Lab 03 Computer Science 1027B Summer 2025

Introduction

The purpose of these labs is to provide practice problems for you to get used to the concepts being covered in class. It is very important that you do these labs on your own without relying on available AI tools. Questions on exams are often very similar to exercises presented in these labs, so understanding them and solving these problems gives you a huge advantage for the exams. Make sure that you complete both the code as well as provide written answers to the blue questions within the boxes.

When you have concluded this lab, you should submit the following files to OWL Brightspace:

- MathUtilities.java
- UserAccount.java
- Geometry.java
- written_answers.txt

Topics Covered

- The **static** and **final** keywords
- Method overloading

Exercise 01

Let's create a class '**MathUtilities**' that is responsible for providing a set of static utility methods and constants for common mathematical operations (you cannot use methods from the Math class in your solution). This class will include the following **constant**:

$\boldsymbol{\pi}$ (**PI**): A 'static final' field representing the mathematical constant π (pi) initialized to 3.14159 (a double).

As well as the following **methods**:

- gcd(int a, int b): A static method to calculate the greatest common divisor (GCD) of two integers using the **Euclidian algorithm** (described below).
    1. Start with two inputs (int **a** and int **b**)
    2. Compute the remainder '**r**' of dividing **a** by **b** (r = a % b)
    3. Replace **a** with **b** and **b** with **r**
    4. Repeat steps 2 and 3 until **b** becomes 0
    5. The GCD is the value of **a** when **b** becomes 0

- power(double base, int exponent): A static method to calculate the result of raising a base to an exponent. Use a loop in this method to calculate the result (again, you cannot use any static methods from the Math class from the java.lang package, so the pow method cannot be used here).

- calculateCircleArea(double radius): A static method to calculate the area of a circle given its radius, using the constant 'PI'. Recall that Area = $\pi * r^2$.

- factorial(int n): A static method to calculate the factorial of a value using an iterative approach. If value is less than 0, throw an IllegalArgumentException with the message "Value must be non-negative". Remember that the factorial equation is:

- $$n! = n \times (n-1) \times (n-2) \times \ldots \times 1$$

By using 'static' methods and fields, we ensure that these utilities can be accessed directly through the class without needing to create an instance. The 'final' keyword ensures that the constants cannot be modified, preserving their values.

Q 01.1

Once you have concluded writing this class, copy and paste the following into a file named Main.java and compile and execute. What is your output?

```java
public class Main {
    public static void main(String[] args) {
        // test the factorial method
        int number = 5;
        long factResult = MathUtilities.factorial(number);
        System.out.println("Factorial of " + number + " is: " + factResult);

        // testing other methods
        System.out.println("GCD: " + MathUtilities.gcd(48, 18));
        System.out.println("Power: " + MathUtilities.power(2, 10));
        System.out.println("Circle Area: " + MathUtilities.calculateCircleArea(3.0));
    }
}
```

Exercise 02

You have been hired by a small but rapidly growing tech company, "SuperTotallyLegitimate Corporation", to develop a user account management system for their new platform. The platform will allow users to register, manage their accounts, and perform various activities. One of your tasks is to ensure that each user has a unique identifier, and that the system can track the total number of registered users at any given time. Additionally, the usernames must be immutable once they are set.

To accomplish this, you need to write a 'UserAccount' class with the following requirements:

1. Each user account should have a unique user ID (an int, **userId**).
2. The user ID should be automatically assigned when a new user account is created.
3. The username should be set during the creation of the account and should not be modifiable afterwards (a String, **username**).
4. The system should be able to provide the total number of user accounts created at any time (an int, **instanceCount**).

I have provided a template below which is, for the most part, complete. You need to copy and paste the code into your own UserAccount.java file and include the class variables, instance variables, and finish writing the constructor.

```java
public class UserAccount {
    // variable to count instances
    // unique ID for each user
    // username for each user

    public UserAccount(String username) {
        // finish this constructor
    }

    public static int getInstanceCount() {
        return instanceCount;
    }

    public int getUserId() {
        return userId;
    }

    public String getUsername() {
        return username;
    }
}
```

Q 02.1

Once you have concluded writing this class, copy and paste the following into a file named Main.java and compile and execute. What is your output?

```java
public class Main {
    public static void main(String[] args) {
        // create several UserAccount instances with different usernames
        UserAccount user1 = new UserAccount("alice");
        UserAccount user2 = new UserAccount("bob");
        UserAccount user3 = new UserAccount("charlie");
        // print the details of each user
        System.out.println("User 1: ID = " + user1.getUserId() + ", Username = " + user1.getUsername());
        System.out.println("User 2: ID = " + user2.getUserId() + ", Username = " + user2.getUsername());
        System.out.println("User 3: ID = " + user3.getUserId() + ", Username = " + user3.getUsername());
        // print the total number of instances created
        System.out.println("Total number of instances created: " + UserAccount.getInstanceCount());
        // create more UserAccount instances
        UserAccount user4 = new UserAccount("david");
        UserAccount user5 = new UserAccount("eve");
        // print the details of the new users
        System.out.println("User 4: ID = " + user4.getUserId() + ", Username = " + user4.getUsername());
        System.out.println("User 5: ID = " + user5.getUserId() + ", Username = " + user5.getUsername());
        // print the updated total number of instances created
        System.out.println("Total number of instances created: " + UserAccount.getInstanceCount());
    }
}
```

Q 02.2

What might we do in the case that we want to add a password associated with each UserAccount? Would it be an instance variable? A class variable? Should it be final? How would a user update it? Discuss!

Exercise 03

Let's write a class, Geometry, that uses method overloading to handle the following scenarios:

- Calculate the area of a rectangle given its width (w) and height (h) (A = w*h)
- Calculate the area of a square given its side length (l) (A = l^2)
- Calculate the area of a circle given its radius (A = $\pi * r^2$)
- Calculate the area of a triangle given its base (b) and height (h) (A = ½ b * h)

The Geometry class should contain four overloaded methods called **area**, each designed to calculate the area of a different geometric shape.

Hint: Consider the area method for rectangle and triangle. They both require two variables (w and h for rectangle, b and h for triangle). But then, the method signatures would have the same name and both take two doubles. So, you need to differentiate between them by either changing the number or type of parameters or by adding an additional parameter to distinguish the methods. For example, you could add a third parameter (a boolean) to one of the methods indicate whether the shape is a triangle or a rectangle. This same problem exists between squares and circles, so you could approach that the same way.

Q 03.1

Why are unique method signatures essential in method overloading? What happens if we try to write non-unique method signatures?

Q 03.2

Once you have concluded writing this class, copy and paste the following into a file named Main.java. Make sure that you replace the '...' with the required arguments based on the values provided in the String before the method call (as well as any other arguments your method requires). Compile and execute. What is your output?

```
public static void main(String[] args) {
    Geometry geo = new Geometry();

    // test the area methods
    System.out.println("Area of rectangle (length 5, height 3): " + geo.area(...));
    System.out.println("Area of square (side length 4): " + geo.area(...));
    System.out.println("Area of circle (radius 3): " + geo.area(...));
    System.out.println("Area of triangle (base 6, height 4): " + geo.area(...));
}
```

Q 03.3

Discuss why method overloading is useful in this scenario and how it improves code readability and organization.

Exercise 04

Consider the following code:

ParentClass.java:

```java
public class ParentClass {
    public final String printMessage() {
        return "Parent class message!";
    }
}
```

ChildClass.java:

```java
public class ChildClass extends ParentClass {
    public String printMessage() {
        return "Child class message!";
    }
}
```

Main.java

```java
public class Main {
    public static void main(String[] args) {
        ChildClass c1 = new ChildClass();
        System.out.println(c1.printMessage());
    }
}
```

Q 04.1

What is wrong with the above code? How could we fix it?