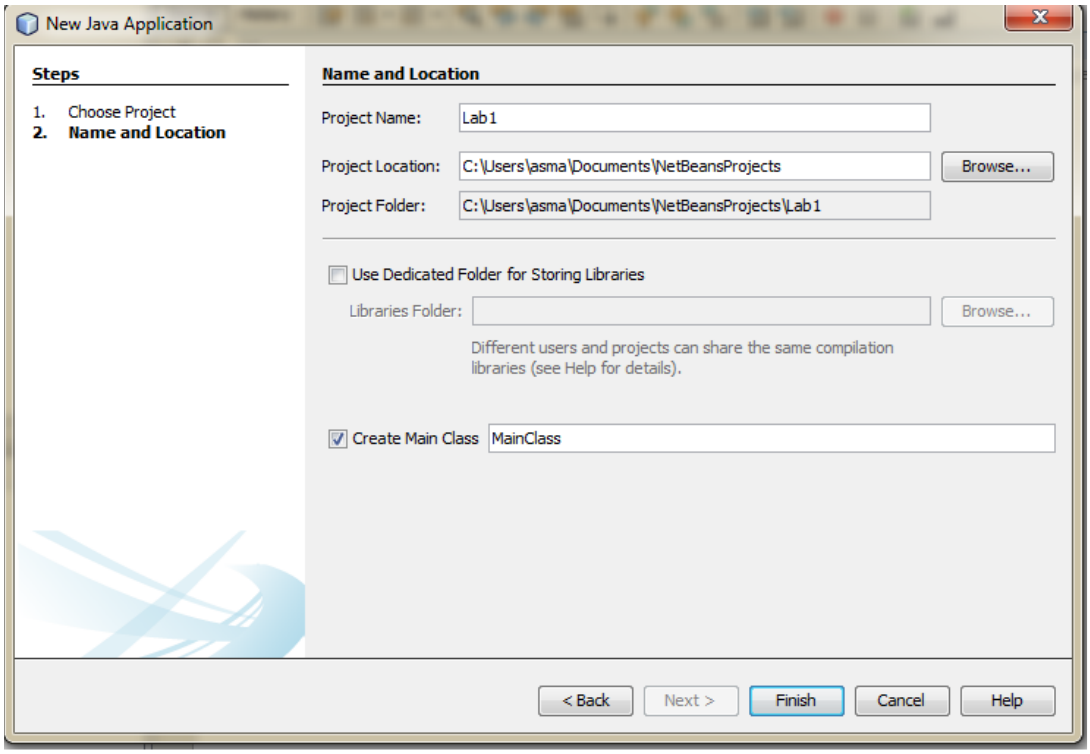
In the *Name* and *Location* page of the wizard, do the following (an example is shown in the figure below):

- In the *Project Name* field, type the name of the project.

- Unless you want to store your project files in another location, keep the *Project Location* field as is. This field is filled automatically with the default location for storing *NetBeans* projects which is a folder named *NetBeansProjects* under your *Documents* folder.

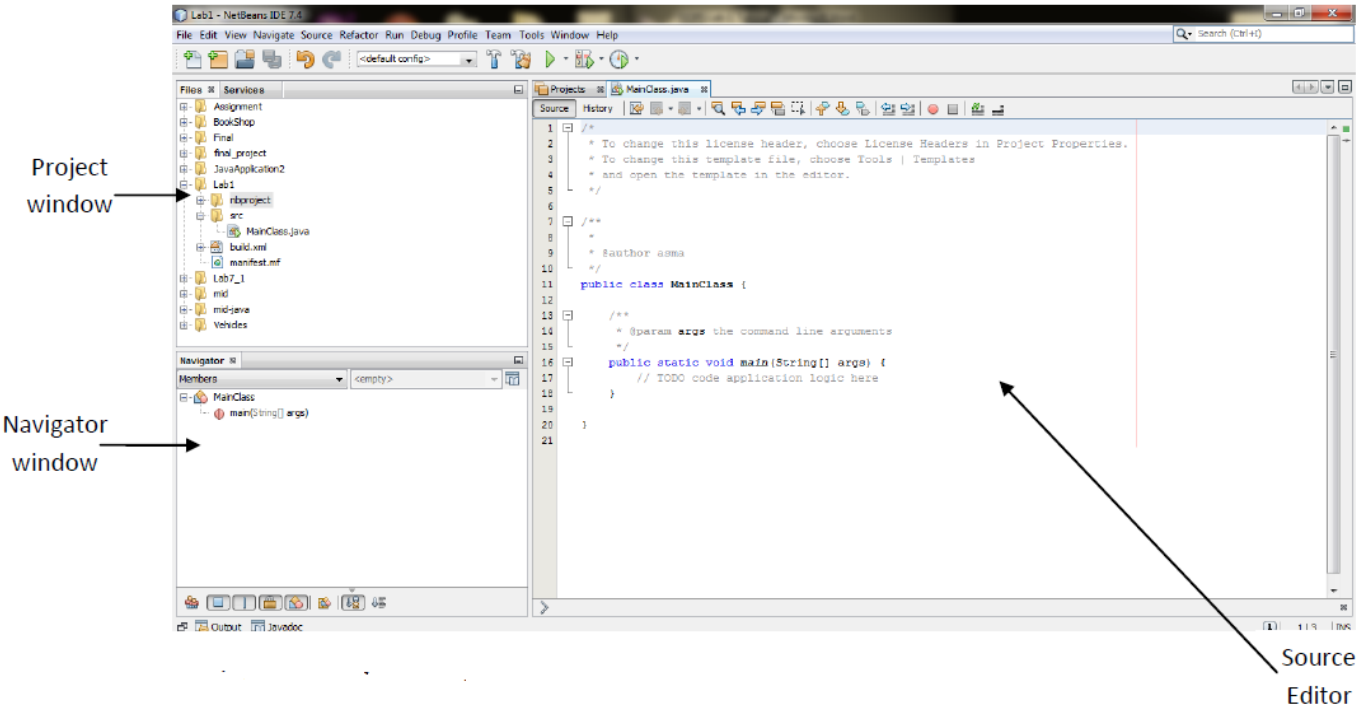
- Leave the *Use Dedicated Folder for Storing Libraries* checkbox unchecked.

- Keep the *Create Main Class* checkbox checked, then type the name of your main class next to it. By leaving this checkbox selected, the IDE will automatically generate the skeleton of the main class: it will define a class with the name you specified and define the main method inside it with an empty body. For this lab, it is preferable to name the main class: *MainClass*.



Click *Finish*. The project is now created and opened in the IDE. You should be able to see the following components:

- The *Project* window which contains a tree view of the components of the project.
- The *Source Editor* window with a file called *MainClass.java* open. Note that the IDE automatically names the .java file with the same name of the main class.
- The *Navigator* window, which you can use to quickly navigate between elements within the selected class.



2. Adding code to the generated source file.

You can add the code to be executed in the main method by replacing the line:
//TODO code application logic here

For example try replacing it with the statement:
System.out.println("Welcome to Java!");

Save the change by choosing *File>Save*.
The file should look something like the following sample:

```
Source  History
1  /*
2   * To change this license header, choose License Headers in Project Properties.
3   * To change this template file, choose Tools | Templates
4   * and open the template in the editor.
5   */
6
7  /**
8   *
9   * @author asma
10  */
11 public class MainClass {
12
13     /**
14      * @param args the command line arguments
15      */
16     public static void main(String[] args) {
17         System.out.println("Welcome to Java!");
18     }
19
20 }
21
```

3. Compiling and running the program.

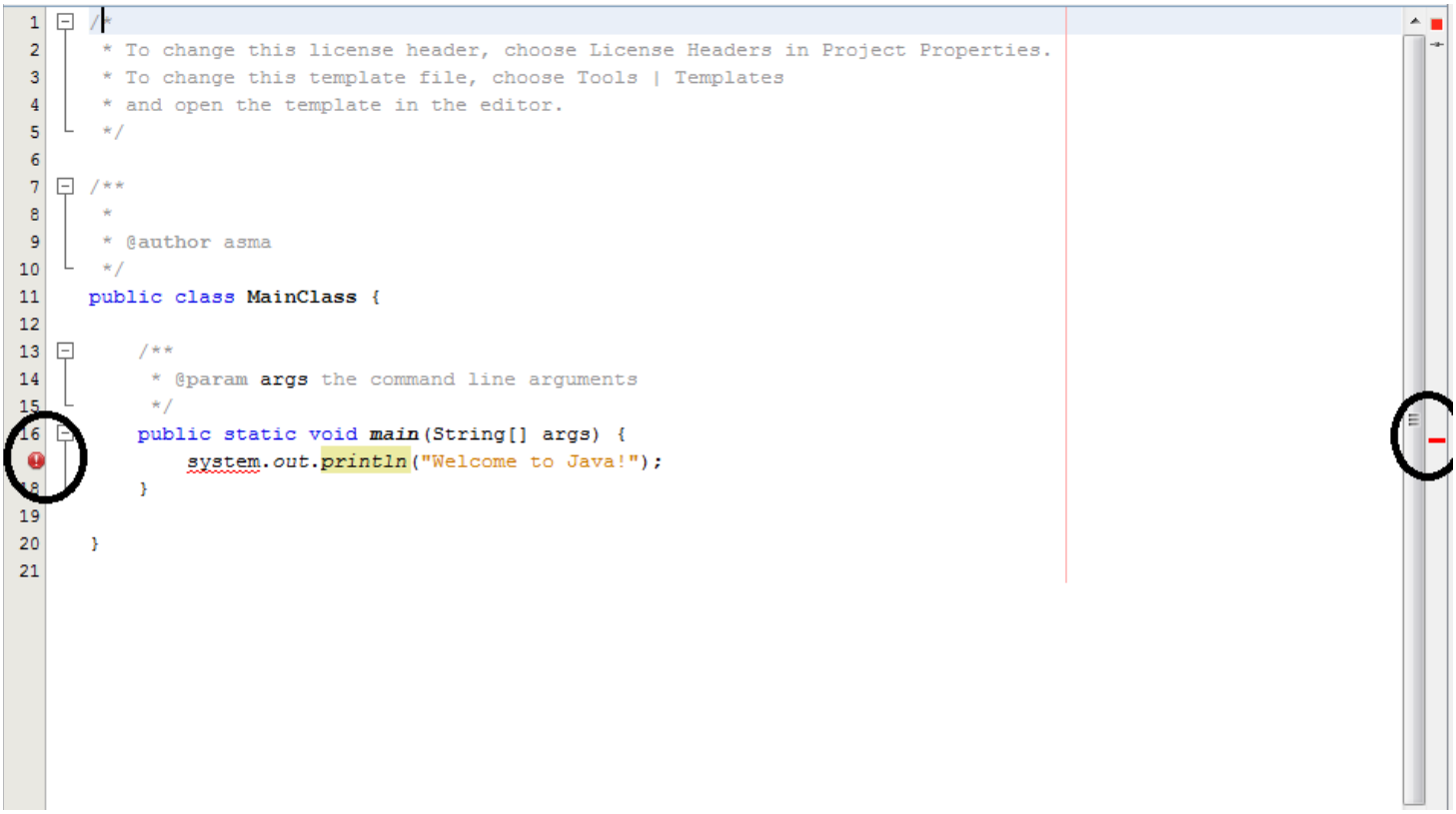
Because of the IDE's *Compile on Save* feature, you do not have to manually compile your project in order to run it in the IDE. When you save a Java source file, the IDE automatically compiles it. However, if you want to compile your file manually, choose *Run* > *Compile File* or press F9.

Note: The *Compile on Save* feature can be turned off in the Project Properties window. Right-click your project, select *Properties*. In the *Propeties* window, choose the *Compiling* tab. The *Compile on Save* checkbox is right at the top.

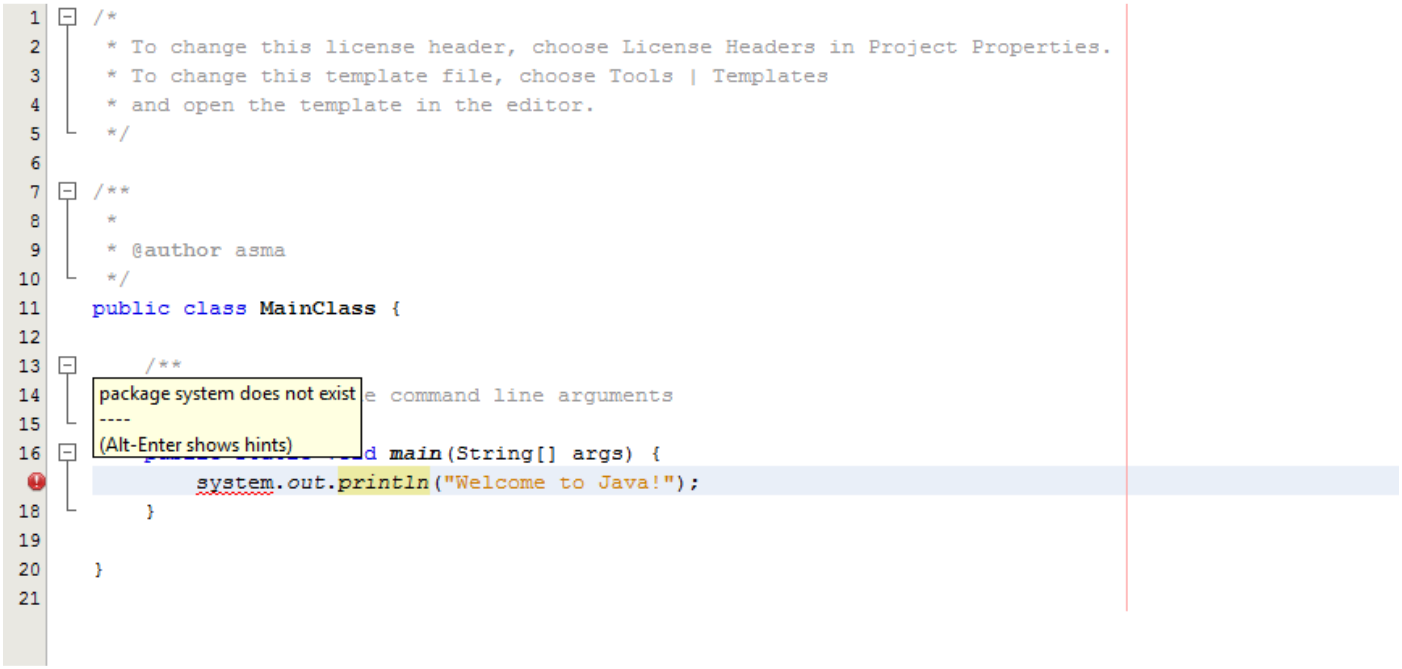
To run the program, choose *Run* > *Run Project* or press F6.
The following figure shows the output of the previous code example:

```
Output - Lab1 (run)
run:
Welcome to Java!
BUILD SUCCESSFUL (total time: 0 seconds)
Output  Javadoc
```

If your code contains compilation errors, they will be marked with red glyphs in the left and right margins of the *Source Editor* as shown in the following figure:



You can mouse over an error mark to get a description of the error as in the following figure:



❖ Part III: Programming Errors Examples (1.10.1-3).

1. Syntax Errors:

Syntax errors result from errors in code construction, such as mistyping a keyword, omitting some necessary punctuation, or using an opening brace without a corresponding closing brace. These errors are usually easy to detect because the compiler tells you where they are and what caused them.

2. Run-time Errors:

Input mistakes typically cause runtime errors. An input error occurs when the program is waiting for the user to enter a value, but the user enters a value that the program cannot handle. For instance, if the program expects to read in a number, but instead the user enters a string, this causes data-type errors to occur in the program.

Another example of runtime errors is division by zero. This happens when the divisor is zero for integer divisions.

```
1  public class ShowRuntimeErrors {
2    public static void main(String[] args) {
3      System.out.println(1 / 0);
4    }
5  }
```

3. Logical Errors:

The following figure shows an example of a logical error. The program must convert Celsius 35 degrees to a Fahrenheit degree. You will get Fahrenheit 67 degrees, which is wrong. It should be 95.0. In Java, the division for integers is the quotient—the fractional part is truncated—so in Java 9 / 5 is 1. To get the correct result, you need to use 9.0 / 5, which results in 1.8.

```
1  public class ShowLogicErrors {
2    public static void main(String[] args) {
3      System.out.println("Celsius 35 is Fahrenheit degree ");
4      System.out.println((9 / 5) * 35 + 32);
5    }
6  }
```

❖ Part IV: Common Errors (1.10.4).

Common Error 1: Missing Braces

```
public class Welcome {

} ←——Type this closing brace right away to match the opening brace
```

Common Error 2: Missing Semicolons

```
public static void main(String[] args) {
  System.out.println("Programming is fun!");
  System.out.println("Fundamentals First");
  System.out.println("Problem Driven")
}
```

Missing a semicolon

Common Error 3: Missing Quotation Marks

```
System.out.println("Problem Driven );
```

Missing a quotation mark

Common Error 4: Misspelling Names

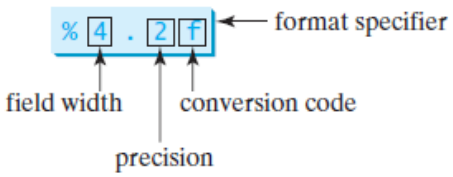
```
public class Test {
  public static void Main(string[] args) {
    System.out.println((10.5 + 2 * 3) / (45 - 3.5));
  }
}
```

❖ Part V: The *print* and *printf* statements (4.6)

The *print* method is identical to the *println* method except that *println* moves to the beginning of the next line after displaying the string, but *print* does not advance to the next line when completed.

You can use the *System.out.printf* method to display formatted output on the console.

```
double amount = 12618.98;
double interestRate = 0.0013;
double interest = amount * interestRate;
System.out.printf("Interest is $%4.2f",
  interest);
```




```
Interest is $16.40
```

The syntax to invoke this method is:
System.out.printf(format, item1, item2, ..., itemk)

Where format is a string that may consist of substrings and format specifiers. A format specifier specifies how an item should be displayed. An item may be a numeric value, a character, a Boolean value, or a string. A simple format specifier consists of a percent sign (%) followed by a conversion code.

TABLE 4.11 Frequently Used Format Specifiers

Format Specifier	*Output*	*Example*
`%b`	a Boolean value	true or false
`%c`	a character	'a'
`%d`	a decimal integer	200
`%f`	a floating-point number	45.460000
`%e`	a number in standard scientific notation	4.556000e+01
`%s`	a string	"Java is cool"

An Example:

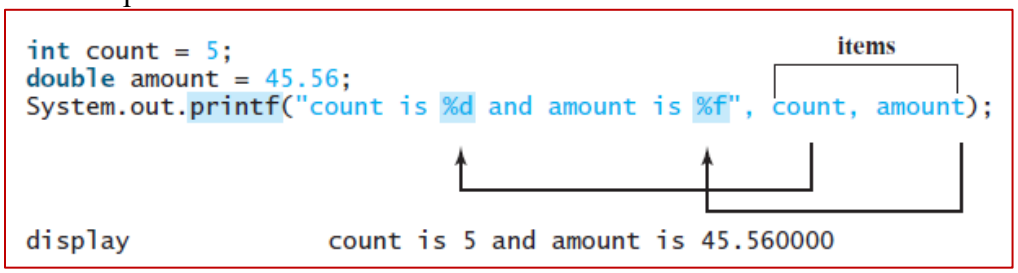


TABLE 4.12 Examples of Specifying Width and Precision

Example	*Output*
`%5c`	Output the character and add four spaces before the character item, because the width is 5.
`%6b`	Output the Boolean value and add one space before the false value and two spaces before the true value.
`%5d`	Output the integer item with width at least 5. If the number of digits in the item is < 5, add spaces before the number. If the number of digits in the item is > 5, the width is automatically increased.
`%10.2f`	Output the floating-point item with width at least 10 including a decimal point and two digits after the point. Thus, there are 7 digits allocated before the decimal point. If the number of digits before the decimal point in the item is < 7, add spaces before the number. If the number of digits before the decimal point in the item is > 7, the width is automatically increased.
`%10.2e`	Output the floating-point item with width at least 10 including a decimal point, two digits after the point and the exponent part. If the displayed number in scientific notation has width less than 10, add spaces before the number.
`%12s`	Output the string with width at least 12 characters. If the string item has fewer than 12 characters, add spaces before the string. If the string item has more than 12 characters, the width is automatically increased.

By default, the output is right justified. You can put the *minus sign (-)* in the format specifier to specify that the item is left justified in the output within the specified field.

The University of Jordan
Faculty of Engineering and Technology
Department of Computer Engineering

Object-Oriented Problem Solving: CPE 342
Lab 1: Getting Started + Fundamentals
Eng. Asma Abdel Karim

---

1. Write a statement that prints your name and ID to one line of the console.

2. Write a statement that prints the result of the operation $\frac{4}{3} * 2.5$ along with the operation as follows:
   *4/3*2.5= 3.333333333333333*

3. Write a statement that prints the result of the operation $\frac{4}{3} * 2.5$ along with the operation as follows (showing two decimal digits only to the right of the point):
   *4/3*2.5= 3.33*

4. Write the following statements:
   a. Define an integer variable named x (without initializing it).
   b. Try printing the value of x using the print statement
      System.out.println("x= "+x);
      Identify the problem.

5. Write statements that:
   a. Declare and initialize a floating-point value that represents the radius of a circle.
   b. Define and initialize π as a constant
   c. Compute the area of the circle.
   d. Print the following statement:

   Circle with radius ___ has area of ____ .

Note: your program must print correct output for any possible radius value.

6. Write statements that:
   a. Declare and initialize two double variables (X with the value 3.5 and Y with the value 2.5).
   b. Print the result of adding the two variables as follows: 3.5 + 2.5 = 6.0
   c. Print the result of subtracting the two variables as follows: 3.5 – 2.5 = 1.0
   d. Print the result of multiplying the two variables as follows: 3.5 * 2.5 = 8.75
   e. Print the result of dividing the two variables as follows: 3.5 / 2.5 = 1.4

Note-1: your code must print correct results if the values of variables X and Y are changed. You must not write your code to assume that the value of X is fixed to 3.5 and the value of Y is fixed to 2.5.

Note-2: when printing the result of each operation, pay attention to the necessity of adding parenthesis. For example, for the "addition" operation try the following two statements and check which one produces correct result:

```
System.out.println(X+" + "+Y+" = "+X+Y);
System.out.println(X+" + "+Y+" = "+(X+Y));
```

Try adding and removing the parenthesis in the print statement of each operation. For which operations is it required to add the parenthesis? For which operations adding the requirements is not a necessity?

7. Write statements that:
   a. Declare and initialize an integer variable 'a' and initialize it. Then declare a double variable 'b' and assign the value of 'a' to it. Does it work?
   b. Declare and initialize a double variable 'c' and initialize it. Then declare an integer variable 'd' and assign the value of 'c' to it. Does it work?
   c. Declare and initialize a float variable 'm' and assign the value 3.5 to it. Does it work?

Jordan University
Faculty of Engineering and Technology
Department of Computer Engineering

Object-Oriented Problem Solving: CPE 342

Lab-2 Tutorial: Programming Fundamentals + Methods

Eng. Asma Abdel Karim

❖ **Tutorial contents:**

- Part I: Common Errors and Pitfalls.
- Part II: Reading Input from the Console
- Part III: Passing Parameters by Values.

❖ Part I: Common Errors and Pitfalls (2.18, 3.6)

Common Error 1: Undeclare/Uninitialized Variables and Unused Variables.

```
double interestRate = 0.05;
double interest = interestrate * 45;
```

```
double interestRate = 0.05;
double taxRate = 0.05;
double interest = interestRate * 45;
System.out.println("Interest is " + interest);
```

Common Error 2: Unintended Integer Division.

```
int number1 = 1;
int number2 = 2;
double average = (number1 + number2) / 2;
System.out.println(average);
```

```
int number1 = 1;
int number2 = 2;
double average = (number1 + number2) / 2.0;
System.out.println(average);
```

Common Error 3: Forgetting Necessary Braces.

```
if (radius >= 0)
  area = radius * radius * PI;
  System.out.println("The area "
    + " is " + area);
```

(a) Wrong

```
if (radius >= 0) {
  area = radius * radius * PI;
  System.out.println("The area "
    + " is " + area);
}
```

(b) Correct

Common Error 4: Wrong Semicolon at the If Line.

Logic error

```
if (radius >= 0);
{
  area = radius * radius * PI;
  System.out.println("The area "
    + " is " + area);
}
```

(a)

Equivalent

Empty block

```
if (radius >= 0) { };
{
  area = radius * radius * PI;
  System.out.println("The area "
    + " is " + area);
}
```

(b)

Common Error 5: Dangling *else* Ambiguity.

```
int i = 1, j = 2, k = 3;

if (i > j)
  if (i > k)
    System.out.println("A");
else
    System.out.println("B");
```

(a)

Equivalent

This is better with correct indentation

```
int i = 1, j = 2, k = 3;

if (i > j)
  if (i > k)
    System.out.println("A");
  else
    System.out.println("B");
```

(b)

Common Pitfall 1: Redundant Input Objects.

BAD CODE

```
Scanner input = new Scanner(System.in);
System.out.print("Enter an integer: ");
int v1 = input.nextInt();

Scanner input1 = new Scanner(System.in);
System.out.print("Enter a double value: ");
double v2 = input1.nextDouble();
```

GOOD CODE

```
Scanner input = new Scanner(System.in);
System.out.print("Enter an integer: ");
int v1 = input.nextInt();
System.out.print("Enter a double value: ");
double v2 = input.nextDouble();
```

Common Pitfall 2: Redundant Testing of Boolean Values.

```
if (even == true)
  System.out.println(
    "It is even.");
```

(a)

Equivalent

This is better

```
if (even)
  System.out.println(
    "It is even.");
```

(b)

Common Pitfall 3: Simplifying *boolean* Variable Assignment.

```
if (number % 2 == 0)
  even = true;
else
  even = false;
```

(a)

Equivalent

This is shorter

```
boolean even
  = number % 2 == 0;
```

(b)

Common Pitfall 4: Avoiding Duplicate Code in Different Cases.

```
if (inState) {
  tuition = 5000;
  System.out.println("The tuition is " + tuition);
}
else {
  tuition = 15000;
  System.out.println("The tuition is " + tuition);
}
```

❖ Part II: Reading Input from the Console (2.3, 2.9.2, 4.4.5, 4.4.6).

Java uses *System.out* to refer to the standard output device and *System.in* to the standard input device. By default, the output device is the display monitor and the input device is the keyboard.
To perform console output, you simply use the *println* method to display a primitive value or a string to the console. Console input is not directly supported in Java, but you can use the *Scanner* class to create an object to read input from *System.in*, as follows:

Scanner input = new Scanner(System.in);

The syntax *new Scanner(System.in)* creates an object of the *Scanner* type. The syntax *Scanner input* declares that input is a variable whose type is *Scanner*. The whole line *Scanner input = new Scanner(System.in)* creates a *Scanner* object and assigns its reference to the variable input. An object may invoke its methods. To invoke a method on an object is to ask the object to perform a task. You can invoke the *nextDouble()* method to read a double value as follows:

double radius = input.nextDouble();

This statement reads a number from the keyboard and assigns the number to *radius*.

```
import java.util.Scanner; // Scanner is in the java.util package

public class ComputeAreaWithConsoleInput {
  public static void main(String[] args) {
    // Create a Scanner object
    Scanner input = new Scanner(System.in);

    // Prompt the user to enter a radius
    System.out.print("Enter a number for radius: ");
    double radius = input.nextDouble();

    // Compute area
    double area = radius * radius * 3.14159;

    // Display results


    System.out.println("The area for the circle of radius " +
      radius + " is " + area);
  }
}
```

import class

create a Scanner

read a double

Methods for reading numbers using the Scanner class:

Method	*Description*
`nextByte()`	reads an integer of the `byte` type.
`nextShort()`	reads an integer of the `short` type.
`nextInt()`	reads an integer of the `int` type.
`nextLong()`	reads an integer of the `long` type.
`nextFloat()`	reads a number of the `float` type.
`nextDouble()`	reads a number of the `double` type.

Examples for reading values of various types from the keyboard:

```
Scanner input = new Scanner(System.in);
System.out.print("Enter a byte value: ");
byte byteValue = input.nextByte();

System.out.print("Enter a short value: ");
short shortValue = input.nextShort();

System.out.print("Enter an int value: ");
int intValue = input.nextInt();

System.out.print("Enter a long value: ");
long longValue = input.nextLong();

System.out.print("Enter a float value: ");
float floatValue = input.nextFloat();
```

To read a string from the console, invoke the *next*() method or the *nextLine* method on a Scanner Object. The *next*() method reads a string that ends with a whitespace character. You can use the *nextLine*() method to read an entire line of text. The *nextLine*() method reads a string that ends with the Enter key pressed.
Example using the next() method:

```
Scanner input = new Scanner(System.in);
System.out.print("Enter three words separated by spaces: ");
String s1 = input.next();
String s2 = input.next();
String s3 = input.next();
System.out.println("s1 is " + s1);
System.out.println("s2 is " + s2);
System.out.println("s3 is " + s3);
```

Example using the *nextLine*() method:

```
Scanner input = new Scanner(System.in);
System.out.println("Enter a line: ");
String s = input.nextLine();
System.out.println("The line entered is " + s);
```

```
Enter a line: Welcome to Java ↵Enter
The line entered is Welcome to Java
```

To read a character from the console, use the *nextLine*() method to read a string and then invoke the *charAt(0)* method on the string to return a character.

```
Scanner input = new Scanner(System.in);
System.out.print("Enter a character: ");
String s = input.nextLine();
char ch = s.charAt(0);
System.out.println("The character entered is " + ch);
```

❖ **Part III: Passing Parameters by Values.**

When you invoke a method in Java, the value of the argument is passed to the parameter. This is referred to as pass-by-value. If the argument is a variable rather than a literal value, the value of the variable is passed to the parameter. The variable is not affected, regardless of the changes made to the parameter inside the method.

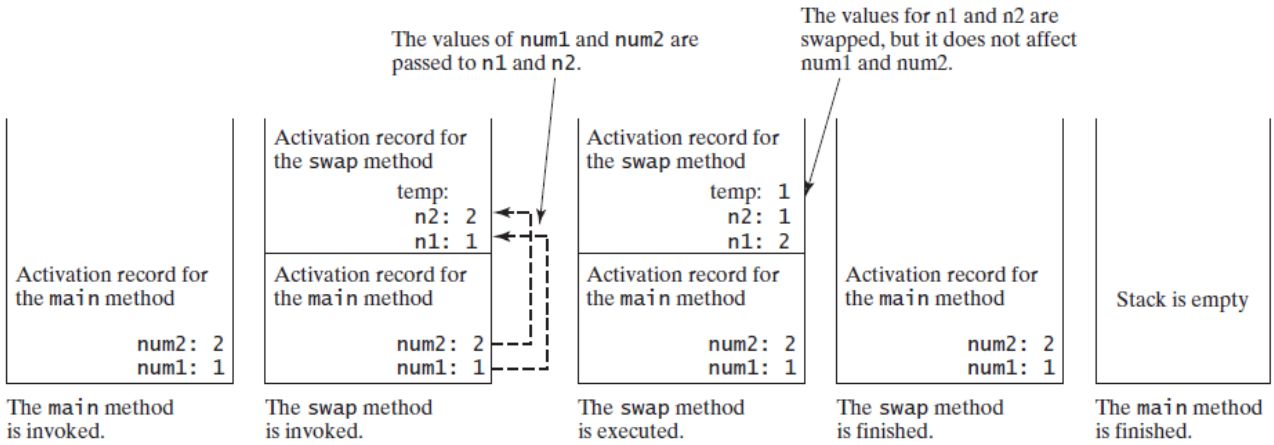
The following example will print the value of max 0, since 1) the variable max inside the max method is different from the variable max in the main method, and 2) the value of the variable max of the main method is passed to the variable max in the max method. Changes to the max variable inside the max method do not affect the max variable in the main method.

```
public class MainClass {

   public static void main(String[] args) {
      int max = 0;
      max (1, 2, max);
      System.out.println(max);
   }

   public static void max(int value1, int value2, int max){
      if (value1 > value2) max = value1;
      else max = value2;
   }

}
```

The following example gives another program that demonstrates the effect of passing by value. The program creates a method for swapping two variables. The *swap* method is invoked by passing two arguments. Interestingly, the values of the arguments are not changed after the method is invoked.

```
1  public class TestPassByValue {
2    /** Main method */
3    public static void main(String[] args) {
4      // Declare and initialize variables
5      int num1 = 1;
6      int num2 = 2;
7
8      System.out.println("Before invoking the swap method, num1 is " +
9        num1 + " and num2 is " + num2);
10
11     // Invoke the swap method to attempt to swap two variables
12     swap(num1, num2);
13
14     System.out.println("After invoking the swap method, num1 is " +
15       num1 + " and num2 is " + num2);
16   }
17
18   /** Swap two variables */
19   public static void swap(int n1, int n2) {
20     System.out.println("\tInside the swap method");
21     System.out.println("\t\tBefore swapping, n1 is " + n1
22       + " and n2 is " + n2);
23
24     // Swap n1 with n2
25     int temp = n1;
26     n1 = n2;
27     n2 = temp;
28
29     System.out.println("\t\tAfter swapping, n1 is " + n1
30       + " and n2 is " + n2);
31   }
32 }
```

```
Before invoking the swap method, num1 is 1 and num2 is 2
  Inside the swap method
    Before swapping, n1 is 1 and n2 is 2
    After swapping, n1 is 2 and n2 is 1
After invoking the swap method, num1 is 1 and num2 is 2
```



The University of Jordan
Faculty of Engineering and Technology
Department of Computer Engineering

Object-Oriented Problem Solving: CPE 342
Lab 2: Programming Fundamentals + Methods
Eng. Asma Abdel Karim

Part-1: In your main method write a program that: (3 points)

Keeps on reading double values from the user, until the user enters the number 0. When the user finishes entering the numbers, the program must print their average to the console as follows:

Average of the numbers you entered is: ……

Part-2: In your main class write the following methods: (3 points)

1. Define a method named *printEven* that takes two integer numbers and prints the even numbers between these two numbers (inclusive) comma separated, then a new line.

   For example, if the passed integers are 1 and 10, the method must print:
   2, 4, 6, 8, 10

   This method assumes that the first passed integer is smaller than the second.
   The header of the method is:

   public static void printEven (int n1, int n2)

2. Define a method named *getEven* that takes two integer numbers and returns a String that includes the even numbers between these two numbers (inclusive) comma separated, then a new line.

   For example, if the passed integers are 1 and 10, the method must return the String:
   2, 4, 6, 8, 10

   This method assumes that the first passed integer is smaller than the second.
   The header of the method is:

   public static String getEven (int n1, int n2)

3. In your main method, read two integer numbers from the user. Prompt the user to enter the required input with appropriate messages as follows: "*Enter first number:* ", "*Enter second number:*". Then, invoke both methods to print their output.

   Note: Make sure you pass the minimum of the two numbers as the first argument and the maximum as the second argument.

.

Part-3: In your main class you are required to write the following methods: (4 points)

1. Define a method named *gcd* that takes two integer numbers and returns the greatest common divisor of these two numbers. The header of the method is:

   public static int gcd (int n1, int n2)

   For example, if the passed integers are 16 and 56, the method must return the number 8.

2. Define a method named *lcm* that takes two integer numbers and returns the least common multiple of these two numbers. The header of the method is:

   public static int lcm (int n1, int n2)

   For example, if the passed integers are 16 and 20, the method must return the number 80.

3. Define a method named *sum* that takes two integers and returns the sum of all numbers between these two integers (inclusive). The header of the method is:

   public static int sum (int n1, int n2)

   For example, if the passed integers are 6 and 10, the method must return the number 40.

   Note: The method must produce the summation even if the first number is greater than the second. For example, if the passed integers are 10 and 6, the method must return 40 as well.

4. In your main method, read two integers from the user then one of the characters 'g', 'l', or 's'. If the user enters any character other than these characters, the program must keep on prompting the user to enter a valid character.
   Then, based on the entered character, invoke the appropriate method from the methods above (g → gcd, l → lcm, s → sum) and print the output properly.

   For example:
   - If the user enters the numbers 6, 10, and the character s, the program must print the statement:

   Sum of numbers between 6 and 10 is 40.

   - If the user enters the numbers 6, 10, and the character g, the program must print the statement:

   GCD of numbers 6 and 10 is 2.

   - If the user enters the numbers 6, 10, and the character l, the program must print the statement:

   LCM of numbers 6 and 10 is 30.

`

The University of Jordan
Faculty of Engineering and Technology
Department of Computer Engineering

Object-Oriented Problem Solving: CPE 342
Lab 3: Arrays
Eng. Asma Abdel Karim

❖ Tutorial Contents:

- Part-1: Common Mathematical Functions
- Part-2: Examples on Processing Arrays.
- Part-3: Passing Arrays to Methods.
- Part-4: Returning Arrays from Methods.

❖ Part-1: Common Mathematical Functions.

Java provides many useful methods in the ***Math* class** for performing common mathematical functions. They can be categorized as ***trigonometric*** methods, ***exponent*** methods, and ***service*** methods. *Service* methods include the rounding, min, max, absolute, and random methods.

In addition to methods, the *Math* class provides two useful double *static* constants, *PI* and *E* (the base of natural logarithms). You can use these constants as ***Math.PI*** and ***Math.E*** in any program.

TABLE 4.1 Trigonometric Methods in the Math Class

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.

TABLE 4.2 Exponent Methods 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.

TABLE 4.3 Rounding Methods in the Math Class

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 *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). For example:
Math.abs(-2) returns 2
Math.abs(-2.1) returns 2.1

The *random* 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. For example:

```
(int)(Math.random() * 10)          ⟶  Returns a random integer between 0 and 9.
50 + (int)(Math.random() * 50)     ⟶  Returns a random integer between 50 and 99.
```

In general:

```
a + Math.random() * b   ⟶  Returns a random number between a and a + b, excluding a + b.
```

❖ Part-2: Examples on Processing Arrays.

- **Finding the largest element**: Use a variable named *max* to store the largest element. Initially *max* is *myList[0]*. To find the largest element in the array *myList*, compare each element with *max*, and update *max* if the element is greater than *max*.

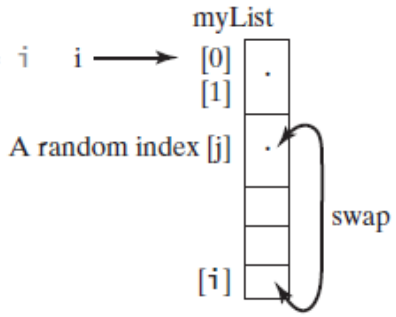
```
double max = myList[0];
for (int i = 1; i < myList.length; i++) {
  if (myList[i] > max) max = myList[i];
}
```

- **Finding the smallest index of the largest element**: Often you need to locate the largest element in an array. If an array has multiple elements with the same largest value, find the smallest index of such an element. Suppose the array *myList* is {1, 5, 3, 4, 5, 5}. The largest element is 5 and the smallest index for 5 is 1. Use a variable named *max* to store the largest element and a variable named *indexOfMax* to denote the index of the largest element. Initially *max* is *myList*[0], and *indexOfMax* is 0. Compare each element in *myList* with max, and update max and *indexOfMax* if the element is greater than max.

```
double max = myList[0];
int indexOfMax = 0;
for (int i = 1; i < myList.length; i++) {
  if (myList[i] > max) {
    max = myList[i];
    indexOfMax = i;
  }
}
```

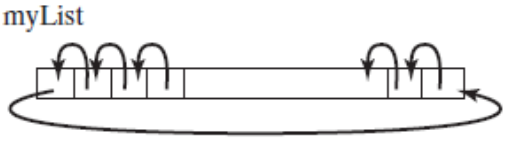
- **Random shuffling**: In many applications, you need to randomly reorder the elements in an array. This is called shuffling. To accomplish this, for each element *myList[i]*, randomly generate an index j and swap *myList[i]* with *myList[j]*, as follows:

```
for (int i = myList.length - 1; i > 0; i--) {
  // Generate an index j randomly with 0 <= j <= i
  int j = (int)(Math.random()
    * (i + 1));

  // Swap myList[i] with myList[j]
  double temp = myList[i];
  myList[i] = myList[j];
  myList[j] = temp;
}
```



- **Shifting elements**: Sometimes you need to shift the elements left or right. Here is an example of shifting the elements one position to the left and filling the last element with the first element:

```
double temp = myList[0]; // Retain the first element

// Shift elements left
for (int i = 1; i < myList.length; i++) {
  myList[i - 1] = myList[i];
}

// Move the first element to fill in the last position
myList[myList.length - 1] = temp;
```



❖ Part-3: Passing Arrays to Methods.

The following example shows the difference between passing a primitive data type value and an array reference variable to a method. The program contains two methods for swapping elements in an array. The first method named *swap*, fails to swap two integer arguments. The second method named *swapFirstTwoInArray*, successfully swaps the first two elements in the array argument.

The output of this program is as follows:

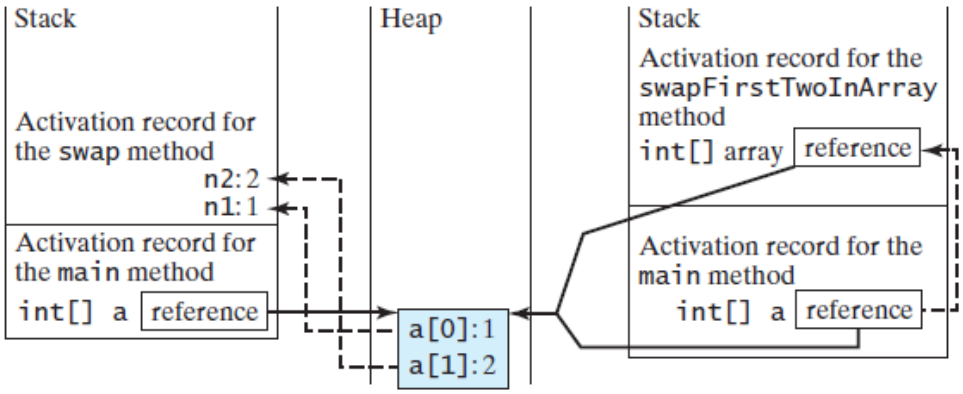
```
Before invoking swap
array is {1, 2}
After invoking swap
array is {1, 2}
Before invoking swapFirstTwoInArray
array is {1, 2}
After invoking swapFirstTwoInArray
array is {2, 1}
```

As shown in the output, the two elements are not swapped using the swap method. However, they are swapped using the *swapFirstTwoInArray* method. Since the parameters in the *swap* method are primitive type, the values of a[0] and a[1] are passed to n1 and n2 inside the method when invoking swap (a[0], a[1]). The memory locations for n1 and n2 are independent of the ones for a[0] and a[1]. The parameter in the *swapFirstTwoInArray* method is an array. The reference of the array is passed to the method. Thus the variables a (outside the method) and array (inside the method) both refer to the same array in the same memory location. This is shown in the following figure:

```
1  public class TestPassArray {
2    /** Main method */
3    public static void main(String[] args) {
4      int[] a = {1, 2};
5
6      // Swap elements using the swap method
7      System.out.println("Before invoking swap");
8      System.out.println("array is {" + a[0] + ", " + a[1] + "}");
9      swap(a[0], a[1]);
10     System.out.println("After invoking swap");
11     System.out.println("array is {" + a[0] + ", " + a[1] + "}");
12
13     // Swap elements using the swapFirstTwoInArray method
14     System.out.println("Before invoking swapFirstTwoInArray");
15     System.out.println("array is {" + a[0] + ", " + a[1] + "}");
16     swapFirstTwoInArray(a);
17     System.out.println("After invoking swapFirstTwoInArray");
18     System.out.println("array is {" + a[0] + ", " + a[1] + "}");
19   }
20
21   /** Swap two variables */
22   public static void swap(int n1, int n2) {
23     int temp = n1;
24     n1 = n2;
25     n2 = temp;
26   }
27
28   /** Swap the first two elements in the array */
29   public static void swapFirstTwoInArray(int[] array) {
30     int temp = array[0];
31     array[0] = array[1];
32     array[1] = temp;
33   }
34 }
```

false swap

swap array elements



Invoke `swap(int n1, int n2)`. The primitive type values in `a[0]` and `a[1]` are passed to the `swap` method.

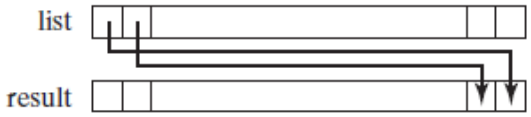
The arrays are stored in a heap.

Invoke `swapFirstTwoInArray(int[] array)`. The reference value in `a` is passed to the `swapFirstTwoInArray` method.

❖ Part-4: Returning Arrays from Methods.

For example, the following method returns an array that is the reversal of another array.

```
1 public static int[] reverse(int[] list) {
2   int[] result = new int[list.length];
3
4   for (int i = 0, j = result.length - 1;
5        i < list.length; i++, j--) {
6     result[j] = list[i];
7   }
8
9   return result;
10 }
```



For example, the following statement returns a new array list2 with elements 6, 5, 4, 3, 2, 1.

```
int[] list1 = {1, 2, 3, 4, 5, 6};
int[] list2 = reverse(list1);
```

`

The University of Jordan
Faculty of Engineering and Technology
Department of Computer Engineering

Object-Oriented Problem Solving: CPE 342
Lab 3: Arrays
Eng. Asma Abdel Karim

Note: You are not allowed to use methods from the *Arrays* class.
In your *main* method, follow the steps below, to write the required code:

1. Declare a reference variable to an array of integer elements. The name of the variable is *list1*.

   Remember that the syntax for declaring a reference variable is:
   elementType [] refVariable;

2. Print the reference variable to the console, as in the statement below, and observe and understand what happens. Then, comment the print statement.
   System.out.println(list1);

3. Create an array of five integer values and assign its reference to the variable *list1*.

   Remember that the syntax for creating an array is:
   new elementType[arraySize];

   In order to assign the returned reference to a reference variable, we put the new statement to the right of an assignment statement to a reference variable.
   refVariable = new elementType[arraySize];

   Combine array declaration and creation such that they are on the same line as in the following syntax:
   elementType [] refVariable= new elementType[arraySize];

4. Try to print the value of the reference variable to the console again and observe the output.

5. Print the values of the array elements all on the same line comma separated using a "for" loop. For this part, use a for loop with the following header:

   for(int i=0; i<list1.length; i++)
   //print statement

   Make sure to print a new line after printing the array elements.

6. Assign values to the array elements using a "for" loop with a header similar to that in the previous step. Read the values to be assigned from the user using an appropriate prompting message.

   Note: prompt the user to enter positive values. For each element, if the user enters an invalid value (zero or negative value), you must keep on prompting the user until a positive value is entered.

7. Print elements values to the console again, on the same line comma separated, this time using the "foreach" statement as follows:

   foreach(int e:list1)
   //print statement

   Make sure to print a new line after printing the array elements.

8. Declare and create an array of ten integers named *list2*, then initialize elements' values with random numbers between [100-200]. Then print elements values to the console on the same comma space separated. Make sure to print a new line after printing the array elements.

9. Declare and create a new array, named *list3*, that is formed by concatenating elements of *list1* and *list2*. Remember to declare the array *list3* such that its length is equal to the sum of *list1* and *list2* lengths.
   - Try copying elements from *list1* and *list2* to *list3* using for loops.
   - Try copying elements from *list1* and *list2* using the *arraycopy* method of the *System* class.

   Keep and submit both approaches (you can comment one them).

   Print the elements of list3 on the same line comma separated. Make sure to print a new line after printing the array elements.

10. Find the maximum and minimum values in the array list3, and print them to the console as follows: *Maximum value in list3 is ……, minimum value is ……*

11. In this step, new elements are to be added to *list3*. Ask the user to enter the number of elements to be added then read elements values from the user. Print the array elements after the new elements are added.

    Remember that in order to add elements to an array, you have to define a new array with the required size (resulting from adding the elements) and copy elements from the old array then insert new values into the extra positions and reassign the new reference to the array variable.

12. In this step, the user must be prompted to enter a value to be removed from *list3*, then remove all occurrences of this value from *list3*. Print the array elements after the value is removed.

    Remember that in order to remove elements from an array, you have to define a new array with the required size (resulting from removing the elements) and only copy required elements then reassign the new reference to the array variable.

Jordan University
Faculty of Engineering and Technology
Department of Computer Engineering

Object-Oriented Problem Solving: CPE 342

Lab-4 Tutorial

Eng. Asma Abdel Karim

❖ Tutorial contents:

- Part I: Class Definition (Without Constructors).
- Part II: Class Definition (With Constructors).

❖ Part I: Class Definition (Without Constructors).

The following UML diagram represents a class called Student.

Student
name: String id: int isScienceMajor: boolean gender: char grades: double[]
printDetails():void getAverage(): double

The Student class contains the following data fields:

- A String data field called *name*.
- An integer data field called *id*.
- A boolean data field called *isScienceMajor*.
- A character data field called *gender*.
- An array of double values that represent the student *grades*.

The Student class contains the following methods:

- A method called *printDetails* that prints the student details: his id, name and gender on one line, his grades on the next line space separated, and the average of his grades on the last line, as in the following example:
  Student name is Asma, Student id is 88888, Gender: F.
  Is the student major scientific? yes
  19.0 18.0 19.0 20.0
  Average= 19
- A method called *getAverage* that computes and returns the average of the student grades.

The code for the *Student* class shown in the UML diagram above is listed in the file **Lab4Part1.java**.

Note that the *name* data field has a default value of *"Unknown"*, the *id* has a default value of -1, the *isScienceMajor* has a default value of true, the *gender* has a default value of 'F', and the *grades* data field has a default creation of an array of three double values (which will be initialized to 0's). However, for this class that has no constructors, if the data fields were not given default values by the programmer, they will be assigned to the language defaults. These are null for the *name* and *grades* data fields. Trying to use a reference variable with null value will cause a run time error called *NullPointerException*.

- **Objects Declaration and Creation.**

Objects are reference data types. In order to define objects in your program, you need to declare a reference variable to the object type, then create the object using the *new* operator. The following main method declares and creates two objects of type student. Note that each object has its own copy of the class data fields: *name*, *id*, and *grades*.

```
public class MainClass {
   public static void main(String[] args) {
      Student s1 = new Student();
      Student s2 = new Student();
   }
}
```

- **Accessing Objects Members.**

In order to access the members (data fields and methods) of an object we use the dot (.) operator. The following *MainClass* is listed in the file Lab5Part1.java.

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

      Student s1 = new Student();
      Student s2 = new Student();

      s1.name = "Ahmed";
      s1.id = 138219;
      s1.grades[0] = 20;
      s1.grades[1] = 19;
      s1.grades[2] = 15;
      System.out.println(s1.name);
      System.out.println(s1.getAverage());

      s2.grades[0]=12;
      s2.grades[1]=19;
      s2.grades[2]=14;
      s2.printDetails();
   }
}
```

❖ Part II: Class Definition (With Constructors).

The following is the UML diagram of the Student class after adding constructors.

Student
name: String id: int isScienceMajor: boolean gender: char grades: double[]
Student() Student(n: String, i: int, g: char, size: int) Student(n: String, i: int, g:char, gr: double[]) printDetails():void getAverage(): double

The following code implements the class after adding the constructors. Each constructor initializes the data fields in a different way.

- The no-arg constructor *Student()* does not overwrite any of the data fields, it only initializes all elements of the array *grades* to 20.
- The second constructor *Student(n: String, i:int, g:char, size:int)* assigns each data field with the corresponding parameter (i.e. *name* is initialized using the parameter *n*, *id* is initialized using the parameter *i*, and *gender* is initialized with the parameter *g*). The parameter *size* is used to create the array *grades*. The array elements are initialized to 20.
- The third constructor *Student(n: String, i:int, g:char, gr:double[])* assigns each data field with the corresponding parameter (i.e. *name* is initialized using the parameter *n*, *id* is initialized using the parameter *i*, and *gender* is initialized with the parameter *g*). The array *grades* is initialized such that its elements have the same values of corresponding elements in the array *gr*. There are two approaches to do that:
  1. Copy the reference in *gr* to the variable *grades*:
     grades = gr;
  2. Create a new array referenced by *grades* with the same size as *gr*, and copy values of *gr* elements to corresponding elements in *grades*.
     grades = new double[gr.length];
     for(int j=0; j<grades.length; j++)
     grades[j]=gr[j];

The implementation of the *Student* class shown in the UML diagram above, and a *MainClass* that includes a test of the *Student* class is listed in the file **Lab4Part2A.java**.

In the *Student* class implemented in **Lab4Part2B.java**, the no-argument constructor is implemented such that it reads all data fields' values from the user.

The University of Jordan
Faculty of Engineering and Technology
Department of Computer Engineering

Object-Oriented Problem Solving: CPE 342
Lab 4: Objects & Classes

Eng. Asma Abdel Karim

❖ **Lab Objectives:**

- To become familiar with the following:
  - Defining classes.
  - Declaring and creating objects.
  - Accessing objects' members.
  - Declaring constructors.

In this lab, you will implement a class that represents a *Course* and place it in the same file that contains your *main* class, as follows:

- Create a project as in the previous labs and create a class that contains the main method named *MainClass*.
- Write the header for the *Course* class in the same file (*MainClass.java*) either before or after (but not inside) the body of the *MainClass*, as shown in the following screenshots.
  Note that you cannot define two *public* classes in the same file. This is why the header of the *Course* class does not include the *public* modifier.

```
1
2   class Course{
3
4   }
5   public class MainClass {
6
7       public static void main(String[] args) {
8
9       }
10
11  }
12
```

```
1
2   public class MainClass {
3
4       public static void main(String[] args) {
5
6       }
7
8   }
9   class Course{
10
11  }
12
```

Part-1: Implementing the Course class without constructors:

Course
name: String number: int instructor: String students: int[]
getNumberOfStudents(): int printStudentsIDs(): void addStudent(): void removeStudent(): boolean

1. **Inside the body of the *Course* class:**

a. Define the data fields *name*, *number*, *faculty*, and *students* according to their data types in the UML diagram. The *students* array includes IDs of students registered in the course.
For example, the following statement declares the data field *name*:
String name;

b. Define the method *getNumberOfStudents* that returns the number of students registered in the course, **which is the size of the *students* array**.

c. Define the void method *printStudentsIDs* which prints IDs of students (elements of the *students* array) on one line space separated and prints a new line afterwards.

d. Define the method *addStudent* which reads an integer from the user that represents a student ID and add it to the array *students*. Prompt the user to enter the ID with the message: "*Enter student ID to be added:* ".

e. Define the method *removeStudent* which reads an integer from the user that represents a student ID, and searches the *students* array for the requested ID. If it is found, it must remove it from the array and return true. If it is not found, it must return false. Prompt the user to enter the ID with the message: "*Enter student ID to be removed:* ".

Note: you may assume that each student ID appears only once in the array *students*.

2. **In your main method:**

a. Declare and create an object of type *Course* named *c1* as in the following statement:
Course c1 = new Course();

b. Print the values of the object *c1* data fields (*name, number, instructor, and students*) and observe the output. Print the values using full statements as follows:
Course name: ….
Course number: ….
Course instructor: ….
Course students reference variable: ….

Note: In order to access data fields of an object we use the dot operator. For example, in order to print the value of *name* of the object *c1*, we use the following statement:
System.out.println("Course name:" + c1.name);

c. Invoke the method *printStudentsIDs* to print values of the *students* array elements. Observe and justify the output.

Note: in order to invoke the *printStudentsIDs* method for the object *c1*, use the dot operator as in the following example:
c1.printStudentsIDs();

Remember that since *printStudentsIDs* is a void method, you must invoke it as a separate statement.

d. In the *Course* class, add initialization to the declaration of the data fields as follows:
String name = "unknown";
int number = -1;
String instructor = "unknown";
int [] students = new int [5];

- Rerun your code and observe the output.

e. In your main method, assign the following values to the data fields of the object *c1*:
Name: Java
Number: 907342
Instructor: Asma
Students IDs: 192, 172, 423, 123, 342.

Note: In order to assign values to data fields we access them using the dot operator as in the following example:
c1.name = "Java";

f. Invoke the method *addStudent* to add an ID to the *students* array. Invoke the *printStudentsIDs* method to print the students IDs and check the added ID.
g. Invoke the method *removeStudent* to remove an ID from the *students* array. Invoke the *printStudentsIDs* method to print the students IDs and check that the ID is removed.

Part-2: Implementing the Course class with constructors:

Course
name: String number: int instructor: String students: int[]
Course() **Course(n:String, num:int, i:String, s: int[])** getNumberOfStudents(): int printStudentsIDs(): void addStudent(id: int): void removeStudent(id:int): boolean

1. Inside the body of the *Course* class:

a. Declare a constructor with the following header (remember that constructors do not have return types):

Course(String n, int num, String i, int[] s)

This constructor should initialize *name, number, instructor* with the values of *n, num, i,* respectively. As for the *students* array, it must be initialized to be a copy of the array passed in the parameter *s*.

Notice the syntax error that appeared in the declaration of the object *c1* in the main method. Justify that.

b. Declare a constructor with the following header (remember that constructors do not have return types):

Course()

This constructor should have an empty body for now. Notice that there is no syntax error in the declaration of the object *c1* in the main method any more.

2. In your main method:

Remove all previous statements in the main method.

a. Declare and create an array of integers named *ids* with the following values: {100, 107, 102, 110, 128}.

b. - Declare and create an object of type *Course* named *c1* as in the following statement:
Course c1 = new Course("Java", 907342, "Asma", ids);
- Print the values of the object *c1* data fields (*name*, *number*, *instructor*) each on a separate line, as follows:
Course name: ....
Course number: ....
Course instructor: ....

- Print the number of students in the course *c1* by invoking the *getNumberOfStudents* method on a separate line, as follows:
Number of registered students: ...

- Invoke the method *printStudentsIDs* for the object *c1* to print students IDs.

Part-3: Reading input from the user:

In this part, you will add code to the no-arg constructor *Course()*, which you defined in the previous part, such that it reads data fields values from the user.

1. Inside the constructor *Course()*:

a. Read the data fields (*name*, *number*, *instructor*) values from the user. Remember to prompt the user with a suitable message for each data field. For example, the following two lines prompt the user to enter the name of the course, read the user input and assign it to the data field *name*:
System.out.println("Please enter course name: ");
name = input.nextLine();

b. Read the number of students, which represents the size of the array *students*, from the user. Use the entered number to create the array *students*. Then iterate over elements in the array to read the values of the array elements (students IDs) from the user.
    a. You must prompt the user by printing appropriate messages to enter the number of students and to enter the ID in each iteration.
    b. The entered number of students must be between [1-60], if the user enters a number that is out of this range, you must keep on prompting the user to enter a valid number between 1and 60 and reading an input from the user.

2. In your main method:

a. Modify the creation of the course object *c1* such that it invokes the no-arg constructor instead of the other constructor. That is, replace the statement:
Course c1 = new Course("Java", 907342, "Asma", ids);
with
Course c1 = new Course();

b. Rerun your code and test it.

Jordan University
Faculty of Engineering and Technology
Department of Computer Engineering

Object-Oriented Problem Solving: CPE 342

Lab-5 Tutorial

Eng. Asma Abdel Karim

❖ Tutorial contents:

- Part I: The Static Modifier.
- Part II: Visibility Modifiers.
- Part III: Creating Packages.
- Part IV: Passing Objects to Methods.
- Part V: Using Classes from the Java Library.

❖ Part I: The Static Modifier.

The following UML diagram represents the *Student* class after adding the Static variables: *maxGrade* and *numOfStudents*, and the Static method *resetNumberOfStudents*.

Student
name: String id: int grades: double[] maxGrade: double numOfStudents: int
Student() Student(n: String, i: int, num: int) Student(n: String, i: int, g: double[]) getAverage(): double printDetails():void resetNumberOfStudents(): void

The *Student* class represented in the above UML diagram is implemented in the file *Student.java* in the folder *Lab5Part1*.

Note the following:

1. The class *Student* is declared public which means that it should be stored in a file named *Student.java*.
2. The data fields *maxGrade*, *numOfStudents* and the method *resetNumOfStudents* are defined static by adding the word static to their definition.
3. The static variable *numOfStudents* gets modified in each constructor such that each time a new object is created it is incremented by 1.
4. When reading the grades from the user in the no-arg constructor, the value entered by the user is checked to be between [0-maxGrade]. If the user enters an invalid value, it keeps on prompting until a valid value is entered.

5. The variable *maxGrade* is printed in the method *printDetails*. **Note that a static member can be accessed from an instance method. Whereas, an instance member cannot be accessed from a static method.**

The *Student* class is tested in the main method implemented in the file *MainClass.java* in the folder *Lab5Part1*. Note how the static variables *maxGrade* and *numOfStudents* are accessed at the beginning of the program even before creating any object. Similarly, the *resetNumOfStudents* method is invoked before creating any object. All static members are accessed using the class name.

Note also that after creating three objects *numOfStudents* will become 3. This value is seen when the variable is printed using the class name or any of the three objects. Note that although it is not recommended, static members are accessible by instances of the class.

❖ Part II: Visibility Modifiers.

The following UML diagram shows the previous *Student* class after encapsulating its data fields, by adding the private modifier to them and providing accessors and mutators. All constructors and methods are declared *public* to be visible from any other class.

Student
- name: String - id: int - grades: double[] - maxGrade: double - numOfStudents: int
+ Student() + Student(n: String, i: int, num: int) + Student(n: String, i: int, g: double[]) + getAverage(): double + printDetails():void + resetNumberOfStudents(): void + setName(n:String): void + getName(): String + setId(i:int): void + getId(): int + setGrades(g: double[]): void + getGrades(): double [] + setMaxGrade(max:double): void + getMaxGrade(): double + getNumOfStudents(): int

The *Student* class represented in the above UML diagram is implemented in the file *Student.java* in the folder *Lab5Part2*.

Note that now that data fields of the *Student* class are all private, they cannot be accessed directly from outside the class. For example, in the main method, writing *s1.name* or *Student.numOfStudents* will cause a compilation error. Similarly, trying to change the values of any of the students objects grades using a statement like this *s2.grades[0]=18* will cause a syntax error as well.

The *Student* class is tested in the main method implemented in the file *MainClass.java* in the folder *Lab5Part2*. In the implemented main method, both instance and static data fields are accessed using their mutator (setter) and accessor (getter) methods.

❖ Part III: Creating Packages.

Benefits of using packages:

1) Packages help the programmer manage the complexity of application programs.
2) Packages facilitate software reuse by enabling programs to import classes from other packages, rather than copying the classes into each program that uses them.

Steps for creating a reusable class in a package are:

1) Declare a public class. If the class is not public, it can be used only by other classes in the same package.
2) Choose a unique package name and add a package declaration to the source code file for the reusable class declaration. In each Java source code file there can be only one package declaration and it must precede all other declarations and statements. Only package declarations, import declarations and comments can appear outside the braces of a class declaration. A Java source code must have the following order: 1. A package declaration (if any), 2. Import declarations (if any), then 3. Class declarations.
   If no package statement is provided, the class is placed in the default package and is accessible only to other classes in the default package that are located in the same directory.
3) Compile the class so that it is placed in the appropriate package directory. When a Java file containing a package declaration is compiled, the resulting class file is placed in the directory specified by the declaration.
4) Import the reusable class into a program and use the class.
   There are two types of import declarations:
   a) Single-type-import declaration in which the import declaration specifies one class to import (e.g. import java.util.Scanner;).
   b) When your program uses multiple classes from the same package, you can import those classes wildcard import (e.g. import java.util.*). Using the asterisk informs the compiler that all public classes from the java.util package are available for use in the program. This is known as a type-import-on-demand declaration. Only the classes from package java.util that are used in the program are loaded by the JVM.

The compiler uses a special object called a *class loader* to locate the classes it needs. The class loader searches for classes in the following order:

a) Search the standard Java classes bundled with the JDK.
b) Search the optional packages. Java provides an extension mechanism that enables new (optional) packages to be added to Java for development and execution purposes
c) Search the classpath, which contains a list of locations in which classes are stored. By default, the classpath consists only of the current directory.

❖ Part IV: Passing Objects to Methods.

You can pass objects and return them from methods. Like passing arrays, passing an object is actually passing the reference of the object. Moreover, returning an object from a method returns the reference of the object. The *Mainclass* implemented in the folder *Lab5Part4* includes examples of:

- A void method (*printStudentDetails*) that takes an object of type *Student* and prints its details.
- A void method (*modifyStudentGrades*) that allows one of the grades of the passed student to be modified by the user.
- A method (*createStudentWithSameGrades*) that takes an object of type *Student* and returns a new *Student* with *name* and *id* read from the user and *grades* copied from the passed *Student* object.

❖ Part V: Using Classes from the Java Library.

- **The *Arrays* class**

The *java.util.Arrays* class contains various *static* methods for sorting and searching arrays, comparing arrays, filling array elements, and returning a string representation of the array. These methods are overloaded for all primitive types.

You can use the *sort* or *parallelSort* method to sort a whole array or a partial array.
For example, the following code sorts an array of numbers and an array of characters.

```
double[] numbers = {6.0, 4.4, 1.9, 2.9, 3.4, 3.5};
java.util.Arrays.sort(numbers); // Sort the whole array
java.util.Arrays.parallelSort(numbers); // Sort the whole array

char[] chars = {'a', 'A', '4', 'F', 'D', 'P'};
java.util.Arrays.sort(chars, 1, 3); // Sort part of the array
java.util.Arrays.parallelSort(chars, 1, 3); // Sort part of the array
```

Invoking *sort*(numbers) sorts the whole array numbers. Invoking *sort*(chars, 1, 3) sorts a partial array from *chars[1]* to *chars[3-1]*. *parallelSort* is more efficient if your computer has multiple processors.

You can use the *binarySearch* method to search for a key in an array. The array must be presorted in increasing order. If the key is not in the array, the method returns *–(insertionIndex+ 1)*. For example, the following code searches the keys in an array of integers and an array of characters.

```
int[] list = {2, 4, 7, 10, 11, 45, 50, 59, 60, 66, 69, 70, 79};
System.out.println("1. Index is " +
       java.util.Arrays.binarySearch(list, 11));
System.out.println("2. Index is " +
       java.util.Arrays.binarySearch(list, 12));

char[] chars = {'a', 'c', 'g', 'x', 'y', 'z'};
System.out.println("3. Index is " +
       java.util.Arrays.binarySearch(chars, 'a'));
System.out.println("4. Index is " +
       java.util.Arrays.binarySearch(chars, 't'));
```

The output of the preceding code is
1. Index is 4
2. Index is -6
3. Index is 0
4. Index is -4

You can use the *equals* method to check whether two arrays are strictly equal. Two arrays are strictly equal if their corresponding elements are the same. In the following code, list1 and list2 are equal, but list2 and list3 are not.

```
int[] list1 = {2, 4, 7, 10};
int[] list2 = {2, 4, 7, 10};
int[] list3 = {4, 2, 7, 10};
System.out.println(java.util.Arrays.equals(list1, list2)); // true
System.out.println(java.util.Arrays.equals(list2, list3)); // false
```

You can use the *fill* method to fill in all or part of the array. For example, the following code fills list1 with 5 and fills 8 into elements *list2[1]* through *list2[5-1]*.

```
int[] list1 = {2, 4, 7, 10};
int[] list2 = {2, 4, 7, 7, 7,  10};
java.util.Arrays.fill(list1, 5); // Fill 5 to the whole array
java.util.Arrays.fill(list2, 1, 5, 8); // Fill 8 to a partial array
```

You can also use the *toString* method to return a string that represents all elements in the array. This is a quick and simple way to display all elements in the array. For example, the following code

```
int[] list = {2, 4, 7, 10};
System.out.println(Arrays.toString(list));
```

displays [2, 4, 7, 10].

- **The *Date* class**

java.util.Date	
+Date()	Constructs a Date object for the current time.
+Date(elapseTime: long)	Constructs a Date object for a given time in milliseconds elapsed since January 1, 1970, GMT.
+toString(): String	Returns a string representing the date and time.
+getTime(): long	Returns the number of milliseconds since January 1, 1970, GMT.
+setTime(elapseTime: long): void	Sets a new elapse time in the object.

You can use the *no-arg* constructor in the *Date* class to create an instance for the current date and time, the *getTime*() method to return the elapsed time since January 1, 1970, GMT, and the *toString*() method to return the date and time as a string. For example, the following code:

```
java.util.Date date = new java.util.Date();
System.out.println("The elapsed time since Jan 1, 1970 is " +
  date.getTime() + " milliseconds");
System.out.println(date.toString());
```

Displays the output like this:

```
The elapsed time since Jan 1, 1970 is 1324903419651 milliseconds
Mon Dec 26 07:43:39 EST 2011
```

The *Date* class has another constructor, *Date(long elapseTime)*, which can be used to construct a Date object for a given time in milliseconds elapsed since January 1, 1970, GMT.

- **The *Random* class**

java.util.Random	
+Random()	Constructs a Random object with the current time as its seed.
+Random(seed: long)	Constructs a Random object with a specified seed.
+nextInt(): int	Returns a random int value.
+nextInt(n: int): int	Returns a random int value between 0 and n (excluding n).
+nextLong(): long	Returns a random long value.
+nextDouble(): double	Returns a random double value between 0.0 and 1.0 (excluding 1.0).
+nextFloat(): float	Returns a random float value between 0.0F and 1.0F (excluding 1.0F).
+nextBoolean(): boolean	Returns a random boolean value.

When you create a *Random* object, you have to specify a seed or use the default seed. A seed is a number used to initialize a random number generator. The *no-arg* constructor creates a Random object using the current elapsed time as its seed. If two Random objects have the same seed, they will generate identical sequences of numbers. For example, the following code creates two Random objects with the same seed, 3.

```
Random random1 = new Random(3);
System.out.print("From random1: ");
for (int i = 0; i < 10; i++)
  System.out.print(random1.nextInt(1000) + " ");

Random random2 = new Random(3);
System.out.print("\nFrom random2: ");
for (int i = 0; i < 10; i++)
  System.out.print(random2.nextInt(1000) + " ");
```

The code generates the same sequence of random *int* values:

```
From random1: 734 660 210 581 128 202 549 564 459 961
From random2: 734 660 210 581 128 202 549 564 459 961
```

The University of Jordan
Faculty of Engineering and Technology
Department of Computer Engineering

Object-Oriented Problem Solving: CPE 342
Lab 5: The Static and Visibility Modifiers + Passing Objects to Methods
Eng. Asma Abdel Karim

❖ **Lab Objectives:**

- To become familiar with the following:
  - ❖ To understand the use of static variables and methods.
  - ❖ To use static methods from the Java library.
  - ❖ To understand visibility modifiers and data field encapsulation.
  - ❖ To pass and return objects from methods.

In this lab, you will continue working on the *Course* class you implemented in Lab4.

1. Place the *Course* class and the *MainClass* in two separate files.
2. Make sure that you have the last implementation of the *Course* class from Lab4.
3. Your main method will be initially empty.

The Course class

The modifications you will apply to the *Course* class are shown in the following UML diagram and explained below:

Course
- name: String - number: int - instructor: String - students: int[] - maxNumOfStudents: int = 60
+ Course() + Course(n: String, num:int, i: String, s: int[]) + getNumberOfStudents(): int + printStudentsIDs(): void + addStudent(): boolean + removeStudent(): boolean + incrementMaxNumOfStudents():void + isRegistered(id: int): boolean + pickRandomStudent(): int + getName(): String + setName(n: String): void + getNumber(): int + setNumber(num: int): void + getInstructor(): String + setInstructor(i: String): void + getMaxNumOfStudents():int + getStudents(): int[]

1. Add the **private** **static** data field *maxNumOfStudents* with default value of 60. This field represents the maximum number of students allowed in a course, and consequently the maximum size allowed for the *students* array.

2. Modify the data fields *name*, *number*, *instructor*, and *students* such that they have **private** visibility.

3. Modify the constructors and methods in your class to have **public** visibility.

4. Add the **public** **static** method *incrementMaxNumOfStudents* which increments the *maxNumOfStudents* by an amount that is read from the user. You must prompt the user to enter the number of students to increase the max with using an appropriate message.

5. Modify the *Course()* constructor such that when reading the number of students (size of array students) from the user, you compare with the static data field *maxNumOfStudents* rather than a fixed value of 60.

6. Modify the constructor *Course(n: String, num:int, i: String, s: int[])* such that it checks the size of the passed array *s* before copying it to the array *students*. If its size is less than or equals *maxNumOfStudents*, it must perform the copying normally. If its size is greater than *maxNumOfStudents*, it must copy only the first *maxNumOfStudents* elements of the array *s* to the array *students*.

7. Modify both constructors such that they sort the students' ids in the *students* array, after initializing it, in ascending order by invoking the *sort* method of the *Arrays* class.

8. Add the public method *isRegistered*, which takes a student id and returns true if he is registered in the course, and false otherwise. The method must search for the passed id in the *students* array. You must use the *binarySearch* method of the *Arrays* class.

9. Add the public method *pickRandomStudent* which returns the id of a student picked randomly from the *students* array.

10. Modify the method *addStudent* such that it does not add the id to the array *students* if:
    - The size of the array is *maxNumOfStudents*. I.e. it must only perform the addition if the size of the array is less than *maxNumOfStudents*.
    - If the entered student is already registered in the course (his id already exists in the array). This check must be done by invoking the *isRegistered* method. If the student is already registered, it must not add the id and print the message "*Student is already registered*".

    The method must be modified to:
    - Return a *boolean* whose value is *true* if the addition is performed and false otherwise.
    - Sort the students array again, if a new student id is added successfully, by invoking the *sort* method of the *Arrays* class again

11. Add the mutators (setters) and accessors (getters) of the data fields *name*, *number*, *instructor*, and the getter of *maxNumOfStudents*.

12. Add the accessor method (getter) for the *students* array. Note that this method must not return the actual reference of the array students, it must return a reference to a copy of it instead.

13. Add the method *prinDetails* which prints the course information as follows:

Course name: ….
Course number: …..
Instructor name: …..
Number of registered students: ..,,/ max
Students IDs: …….

The method must print the students IDs by invoking the *printStudentsIDs* method.

The Main class

In your main class define the following methods:

a. A method named *printCourseDetails* that takes an object of type *Course* and prints all its details as follows:

course_number – course_name
Instructor name: ……
Number of registered students: …..
Students IDs:
.. ..

Note: try printing students IDs once by invoking the *printStudentsIDs* method, and once without invoking it by iterating over the *students* array.

b. A method named *operateCourse* that takes an object of type course and continuously prints the following menu and takes the number that represents the user choice to know what he wants to do:
What to do :

1. ***Print course info*** (Invoke the *printDetails* method)
2. ***Add student***
   (Invoke the *addStudent* method, if the addition fails print error msg "*Could not add student*" to the console)
3. ***Remove student***
   (Invoke the *removeStudent* method, if the removing fails print error msg "*Could not remove student*" to the console)
4. ***Select random student***
   Select a random student from the students array and print the randomly selected ID, by invoking the *pickRandomStudent* method, as follows:
   Randomly selected ID: ….
5. ***Increment maximum number of students***
   (Invoke the *incrementMaxNumOfStudents* method)
6. ***Exit:*** return to main method.

c. In your main method:
   - Create a course object by invoking the *no-arg* constructor of the *Course* class.
   - Invoke the method *printCourseDetails* to print the course details.
   - Invoke the method *operateCourse* to continuously operate the course by taking input from the user.

Jordan University
Faculty of Engineering and Technology
Department of Computer Engineering

Object-Oriented Problem Solving: CPE 342

Lab-6 Tutorial

Eng. Asma Abdel Karim

❖ Tutorial contents:

- Part I: Arrays of Objects.
- Part II: Object Composition.
- Part III: Using Classes from the Java Library.

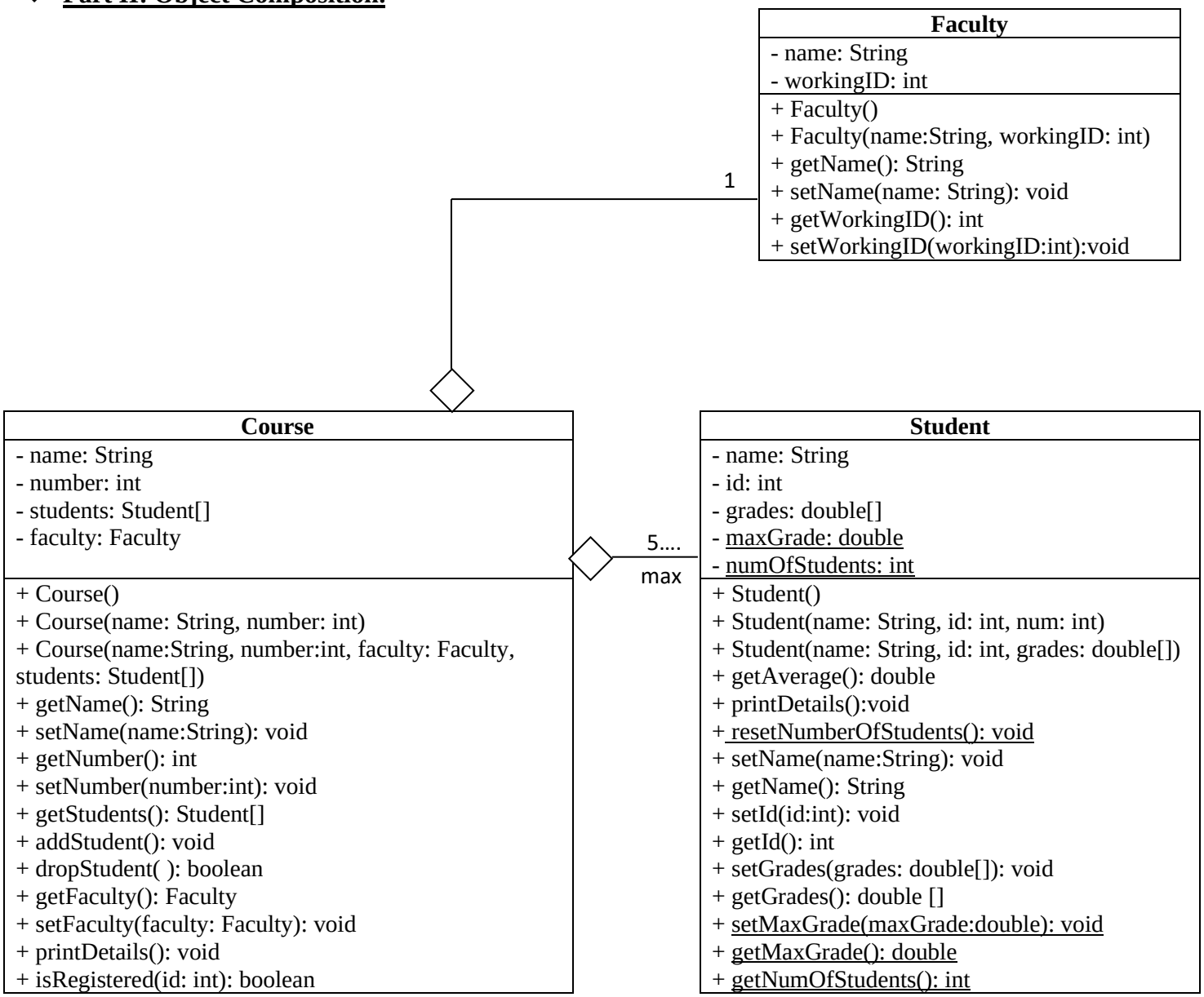
❖ Part I: Array of Objects.

The following is the UML diagram of the *Student* class we implemented in the previous lab tutorial.

Student
- name: String - id: int - grades: double[] - maxGrade: double - numOfStudents: int
+ Student() + Student(n: String, i: int, num: int) + Student(n: String, i: int, g: double[]) + getAverage(): double + printDetails():void + resetNumberOfStudents(): void + setName(n:String): void + getName(): String + setId(i:int): void + getId(): int + setGrades(g: double[]): void + getGrades(): double [] + setMaxGrade(max:double): void + getMaxGrade(): double + getNumOfStudents(): int

The *MainClass* of *Lab6Part1* includes three methods: *main*, *printStudentsGrades*, and *getStudentWithMaxAverage*. The main method invokes the *printDetails* method of the *Student* class on the array elements. It also invokes the *printStudentsGrades* method by passing the *students* array to print grades of its *Student* objects, and invokes the *getStudentWithMaxAverage* method by passing the *students* array to get the *Student* object with the maximum average and print its name..

❖ Part II: Object Composition.



The files that implement these classes are attached with this lab tutorial in *Lab6Part2*. Study them and note how to implement object composition using one object and using an array of objects. Note the use of the *this* keyword in the constructors and mutators of these classes. The MainClass includes methods that operate on one *Course* object and arrays of *Course* objects.

❖ Part III: Using Classes from the Java Library.

- The Character Class

For convenience, Java provides the following methods in the *Character* class for testing characters as:

Method	*Description*
`isDigit(ch)`	Returns true if the specified character is a digit.
`isLetter(ch)`	Returns true if the specified character is a letter.
`isLetterOfDigit(ch)`	Returns true if the specified character is a letter or digit.
`isLowerCase(ch)`	Returns true if the specified character is a lowercase letter.
`isUpperCase(ch)`	Returns true if the specified character is an uppercase letter.
`toLowerCase(ch)`	Returns the lowercase of the specified character.
`toUpperCase(ch)`	Returns the uppercase of the specified character.

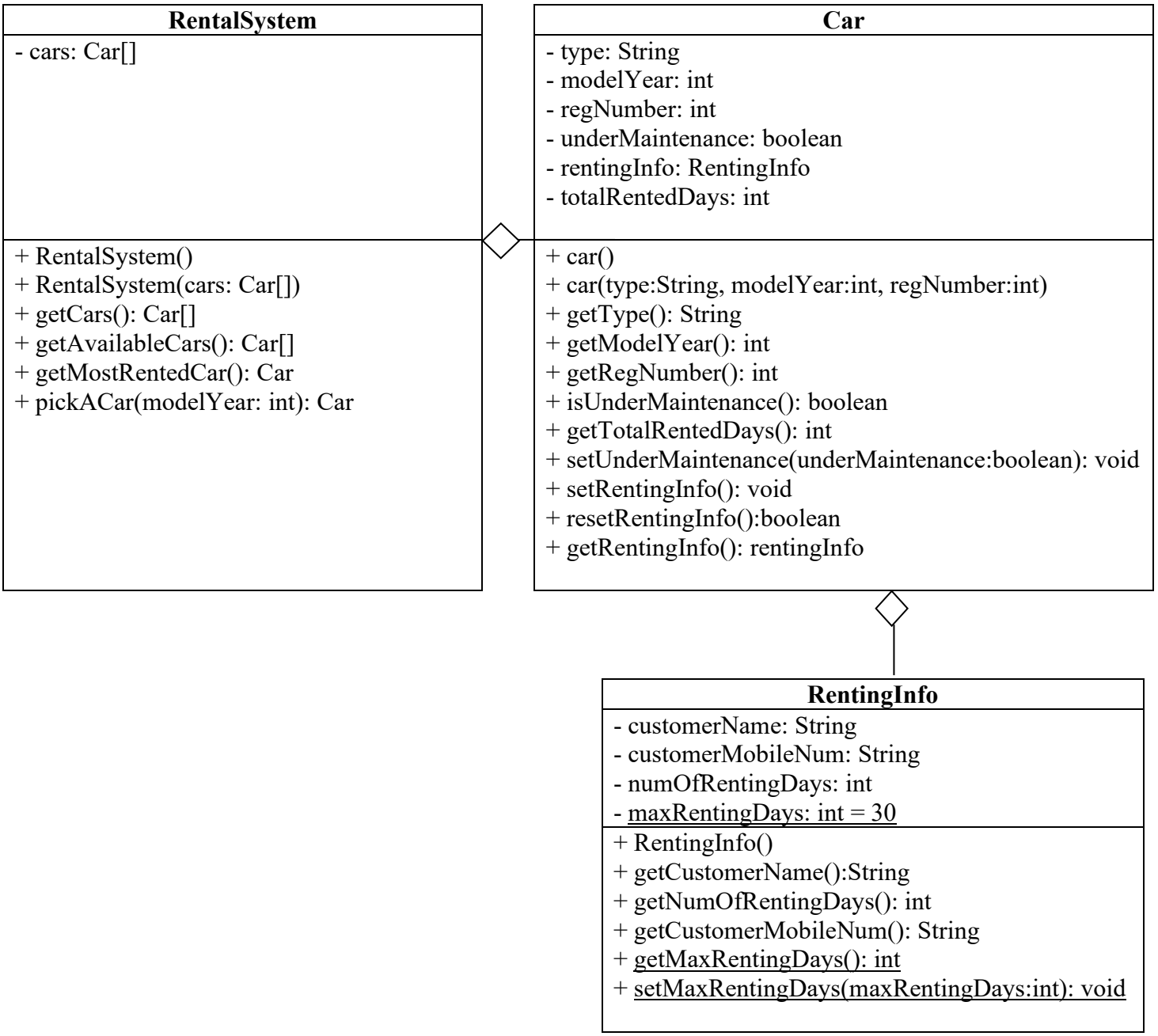
The University of Jordan
Faculty of Engineering and Technology
Department of Computer Engineering

Object-Oriented Problem Solving: CPE 342
Lab 6: Object Composition and Arrays of Objects
Eng. Asma Abdel Karim

❖ Lab Objectives:

- To understand the concept of object composition.
- To understand how to create and use arrays of objects.

❖ Given the *RentingInfo* class implementation, implement the following classes named *Car* and *RentalSystem* with the following UML diagram:



1. **For the *Car* class:**

- The *underMaintenance* (تحت الصيانة) data field must be always initialized to false, and the *totalRentedDays* data field must be always initialized to 0.
- The *rentingInfo* data field must be always initialized to null. This field indicates whether the car is rented or not, if it contains null then the car is not rented. Otherwise, if it contains a reference, this means that the car is rented, and the reference is to a *RentingInfo* object that includes info of the car rental.
- The *no-arg* constructor must initialize the *type*, *modelYear*, *regNum* by reading their values from the user.
- The constructor *car(type:String, modelYear:int, regNumber:int)* must initialize the *type*, *modelYear*, *regNum* with the corresponding parameters values.
- The *setRentingInfo* method is invoked when a car is to be rented to set the renting info of the car by creating a new *rentingInfo* object using the *no-arg* constructor and assign it to *rentingInfo*.
- The *resetRentingInfo* method is invoked whenever a car is returned from rent to reset the renting info of the car to indicate that it's not rented and update the total renting days. It must first check that the car is currently rented (*rentingInfo* data field is not null). If so, it must add the number of renting days of the *rentingInfo* data field to the *totalRentedDays* of the car, then it must set the *rentingInfo* data field to null and return true. Otherwise, if the *rentingInfo* is already null, the method must return false.

2. **For the *RentalSystem* class:**

- The *no-arg* constructor must ask the user to enter the number of cars in the system, then create the *cars* array accordingly. Then, create the car objects using the *no-arg* constructor of the *Car* class.
- The constructor *RentalSystem(cars: Car[])* must initialize the *cars* array using the passed array by creating a new array and new object for each element.
- The *getCars* method must return the *cars* array reference.
- The *getAvailableCars* method must return an array that includes cars in the system that are available for renting. In order for a car to be available for renting, it must be unrented and not under maintenance.
- The *getMostRentedCar* method must return a reference to the car that has been most rented: the car with the maximum total rented days.
- The *pickACar* method must take a model year, then pick a car randomly with a model year that is newer (greater) than the passed year.

3. **In your main class:**

- **Define a method named *printCar* that takes a car object and prints its details as follows:**
  Car type: ....
  Car registration number: ...
  Car model year: ...

  Then if the car is under maintenance the method must print on a new line: *Car is under maintenance*. If not, it must print: *Car is not under maintenance*.

  Then based on the value of *rentingInfo* of the car:
  - If the car is not rented it must print: *The car is not rented*.
  - If the car is rented it must print: The car is rented to (customer name in *rentingInfo*) with mobile number (customer mobile number in *rentinInfo*) for (number of renting days in *rentingInfo*).

- **Define a method named *searchCars* that** takes an integer, that represents a registration number, and an array of cars. The method searches the array for a car with the required registration number. It returns a reference to the car object that has the entered registration number or null if the car is not found.

- **In your main method:**

- Create a *RentalSystem* object using the *no-arg* constructor of the *RentalSystem* class.
- Continuously display the following menu to the user and take the number that represents the user choice to know what he wants to do:
  What to do :

1. ***Rent a car****:* this option must read a registration number from the user, and invoke the *searchCars* method to search for the car with the entered registration number among the **<u>available</u>** cars (invoke *getAvailableCars*()).
   a. If no car with the entered registration number is found, the following message must be printed "*No available car with entered registration number*".
   b. If a car is found, it must invoke the method *setRentingInfo* on the found car.
2. ***Return a car****:* this option must take the registration number of the car to be returned from the user and invoke the *searchCars* method to search for the car with the entered registration number in all the rental system cars (invoke *getCars*()).
   a. If no car with the entered registration number is found, the following message must be printed "*No car with entered registration number*".
   b. If a car is found, it must invoke the method *resetRentingInfo* on the car. If the method returns false, the following message must be printed "Weird! Car is not rented!".
3. ***Print most rented car****:* this option must invoke the *getMostRentedCar* method of the rental system. Then print the details of the car by invoking the *printCar* method.
4. ***Print all available cars****:* this option must invoke the *getAvailableCars* method, and print the available cars info in a table as follows:
   Car Registration Number Car Type Car Model Year Total Rented days
   .
   .
   .
5. ***Pick based on model year****:* this option must read the model year (above which a car is to be picked) and invoke the *pickACar* method, then print the details of the returned car by invoking the *printCar* method.
6. ***Exit****.*

Jordan University
Faculty of Engineering and Technology
Department of Computer Engineering

Object-Oriented Problem Solving: CPE 342

Lab-7 Tutorial

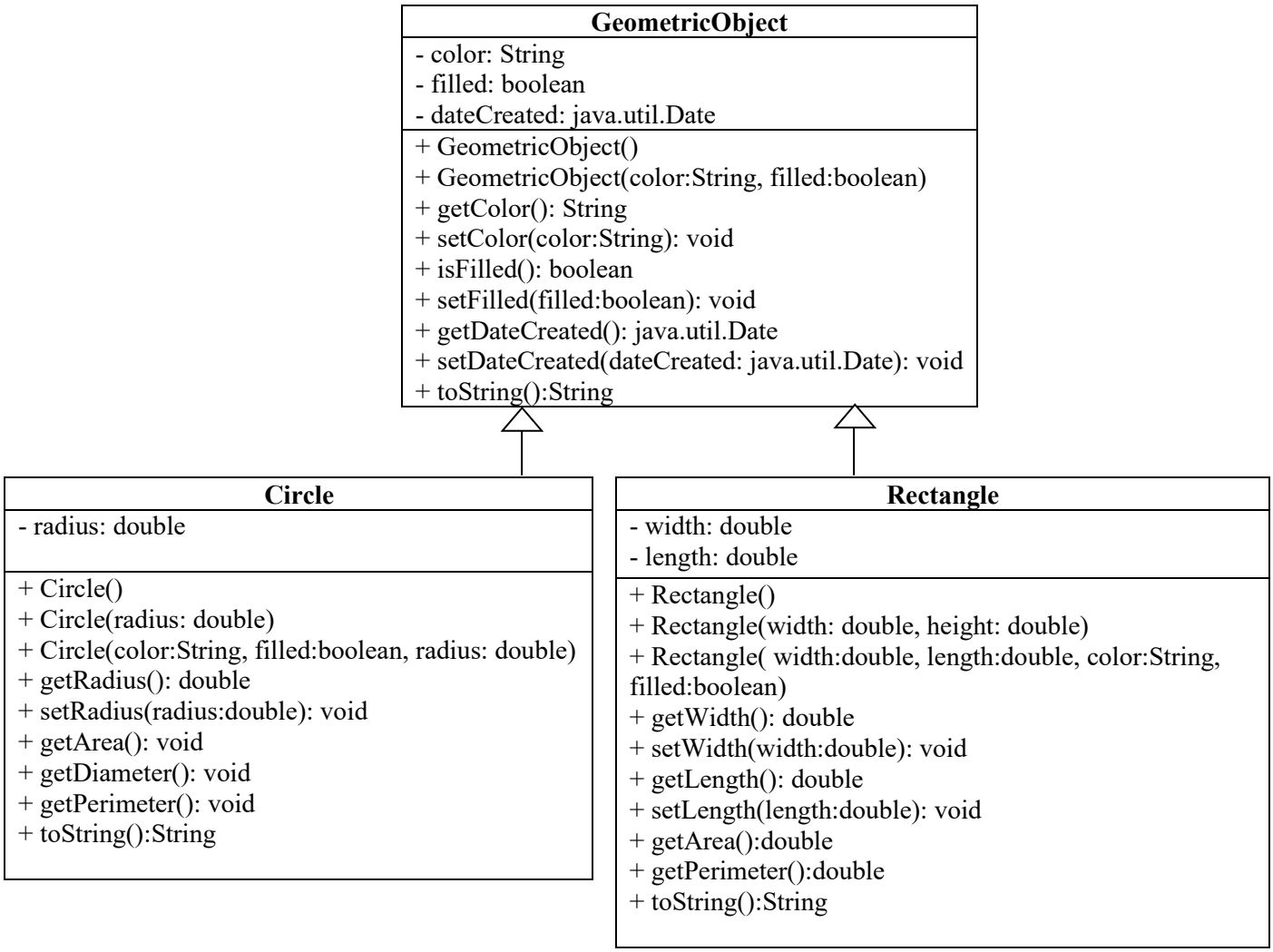
Eng. Asma Abdel Karim

❖ Tutorial contents:

- Part I: Inheritance + super keyword.
- Part II: Using dialog boxes from the JOptionPane class of javax.swing.
- Part III: Polymorphism and objects casting.

❖ Part-I: Inheritance + super keyword.

In this lab tutorial, we will implement the inheritance hierarchy shown in the following UML diagram:



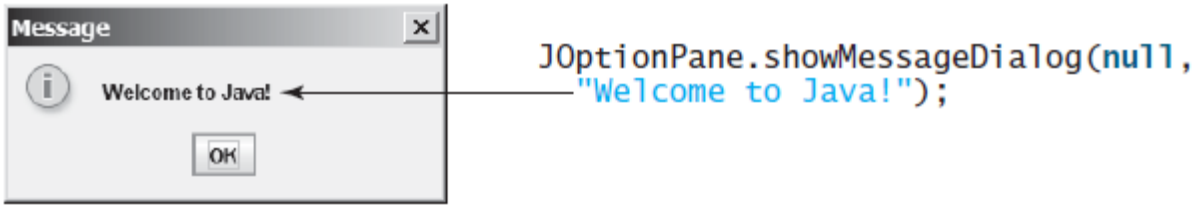
The implementation of classes in this UML diagram along with a test that includes creating objects of the three types and printing the *toString* method of the created objects, are included in the folder Lab7-Part1.

❖ **Part II: Using dialog boxes from the JOptionPane class of javax.swing.**
❖ **Part II-A: Displaying info in a dialog box.**

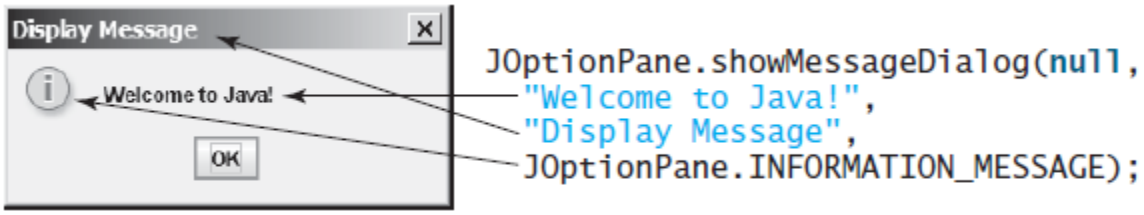
In order to display text in a message dialog box, you need to use the *showMessageDialog* method in the *JOptionPane* class. *JOptionPane* is defined in the *javax.swing* package, so you need to import it in your code. Note that the method *showMessageDialog* is static, and hence can be invoked using the class name (*JOptionPane*).

There are several ways to use the *showMessageDialog* method. Two of these ways are:

1) *JOptionPane.showMessageDialog(null, x);*
   The first argument can always be null and x is a string for the text to be displayed.
   Example:



2) *JOptionPane.showMessageDialog(null, x, y, JOptionPane.INFORMATION_MESSAGE);*
   As in the first way, the first argument is null, x is a string for the text to be displayed, y is a string for the title of the message box, the fourth argument can be JOptionPane. INFORMATION_MESSAGE which causes the information icon to be displayed in the message box.
   Example:



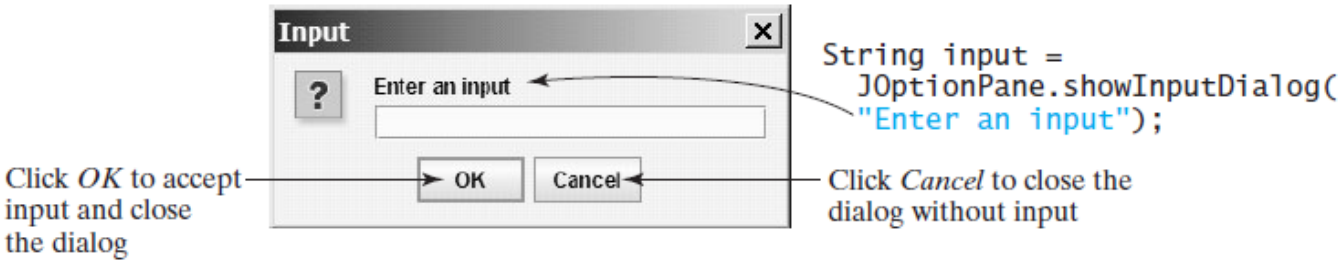
In Lab7-Part2, the main method is rewritten to print the *toString* method of the objects in message dialog boxes.

❖ **Part II-B: Reading values from input dialog boxes.**

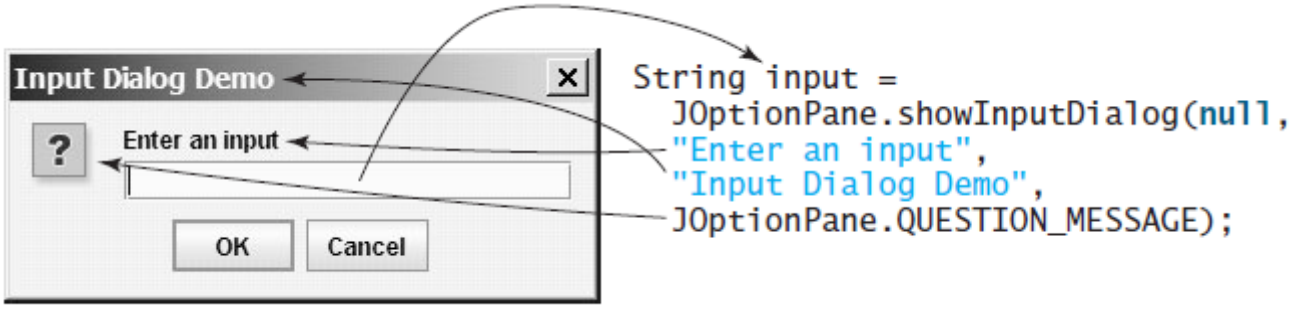
An input dialog box prompts the user to enter an input graphically. You can obtain input from an input dialog box by invoking the *JOptionPane.showInputDialog* methods. The input is returned from the method as a string. There are several ways for using the *showInputDialog* method, two of these ways are:

1) *String S = JOptionPane.showInputDialog(x);*
   where x is a string for the prompting message.

   Example:



2) *JOptionPane.showInputDialog(null, x, y, JOptionPane.QUESTION_MESSAGE);*
where x is a string for the prompting message, and y is a string for the title of the input dialog.
Example:



The input returned from the input dialog box is a string. If you enter a numeric value such as 123, it returns "123". You have to convert a string into a number to obtain the input as a number.

To convert a string into an int value, use the *Integer.parseInt* method as follows:
int intValue = Integer.parseInt(intString); //intString is a numeric string such as "123"

To convert a string into a double value, use the *Double.parseDouble* method as follows:
double doubleValue = Double.parseDouble(doubleString); //doubleString is a numeric string such as "12.3"
The *Integer* and *Double* classes are both included in the *java.lang* package, and thus are automatically imported.

In Lab7-Part2, the no-arg constructor of each of the three classes is modified such that it prompts the user to enter the value of each data field and initializes it with the entered value using an input dialog box. Note that code written in the *GeometricObject* class to initialize the data fields *color* and *filled* does not have to be re-written in the subclasses *Circle* and *Rectangle*. Constructors of the subclasses should invoke the super-class constructor to initialize these data fields.

❖ Part-III: Polymorphism and objects casting.

Remember that polymorphism means that a variable of a supertype can refer to a subtype object.

Create a class that contains your main method (i.e. *MainClass*), then write a code that performs the following:

1. In your *main* method:
   a. Declare an array of three objects of type *GeometricObject* named *myArray*.

   b. Create objects of the array such that:
   - The first object's actual type is *GeometricObject* whose *color* is red and is not filled.
   - The second object's actual type is *Circle* whose *color* is blue, is filled and its *radius* is 5.
   - The third object's actual type is *Rectangle* whose *color* is red, is not filled, with *width*=3 and *height*=4.

   c. Write a for loop that prints the details of each object in the array by invoking its *toString* method. Observe the output of the print statements and justify how signature matching and dynamic binding work in this case.

2. Define a new method named *printArea* with the following header:

   public static void printArea(GeometricObject g)

a. Try to print the area of the passed object by invoking the getArea method (g.getArea()), and justify the resulting syntax error.

b. Now remove the print statement you wrote in the previous step, and use the *instanceof* operator to check whether the actual type of the passed object is a *Circle* or a *Rectangle*. Then, cast the object into its actual type in order to invoke the *getArea* method.

   The following code checks whether the passed object is a circle and invokes the *getArea* method after down-casting the passed object:

```
if (g instanceof Circle){
    System.out.println("Circle area = "+((Circle)g).getArea());
}
```

c. In your main method, add a for loop that invokes the *printArea* method by passing elements of the array you created, one element in each iteration. Observe and justify the output.

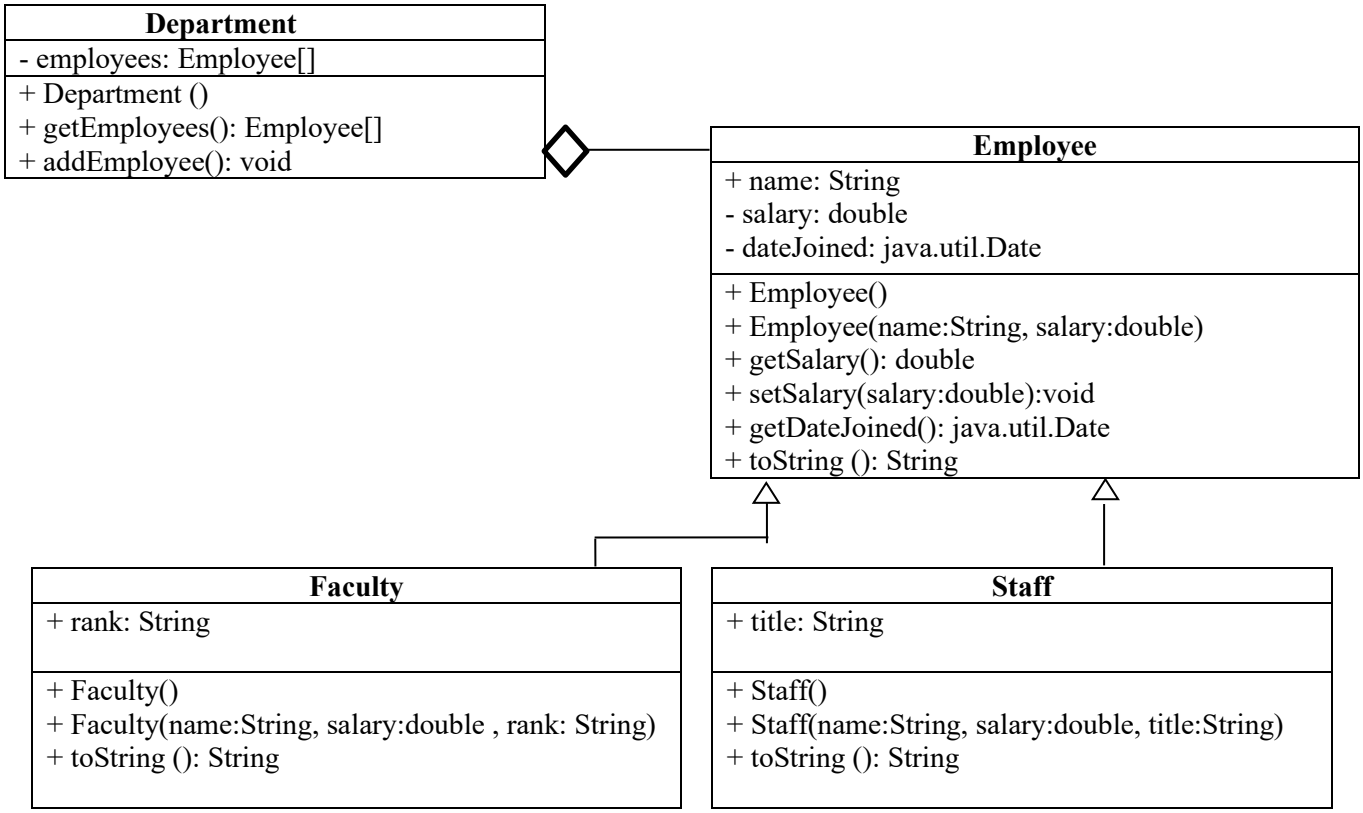
The University of Jordan
Faculty of Engineering and Technology
Department of Computer Engineering

Object-Oriented Problem Solving: CPE 342
Lab 7
Eng. Asma Abdel Karim

❖ Lab Objectives:

- Define classes that extend other classes to form the inheritance relationship.
- Override methods of the superclass in its subclasses.
- Use the super keyword to call constructors and overloaded methods of the superclass.
- Apply the concept of polymorphism and objects' casting.

❖ Implement the following classes using Java:



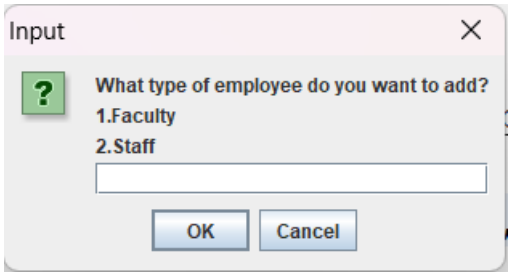
- **In the *Employee, Staff, Faculty* classes:**

- The *no-arg* constructors of the three classes should obtain initial values of data fields using input dialog boxes by prompting the user with proper messages to enter required values. Note that you MUST NOT re-read inherited data fields values in the sub-classes' constructors; you must invoke the super-class constructor instead.
- The constructors *Faculty(name:String, salary:double , rank: String)* and *Staff(name:String, salary:double, title:String)* must initialize inherited data fields by invoking the constructor *Employee(name:String, salary:double)*.

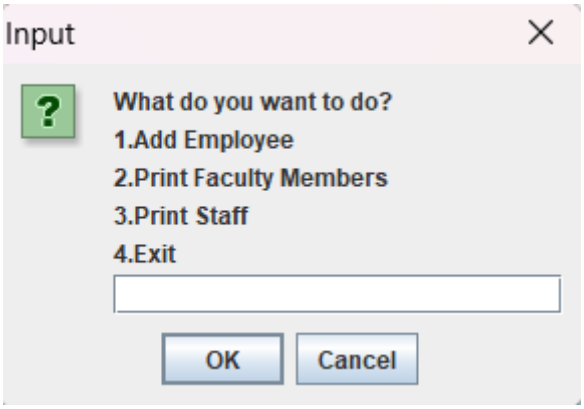
- The *toString()* method of the three classes should return a *String* that contains the object's details as follows:
  For the *Employee* class it should return: *Name is: ……, Salary: ……., Date Joined:……..*
  For the *Faculty* class it should return:
  Name is: ……., Salary: ……., Date Joined:……..
  Rank:……..
  For the *Staff* class it should print:
  Name is:……., Salary: ……. Date Joined:……..
  Title:……..
  You must not re-form the *String* to include inherited data fields of the *Faculty* and *Staff* classes. You must invoke the *toString* method of the superclass instead.

- **In the *Department* class:**

- The array *employees* must be initialized to be empty.
- The *addEmployee* method must ask the user in an input dialog box about the type of employee he wants to add as in the following figure, then based on the user input (1 or 2) either add a faculty or staff by creating a new object using the no-arg constructor. If the user enters a wrong input (neither 1 nor 2), it must display the message "Failed to add employee: wrong input!" in a message dialog box and return from the method. If the user presses Cancel, the method must return.



- **In your Main Class:**
  1. Create an object of type *Department* named *myDepartment*.
  2. Continuously display the following menu to the user, and based on his input (number) invoke the appropriate method from the *Department* class on *myDepartment*.



  1. Add Employee: invokes the *addEmployee* method on *myDepartment*.
  2. Print Faculty Members: print details of all faculty objects in *myDepartment* in a message dialog box by invoking the *toString* method for each object on a separate line.
     The following is a sample message dialog box that will be displayed for this option:

3. Print Staff: print details of all staff objects in *myDepartment* in a message dialog box by invoking the *toString* method for each object on a separate line. (Similar to the previous option)
4. Exit: exit the program.

If the user enters any other invalid number, the following message dialog box must be displayed, and the program must return to the main menu and take a new input:

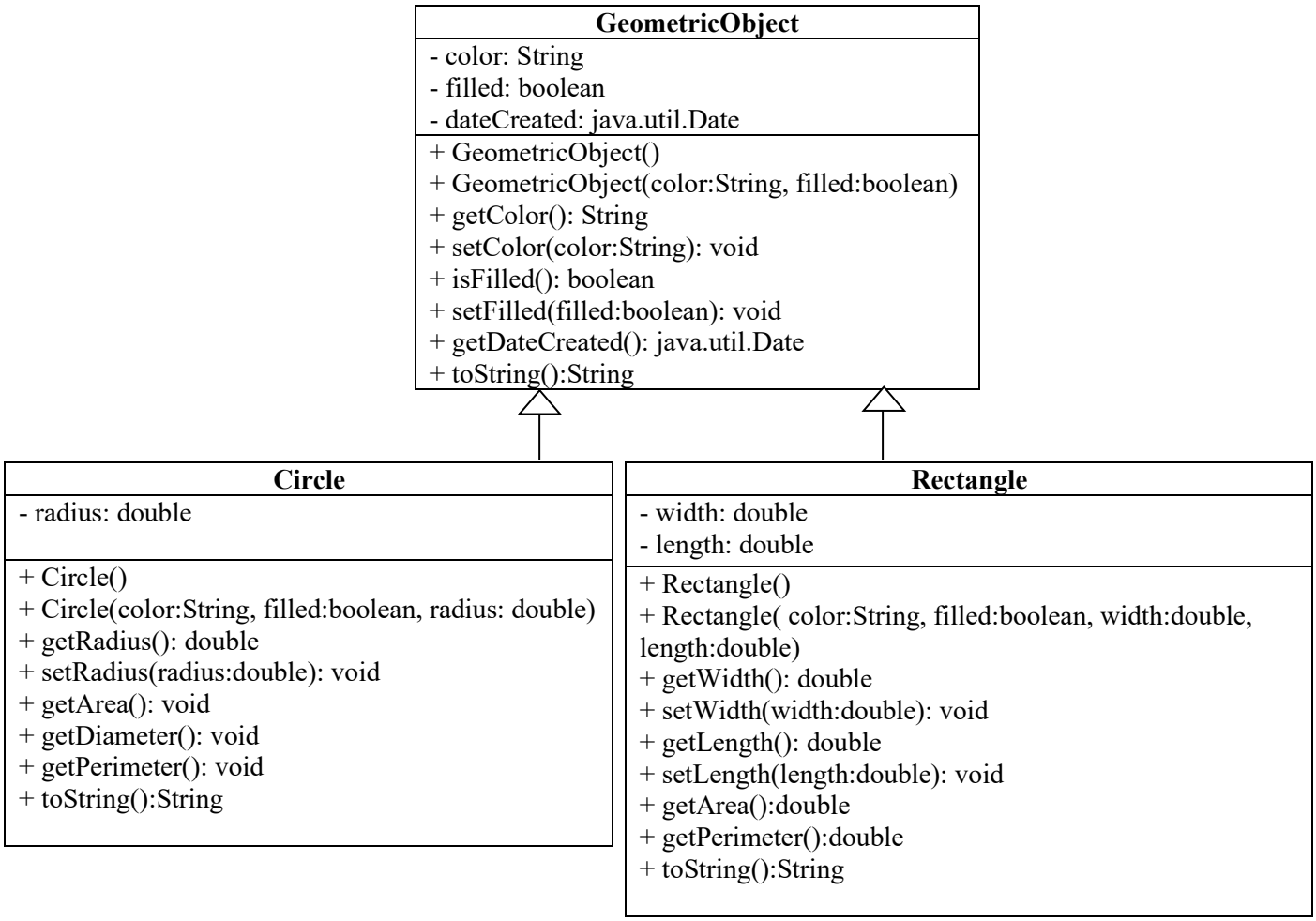
Jordan University
Faculty of Engineering and Technology
Department of Computer Engineering

Object-Oriented Problem Solving: CPE 342

Lab-8 Tutorial

Eng. Asma Abdel Karim

In this lab tutorial, we will use the classes we implemented in lab7 tutorial: *GeometricObject*, *Circle*, and *Rectangle*, which are shown in the following UML class diagram.



❖ Part I: Overriding the Object's equals method.

The following shows an example of an overridden implementation of the *Object*'s *equals* method in the *GeometricObject* class which checks whether two objects are equal based on their *color*.

```
@Override
  public boolean equals (Object O){
     if (O instanceof GeometricObject)
        return color.equals(((GeometricObject)O).color);
     else return false;
  }
```

1. Add the previous implementation of the *equals* method to the *GeometricObject* class.

2. Add an implementation that overrides the *equals* method in the *Circle* class by checking whether two circles are equal based on their *radii.*

3. Add an implementation that overrides the *equals* method in the *Rectangle* class by checking whether two rectangles are equal based on their *width* and *length*. (1 point)

4. In your main method:

1. In your *main* method:
   a. Declare an array of three objects of type *GeometricObject* named *myArray*.

   Create objects of the array such that:
   - The first object's actual type is *GeometricObject* whose *color* is red and is not filled.
   - The second object's actual type is *Circle* whose *color* is blue, is filled and its *radius* is 5.
   - The third object's actual type is *Rectangle* whose *color* is red, is not filled, with *width*=3 and *height*=4.

   b. Create a new *Circle* object named *myCircle* whose *color* is red, is filled and its *radius* is 5.

   c. Print the output of invoking the equals method by passing the *Circle* object you created as a parameter (i.e. *myArray[i].equals(myCircle)*).

   d. Print the output of invoking the equals method by passing elements of the array *myArray* as a parameter (i.e. *myCircle.equals(myArray[i])*).

   e. Observe and justify the difference in the output in the last two steps.

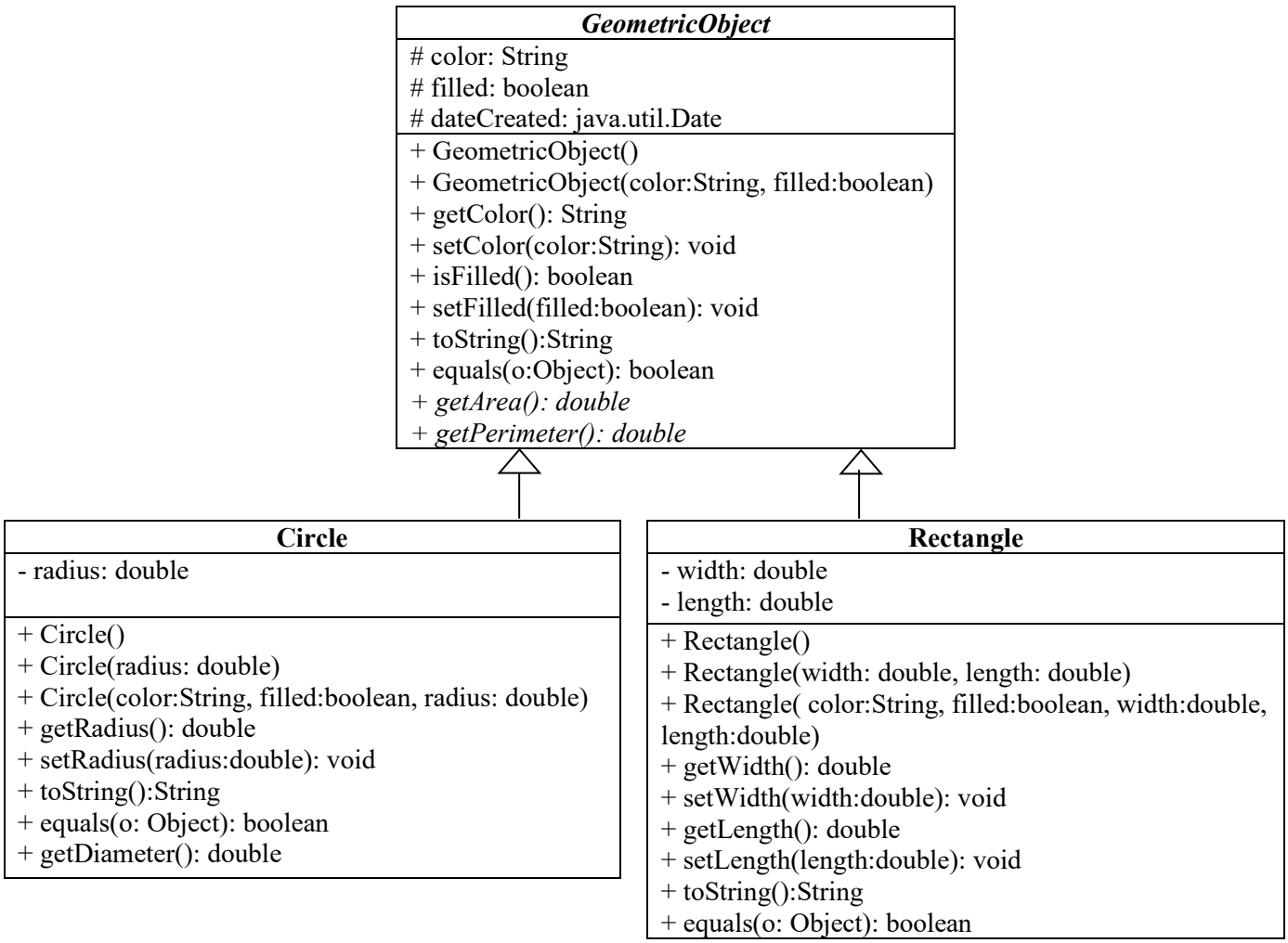
❖ Part II: Protected class members.

Protected class members can be accessed only from inside the class and inside its subclasses. The following is the UML class diagram of the *GeometricObject* class with its *color* and *filled* data fields defined protected.

GeometricObject
color: String # filled: boolean # dateCreated: java.util.Date
+ GeometricObject() + GeometricObject(color:String, filled:boolean) + getColor(): String + setColor(color:String): void + getDateCreated(): java.util.Date + isFilled(): boolean + setFilled(filled:boolean): void + toString():String

1. Change the visibility of the data fields *color*, *filled*, and *dateCreated* in the *GeometricObject* class to *protected*.
2. Try to access these data fields from any method in the subclasses *Circle* and *Triangle* and observe the change in its visibility.

❖ Part III: Abstract classes & Methods.



The inheritance hierarchy above is implemented in the attached files (Part-3&4).

Note that the *GeometricObject* class should be defined abstract since it contains abstract methods *getArea()* and *getPerimeter()*. The *getPerimeter* and *getArea* methods must be implemented in the subclasses *Circle* and *Rectangle* since they are concrete.

❖ Part IV: The ArrayList Class.

Arrays can be used to store objects, but once the array is created, its size is fixed. Java provides the *ArrayList* class, which can be used to store an unlimited number of objects. The following table compares between using arrays and *ArrayLists* to store objects.

Operation	*Array*	*ArrayList*
Creating an array/ArrayList	`String[] a = new String[10]`	`ArrayList<String> list = new ArrayList<>();`
Accessing an element	`a[index]`	`list.get(index);`
Updating an element	`a[index] = "London";`	`list.set(index, "London");`
Returning size	`a.length`	`list.size();`
Adding a new element		`list.add("London");`
Inserting a new element		`list.add(index, "London");`
Removing an element		`list.remove(index);`
Removing an element		`list.remove(Object);`
Removing all elements		`list.clear();`

In your *MainClass*, create an *ArrayList* of *GeometricObject* objects and add a number of circles and rectangles to the *ArrayList* using the no-arg constructor. Note that an array can be retrieved from the *ArrayList* object using the *toArray* method.

The *ArrayList* class includes an *Iterator* object which allows removing elements from the *ArrayList* while iterating over its elements. *Iterator* is an interface defined in the *java.util* package, and hence it should be imported as follows: *import java.util.Iterator*; In order to get the iterator of an *ArrayList* object, we can use the following statement:

Iterator i = arrayListObject.iterator();

Afterwards, the following methods of the Iterator interface can be used:
1. *hasNext()*: returns a *boolean* which indicates whether the *ArrayList* has more elements.
2. *next()*: returns the next element in the *ArrayList*. Note that this method returns an Object, so you should cast the returned object in the specific type of which the *ArrayList* is constructed.
3. *remove()*: removes the last element returned by this iterator.

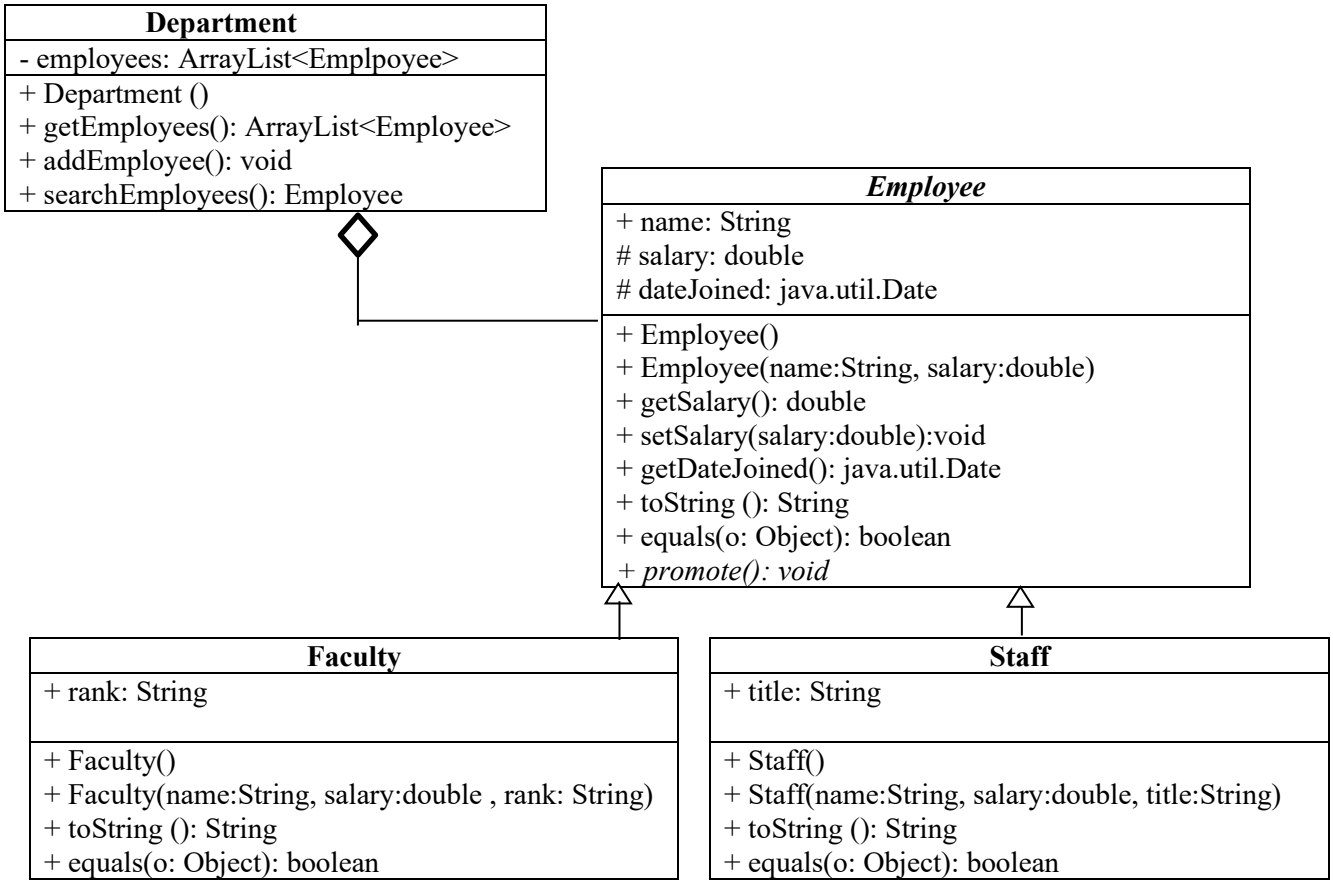
The University of Jordan
Faculty of Engineering and Technology
Department of Computer Engineering

Object-Oriented Problem Solving: CPE 342
Lab 8
Eng. Asma Abdel Karim

❖ Lab Objectives:

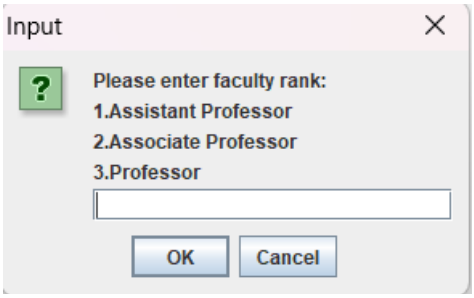
- Override the *equals* method of the *Object* class.
- Use the *ArrayList* class to define collections of objects.
- Define *protected* class members.
- Apply the concept of *abstract* classes and methods.

❖ Use the classes *Employee*, *Staff*, *Faculty, Department,* and *Main* that you implemented in Lab7, and perform the following:



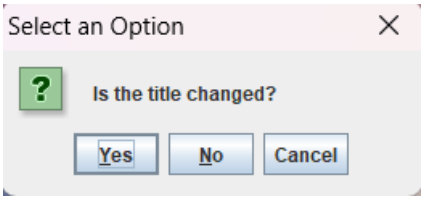
- Modify the visibility of *salary* and *dateJoined* in the *Employee* class to *protected*.
- Override the *equals* method in the *Employee* class such that two employees are equal if they have the same salary.
- Override the equals method in the *Faculty* class such that two faculty members are equal if they have the same rank. If the passed object is an employee but not a faculty member, the *equals* method of the superclass *Employee* must be invoked. If the passed object is neither a faculty nor an employee, it returns false.
- Override the equals method in the *Staff* class such that two staff members are equal if they have the same salary and the same title. If the passed object is not a staff, it returns false.

- Define the abstract method *promote* in the *Employee* class as shown in the UML diagram. Note the syntax error that occurs and understand why it occurs. Then, re-define the *Employee* class to be abstract.
- Implement (override) the *promote* method in the *Faculty* class such that it:
    1. Changes the rank of the faculty as follows:
        - If the current rank is “Assistant Professor”, it must be changed to “Associate Professor”.
        - If the current rank is “Associate Professor”, it must be changed to "Professor”.
        - If the current rank is “Professor” it must print the following message in a message dialog box: “The faculty already has the highest rank”.
    2. Asks the user about the amount the salary will be incremented with (in an input dialog box) and adds the entered amount to the faculty salary.
- Modify the *Faculty()* constructor such that it asks the user to enter the rank in an input dialog and takes the choice as a number as follows:



If the user enters any choice less than 1 or greater than 3, it must keep on displaying the same input dialog box.

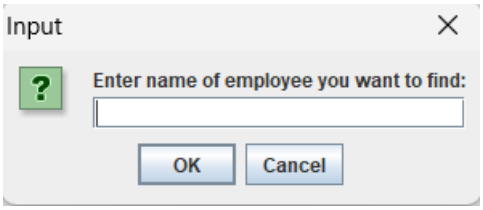
- Implement (override) the *promote* method in the *Staff* class such that it:
    1. Asks the user if the title is to be changed using a confirm dialog box. If the user selects “Yes”, it must ask him about the new title in an input dialog box. If the user selects “No” or “Cancel”, it must not perform anything.



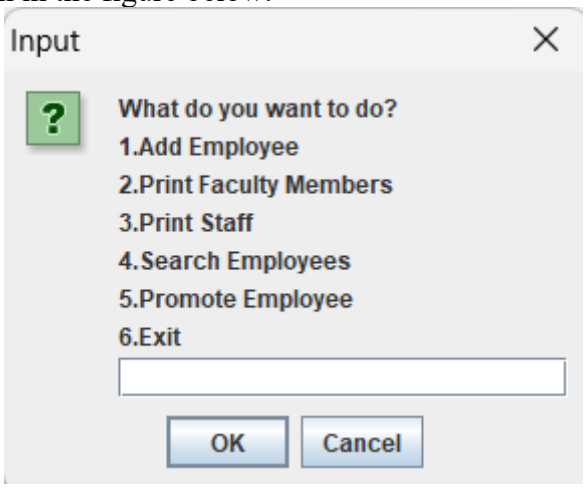
    2. Asks the user about the amount the salary will be incremented with (in an input dialog box) and adds the entered amount to the staff salary.

- **In the *Department* class:**

- Modify *employees* to be an *ArrayList* and initialize it to be empty.
- Modify *getEmployees* and *addEmployee* to work on an *ArrayList* of *employees*.
- The *searchEmployees* method must ask the user about the name of the employee he wants to find as in the following figure, then search the *employees ArrayList* for the entered name. If an employee with the entered name is found, it must return it. If it is not found it must return null.



- **In your main method:**
  - Modify case2 (Print faculty members) and case3 (Print staff) according to the change in *employees* to *ArrayList*.
  - Modify the menu as shown in the figure below:



    - *Search Employees*: invokes the *searchEmployees* method on *myDepartment*. If the method returns null, it must print the message "*Could not find employee with the specified name!*" in a message dialog box. If not, it must print the *String* returned by the *toString* method for the returned *Employee* object in a message dialog box.
    - *Promote Employee*: invokes the *searchEmployees* method of the department class on *myDepartment* then invoke the *promote* method on the returned employee. If *searchEmployees* returns null, it must print the message "*Sorry no employee with entered name!*" in a message dialog box.