OOP LAB FINAL (WRITING PART)

1. List all the topics discussed in the lab work.

Topics Discussed in Lab Work:

1. Class and Object	2. Inheritance	3. Interface
4. Abstract Class	5. Packages	6. Access Modifiers
7. Encapsulation (getter-setter method)	8. Static Methods	9. Exception Handling
10. Wrapper Classes	11. Polymorphism	

What have you learned after completing this lab work? 10

What I have learned after completing this lab work:

Through the lab work, I gained practical experience with OOP concepts, including creating and instantiating classes and objects, implementing inheritance for code reuse, and using interfaces for multiple inheritance. I learned to design abstract classes, organize code with packages, and control data visibility with access modifiers. Encapsulation was practiced using private variables with getter and setter methods, while wrapper classes handled primitive values as objects. I utilized static methods for utility functions and effectively managed errors using exception handling. Finally, I demonstrated polymorphism through method overriding and overloading, enhancing my ability to design and implement robust programs.

2. What are the different types of inheritance? Differentiate between abstract class and encapsulation.

Types of Inheritance:

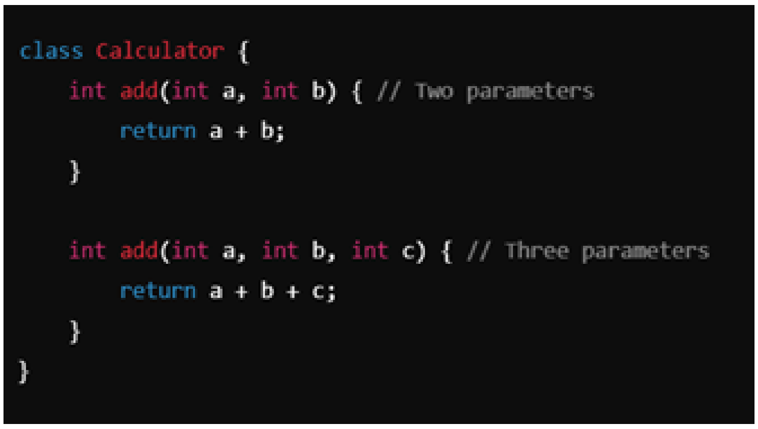
1. **Single Inheritance:**
   a. Example: class A{} → class B extends A{}
2. **Multilevel Inheritance:**
   a. Example: class A{} → class B extends A{} → class C extends B{}
3. **Hierarchical Inheritance:**
   a. Example: class A{} → class B extends A{}, class C extends A{}.
4. **Multiple Inheritance (via Interface):**
   a. A class implements multiple interfaces to achieve multiple inheritance in Java.
5. **Hybrid Inheritance (via Interface):**
   a. A combination of two or more types of inheritance achieved using interfaces.

Difference Between Abstract Class and Encapsulation:

Abstract Class	Encapsulation
A class that provides a template for subclasses.	Hiding data and providing controlled access through methods.
To ensure certain methods are implemented by subclasses.	To protect data and prevent unauthorized access.
Can have both abstract (no body) and concrete methods.	Uses private fields and public getter/setter methods.
Achieved using the abstract keyword.	Achieved by using access modifiers (e.g., private, public).

3. Give example of method overloading and method overriding.

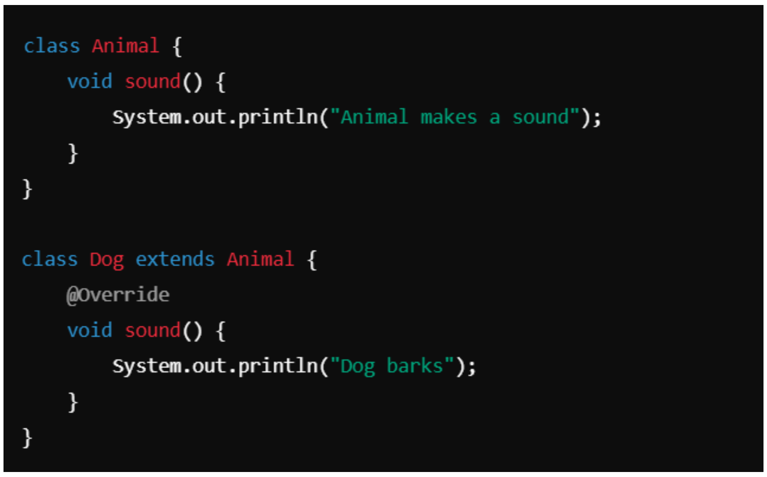
<u>Method Overloading Example:</u>



```java
class Calculator {
    int add(int a, int b) { // Two parameters
        return a + b;
    }

    int add(int a, int b, int c) { // Three parameters
        return a + b + c;
    }
}
```

<u>Method Overriding Example:</u>



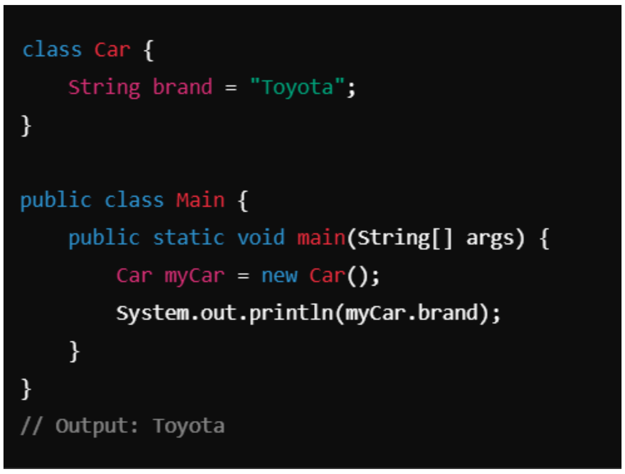
```java
class Animal {
    void sound() {
        System.out.println("Animal makes a sound");
    }
}

class Dog extends Animal {
    @Override
    void sound() {
        System.out.println("Dog barks");
    }
}
```

4. What are the main features of OOPs? Give example for each.

<u>Main Features of OOPs:</u>

1. **<u>Classes and Objects:</u> Class as a blueprint; object as its instance.**

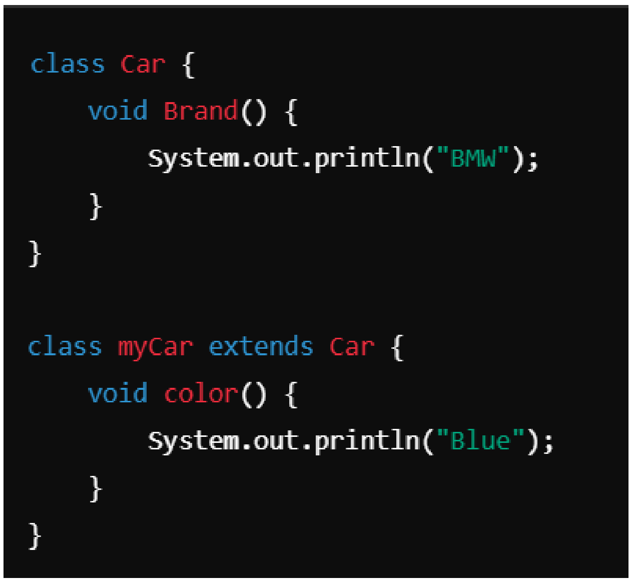


```java
class Car {
    String brand = "Toyota";
}

public class Main {
    public static void main(String[] args) {
        Car myCar = new Car();
        System.out.println(myCar.brand);
    }
}
// Output: Toyota
```

2. **<u>Inheritance</u>**

Reusing code from a parent class.

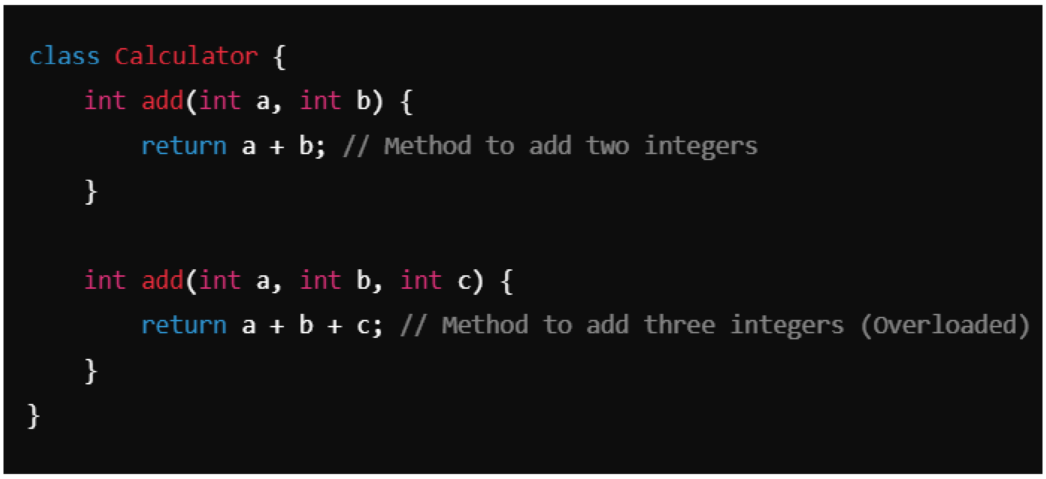


```
class Car {
    void Brand() {
        System.out.println("BMW");
    }
}

class myCar extends Car {
    void color() {
        System.out.println("Blue");
    }
}
```

3. Polymorphism

Multiple forms of a method (overloading and overriding).

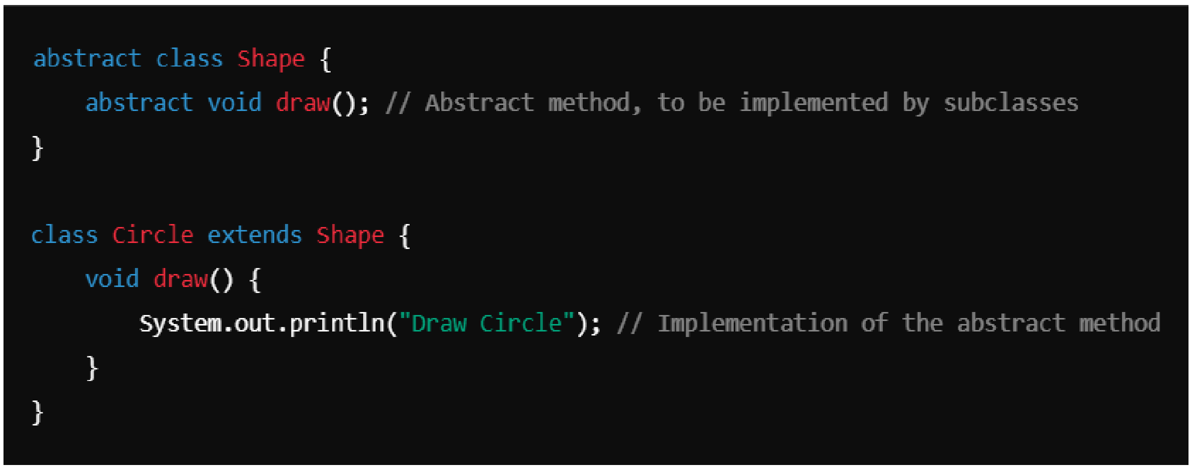


```
class Calculator {
    int add(int a, int b) {
        return a + b; // Method to add two integers
    }

    int add(int a, int b, int c) {
        return a + b + c; // Method to add three integers (Overloaded)
    }
}
```

4. Abstraction

Hiding implementation details using abstract classes or interfaces.



```
abstract class Shape {
    abstract void draw(); // Abstract method, to be implemented by subclasses
}

class Circle extends Shape {
    void draw() {
        System.out.println("Draw Circle"); // Implementation of the abstract method
    }
}
```

5. **Encapsulation**

   Hiding data and providing access via methods.

```
class Student {
    private String name;
    public String getName() { return name; }
    public void setName(String name) { this.name = name; }
}
```

5. Can you call the base class method without creating an instance?

Yes, I can call a base class method without creating an instance if the method is static. Static methods belong to the class itself, not to an instance of the class, so I can call them directly using the class name.

6. What is the difference between public, private and protected access modifiers?

Access Modifier	Access Level	Usage
Public	Accessible everywhere (within class, package, and outside)	Used for unrestricted access
Private	Accessible only within the same class	Used to restrict access to the class itself
Protected	Accessible within the same package and subclasses	Used for access within the package and by subclasses

7. Write a complete scenario with an example that reflects your overall understanding of this course. (21-22, 20-21, 19-20, 18-19)

Scenario: School Management System

In the school management system, the goal is to manage different individuals, such as students and teachers. Each individuals has common attributes like name and age, but their roles are different. Now creating a scenario by demonstrating all the main features of oops.

1. Abstraction:

Abstract class - Person, Represents the common behavion (like name of the school. and age of all people

2. Encapsulation:

Encapsulate the name and age attributes by using private and only accessed through getter setter methods.

3. In heritance:

Student and Teacher inherit the parent class ‘Person' to implement the properties of "Person” class.

4. Polymorphism:

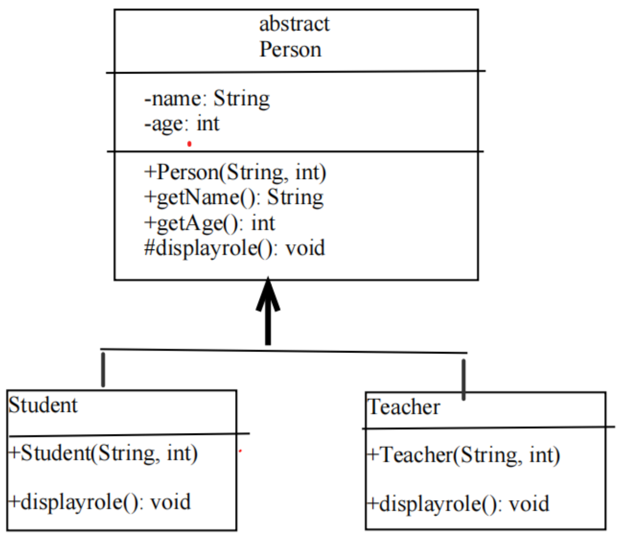
The "displayrole" method is overriding in both ‘Student’ and ‘Teacher’ class.

5. Class and Object:

In the Main" method, create the instance of "Person" class.

Class Diagram:

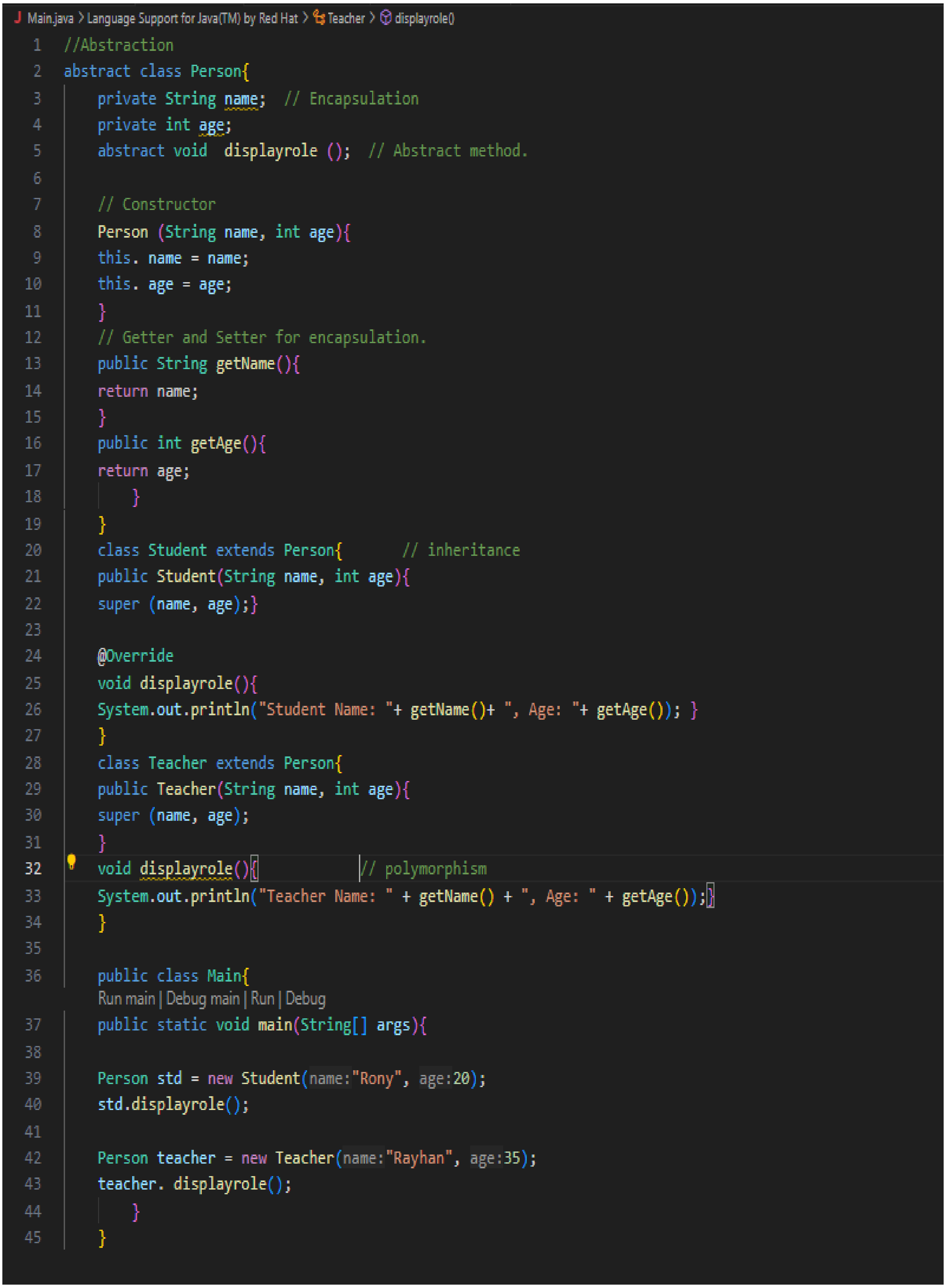
Here is a simple UML diagram of my scenario:



Now here is the implementation of this class diagram:

↓

P.T.O


Main.java > Language Support for Java(TM) by Red Hat > Teacher > displayrole()

```java
//Abstraction
abstract class Person{
    private String name;  // Encapsulation
    private int age;
    abstract void  displayrole ();  // Abstract method.

    // Constructor
    Person (String name, int age){
    this. name = name;
    this. age = age;
    }
    // Getter and Setter for encapsulation.
    public String getName(){
    return name;
    }
    public int getAge(){
    return age;
        }
    }
    class Student extends Person{       // inheritance
    public Student(String name, int age){
    super (name, age);}

    @Override
    void displayrole(){
    System.out.println("Student Name: "+ getName()+ ", Age: "+ getAge()); }
    }
    class Teacher extends Person{
    public Teacher(String name, int age){
    super (name, age);
    }
    void displayrole(){             // polymorphism
    System.out.println("Teacher Name: " + getName() + ", Age: " + getAge());}
    }

    public class Main{
    Run main | Debug main | Run | Debug
    public static void main(String[] args){

    Person std = new Student(name:"Rony", age:20);
    std.displayrole();

    Person teacher = new Teacher(name:"Rayhan", age:35);
    teacher. displayrole();
        }
    }
```