Data Structures
(UNC402)

Submitted by:

Name:- Aarav Gupta
Roll No:-
1024060206
Subgroup:- 2F31 (ECE)

THAPAR INSTITUTE
OF ENGINEERING & TECHNOLOGY
(Deemed to be University)

Submitted to:-

Dr. Poonam Verma Bhardwaj Ma'am

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING

THAPAR INSTITUTE OF ENGINEERING AND TECHNOLOGY,
(DEEMED TO BE UNIVERSITY),

PATIALA, PUNJAB
INDIA

Lab Assignment 3
UNC402_Data
Structures
Programs based on inheritance & polymorphism

Part -1 Understanding Access Specifiers in Inheritance:

In C++, access specifiers control the accessibility of class members in inheritance.

- Private members are accessible only within the class where they are declared and cannot be inherited by derived classes.
- Protected members are accessible within the class itself and can be inherited by derived classes, making them available within the derived class but not accessible from outside.
- Public members are accessible both within and outside the class, including in derived classes and external code.

1. Single Inheritance

Write a C++ program to demonstrate single inheritance. In this program, create a base class `Person` with protected data members `name` and `age`. Then, derive a class `Student` from `Person`, which adds a private data member `rollNo`. The `Student` class should access the base class members and provide functionality to display the personal information along with the roll number

2. Multiple Inheritance

Write a C++ program to demonstrate multiple inheritance. In this example, create two base classes: `Engineer` with a protected data member `engField`, and `Manager` with a protected data member `teamSize`. Then, derive a class `TechManager` from both `Engineer` and `Manager`, which will combine the attributes of both base classes. The `TechManager` class should display details related to both engineering field and team management.

3. Hybrid Inheritance

Write a C++ program to demonstrate hybrid inheritance without using virtual classes or functions. In this program, create a base class `Person` with a protected data member `name`. Derive two classes: `Student` with a protected data member `rollNo` and `Employee` with a protected data member `employeeID`. Finally, create a class `WorkingStudent` that inherits from both `Student` and `Employee`, combining the attributes of both classes. The `WorkingStudent` class should display details like name, roll.

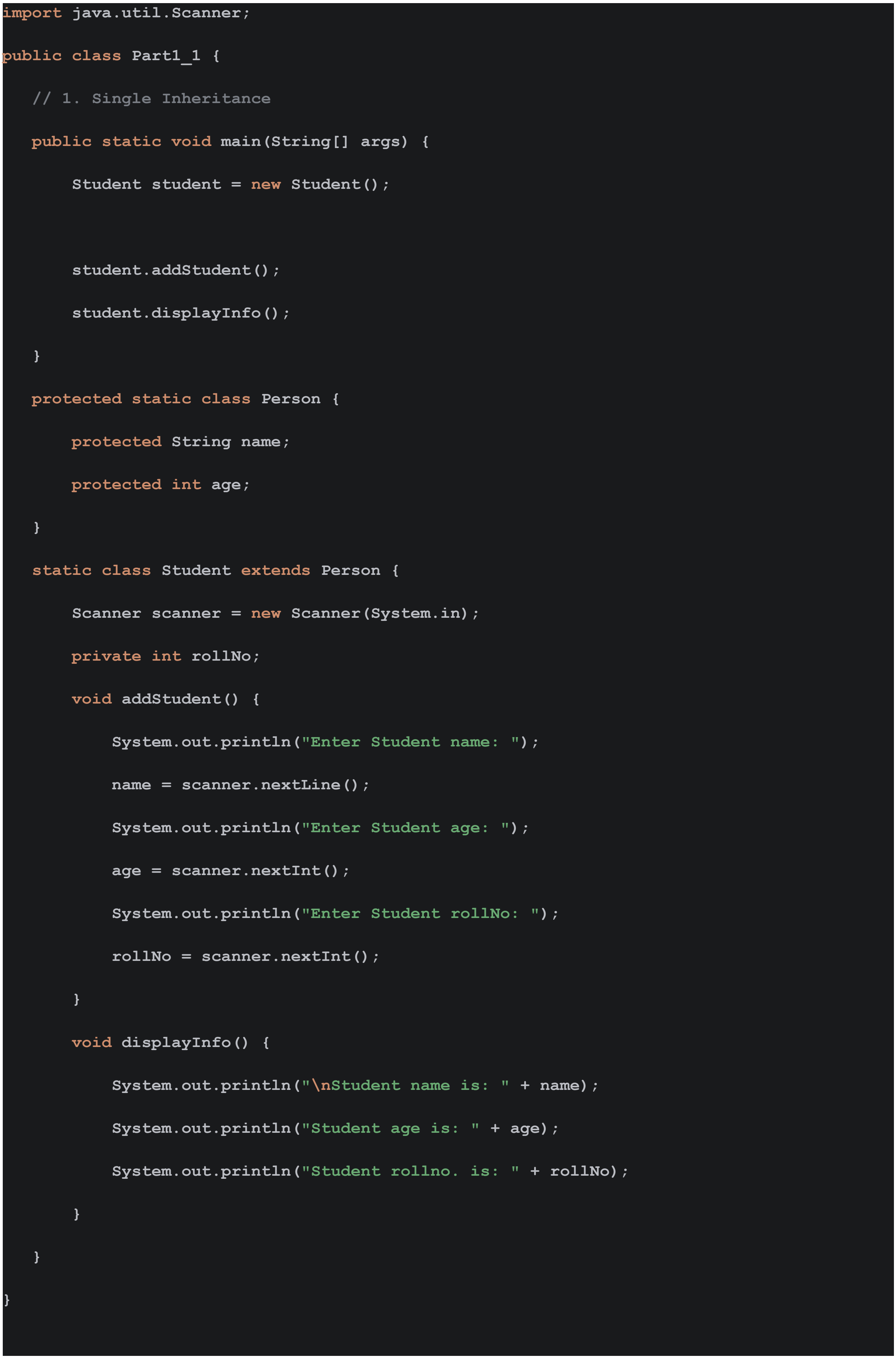
Part -2 Understanding of Virtual class

Implement class hierarchies to explore inheritance and virtual inheritance in C++.

- **Classes A (Without Virtual Inheritance), B, C (Without Virtual Inheritance), and D (Without Virtual Inheritance)**
- Create a base class A with an integer member data and a method display().
- Implement classes B and C inheriting from A.
- Implement class D inheriting from both B and C. Handle any ambiguity in accessing data.

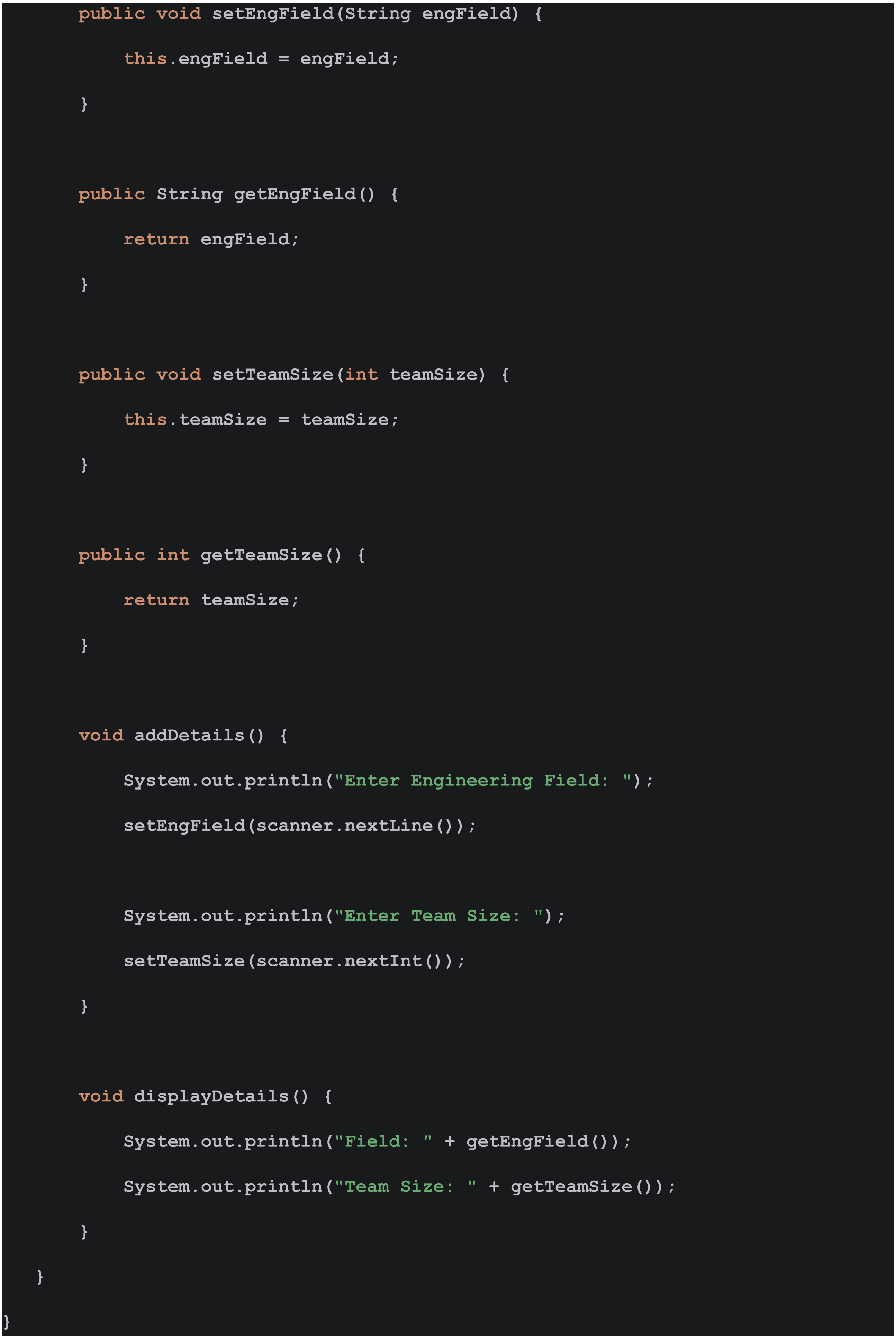
- **Classes A (Without Virtual Inheritance), B, C (With Virtual Inheritance), and D (Without Virtual Inheritance)**
- Same as above, but make class C inherit from A with virtual inheritance.
- Implement class D and manage access to data.

- **Classes A (Without Virtual Inheritance), B (With Virtual Inheritance), C (With Virtual Inheritance), and D (Without Virtual Inheritance)**
- Make both classes B and C use virtual inheritance.
- Implement class D and ensure proper access to data.



```java
import java.util.Scanner;
public class Part1_1 {
    // 1. Single Inheritance
    public static void main(String[] args) {
        Student student = new Student();

        student.addStudent();
        student.displayInfo();
    }
    protected static class Person {
        protected String name;
        protected int age;
    }
    static class Student extends Person {
        Scanner scanner = new Scanner(System.in);
        private int rollNo;
        void addStudent() {
            System.out.println("Enter Student name: ");
            name = scanner.nextLine();
            System.out.println("Enter Student age: ");
            age = scanner.nextInt();
            System.out.println("Enter Student rollNo: ");
            rollNo = scanner.nextInt();
        }
        void displayInfo() {
            System.out.println("\nStudent name is: " + name);
            System.out.println("Student age is: " + age);
            System.out.println("Student rollno. is: " + rollNo);
        }
    }
}
```

```java
import java.util.Scanner;

public class Part1_2 {
    // Multiple Inheritance
    public static void main(String[] args) {
        TechManager techManager = new TechManager();
        techManager.addDetails();
        techManager.displayDetails();
    }

    /// java doesn't support multiple inheritance via subclassing so i
implemented interfaces
    /// which means `class TechManager extends Engineer, Manager` -> will not
work.
    /// (protocols defining the functions that Engineer/Manager Class would have
had and behaves similar to how multiple inheritance works in cpp.)
    interface Engineer {
        void setEngField(String Field);
        String getEngField();
    }

    interface Manager {
        void setTeamSize(int size);
        int getTeamSize();
    }

    static class TechManager implements Engineer, Manager {
        Scanner scanner = new Scanner(System.in);

        String engField;
        int teamSize;
```



```
        public void setEngField(String engField) {
            this.engField = engField;
        }

        public String getEngField() {
            return engField;
        }

        public void setTeamSize(int teamSize) {
            this.teamSize = teamSize;
        }

        public int getTeamSize() {
            return teamSize;
        }

        void addDetails() {
            System.out.println("Enter Engineering Field: ");
            setEngField(scanner.nextLine());

            System.out.println("Enter Team Size: ");
            setTeamSize(scanner.nextInt());
        }

        void displayDetails() {
            System.out.println("Field: " + getEngField());
            System.out.println("Team Size: " + getTeamSize());
        }
    }
}
```

```java
import java.util.Scanner;

public class Part1_3 {
    // 3. Hybrid Inheritance

    public static void main(String[] args) {
        WorkingStudent workingStudent = new WorkingStudent();
        workingStudent.addDetails();
        workingStudent.displayDetails();
    }

    static class Person {
        protected String name;
    }

    interface Student {
        void setRollNo(int rollNo);
        int getRollNo();
    }

    interface Employee {
        void setEmpID(int id);
        int getEmpID();
    }

    // The Hybrid Class
    // Extends Person (for Name) and Implements Interfaces (for RollNo & ID)
    static class WorkingStudent extends Person implements Student, Employee {
        Scanner scanner = new Scanner(System.in);

        int rollNo;

        int empID;


        public void setRollNo(int rollNo) { this.rollNo = rollNo; }

        public int getRollNo() { return rollNo; }


        public void setEmpID(int id) { this.empID = id; }

        public int getEmpID() { return empID; }


        void addDetails() {

            System.out.println("Enter Student Details");

            System.out.print("Enter Name: ");

            this.name = scanner.nextLine();


            System.out.print("Enter Roll No: ");

            setRollNo(scanner.nextInt());


            System.out.print("Enter Employee ID: ");

            setEmpID(scanner.nextInt());

        }


        void displayDetails() {

            System.out.println("\nStudent Details:");

            System.out.println("Name: " + name);

            System.out.println("Roll No: " + getRollNo());

            System.out.println("Emp ID: " + getEmpID());

        }

    }

}
```

```cpp
#include <iostream>
// 'virtual inheritance' does not exist as a concept in Java.
class A {
public:
    int data;
    void display() {
        std::cout << "Data in A: " << data << std::endl;
    }
};
// 'virtual' keyword prevents creating two copies of A in class D
class B: virtual public A {};
class C: virtual public A {};
// Inherits from both B and C.
// D has only one copy of 'data'.
class D: public B, public C {
public:
    void setData(int value) {
        data = value;
    }
};

int main() {
    D obj;
    std::cout << "Virtual Inheritance Demo" << std::endl;
    obj.setData(100);
    obj.display();
    return 0;
}
```