SCHOOL MANAGEMENT SYSTEM

PROJECT REPORT

T.ABBISHANTH

UWU/CST/22/059

Introduction

The School Management System is a Java application designed to demonstrate key Object-Oriented Programming (OOP) concepts. The system simulates basic operations of a school, including management of teachers, students, and courses.

Features

1 . Add Student
2 . Add Teacher
3 . Delete Record
4 . Display Records
5 . Exit

Topics Covered

1 . Inheritance

The `**Student**` and `**Teacher**` classes inherit from the `**Person**` class, allowing code reuse and the establishment of a hierarchical relationship.

2 . Polymorphism

The `**displayInfo**` method in the `**Person**` class is overridden by the `**Student**` and `**Teacher**` classes. This allows for dynamic method binding and ensures that the correct method is called based on the object type at runtime.

3 . Encapsulation

The fields in each class are marked as private and accessed through public getter methods. This ensures that the internal state of an object is protected and can only be modified through controlled methods.

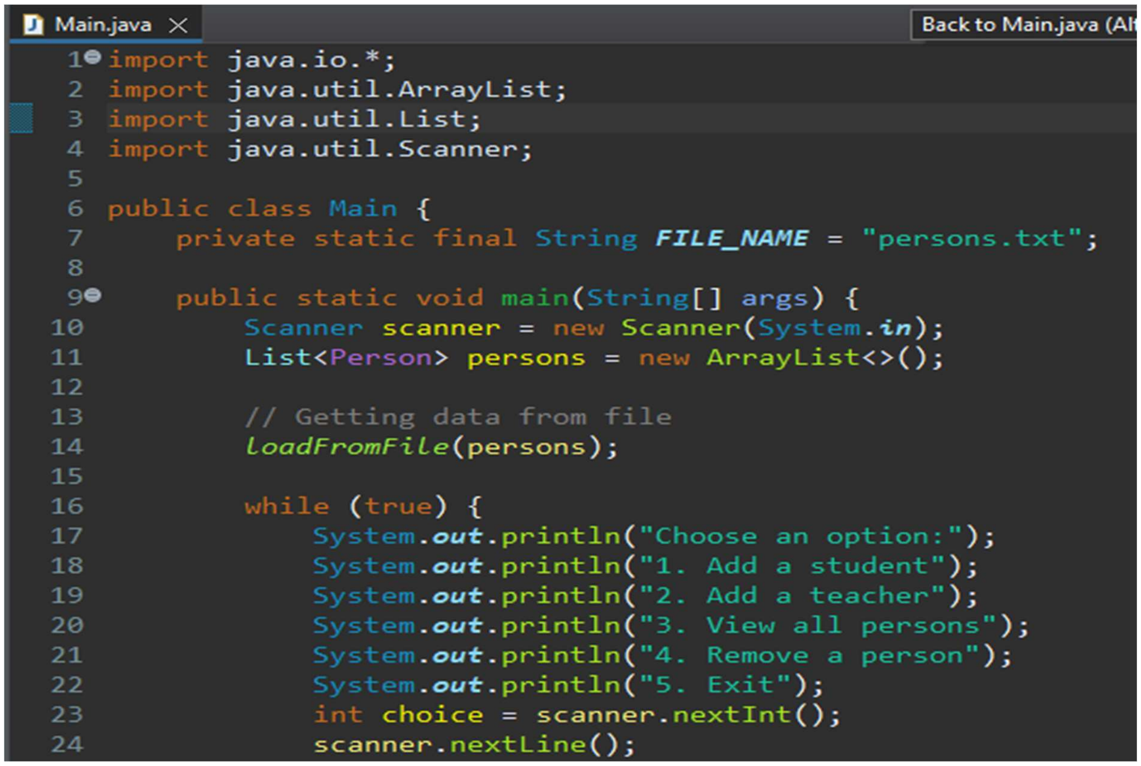
4 . Abstraction

The `**Person**` class is defined as an abstract class and cannot be instantiated. This class provides a template for `**Student**` and `**Teacher**` classes and enforces the implementation of the `**displayInfo**` method.

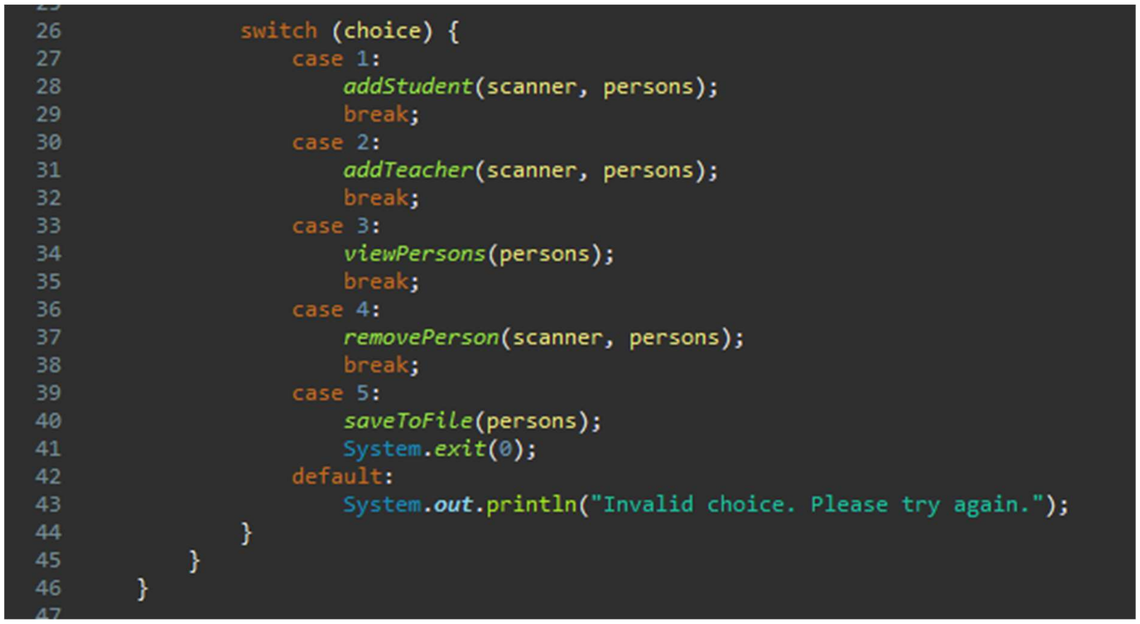
- **Code Explanation**

1.Main.java

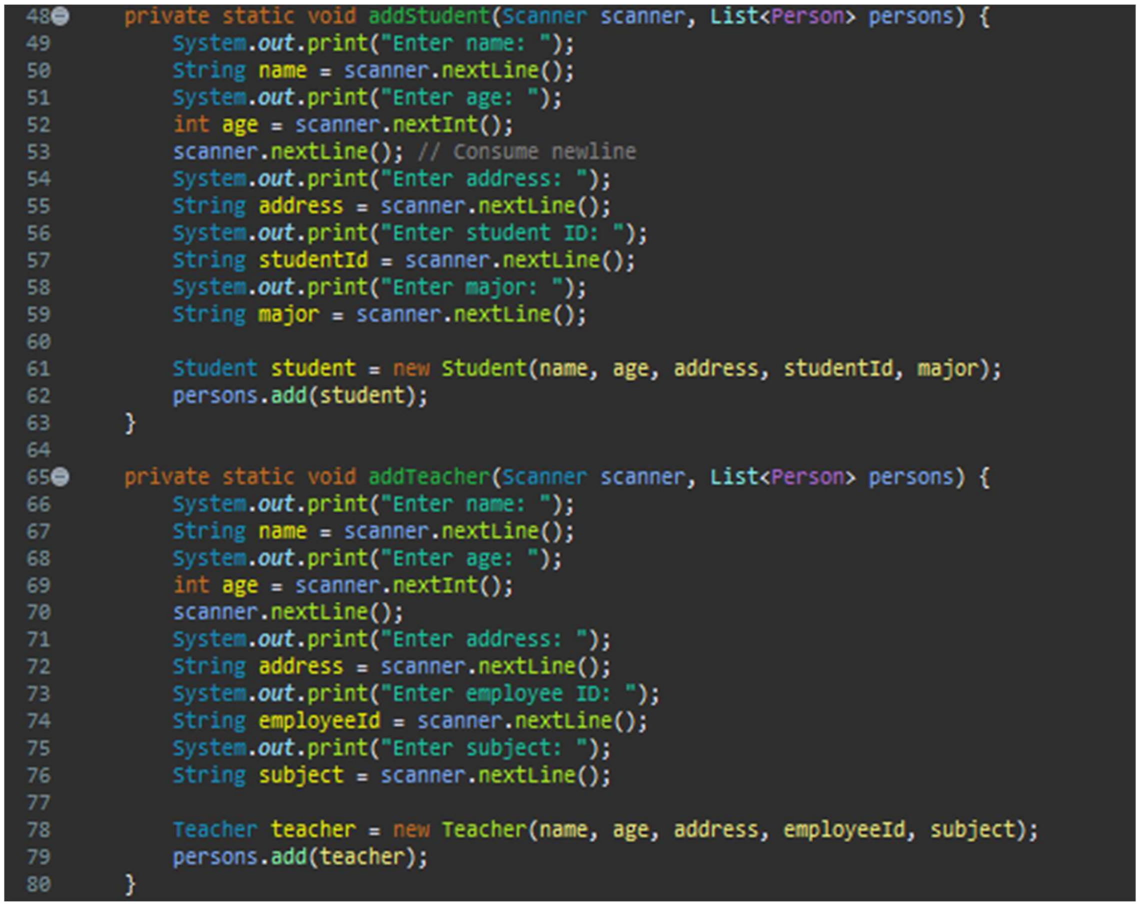
This is the entry point of the application. It presents a menu to the user to perform various operations such as adding a student or teacher, deleting a record, and displaying records.



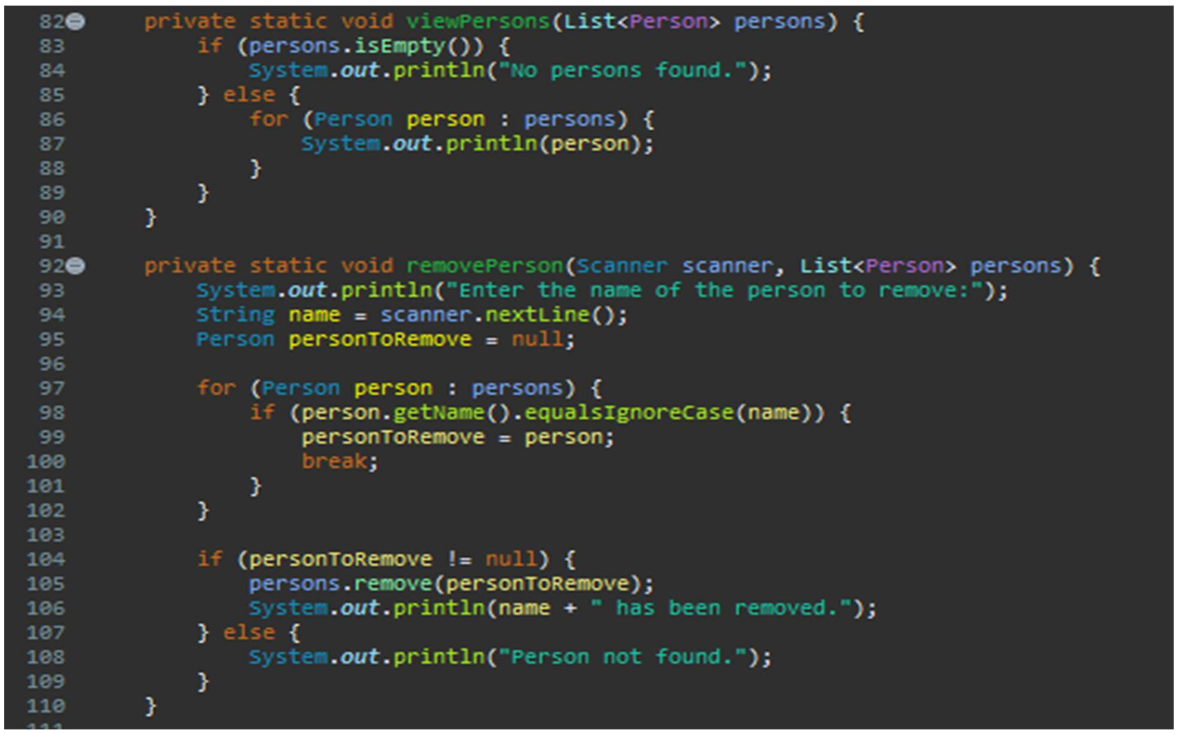
```java
import java.io.*;
import java.util.ArrayList;
import java.util.List;
import java.util.Scanner;

public class Main {
    private static final String FILE_NAME = "persons.txt";

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        List<Person> persons = new ArrayList<>();

        // Getting data from file
        loadFromFile(persons);

        while (true) {
            System.out.println("Choose an option:");
            System.out.println("1. Add a student");
            System.out.println("2. Add a teacher");
            System.out.println("3. View all persons");
            System.out.println("4. Remove a person");
            System.out.println("5. Exit");
            int choice = scanner.nextInt();
            scanner.nextLine();
```



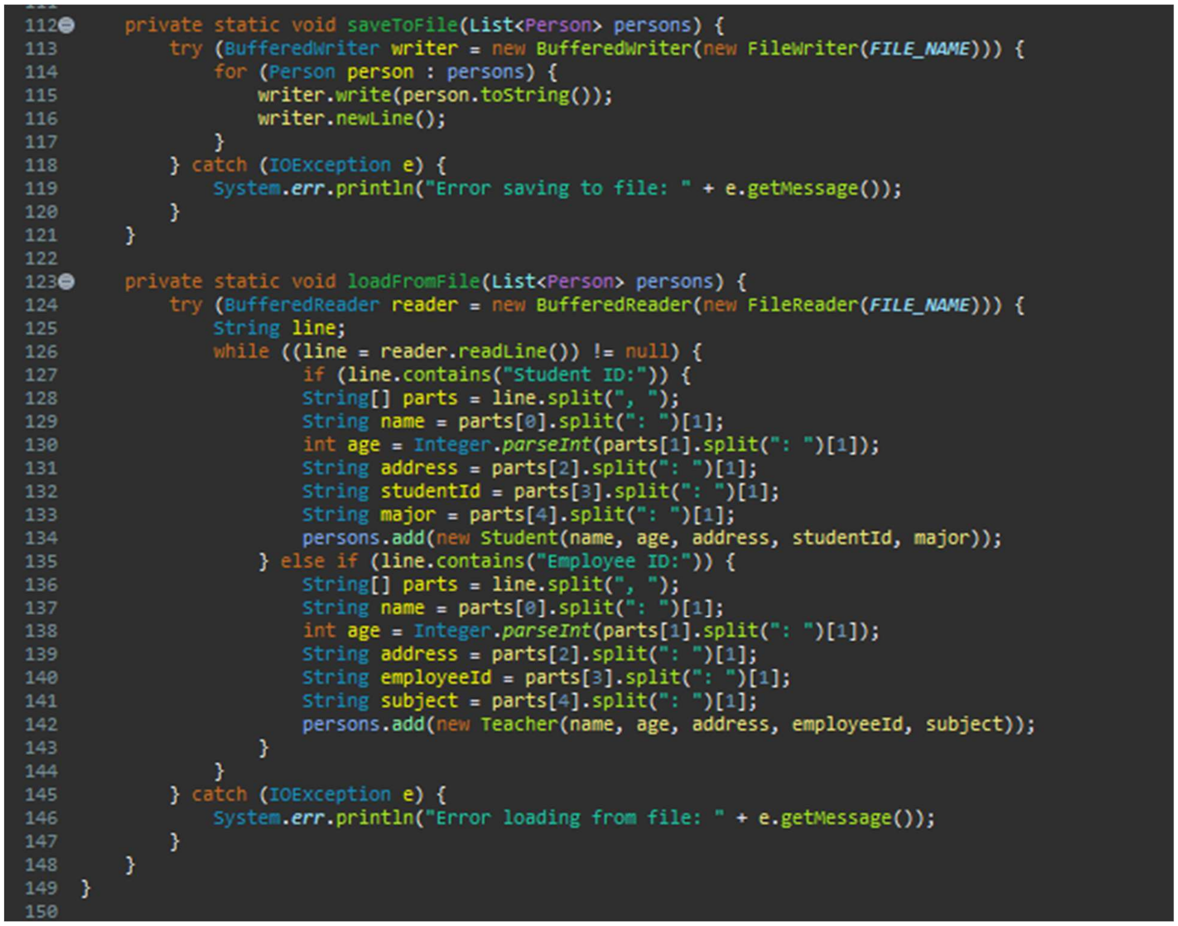
```java
            switch (choice) {
                case 1:
                    addStudent(scanner, persons);
                    break;
                case 2:
                    addTeacher(scanner, persons);
                    break;
                case 3:
                    viewPersons(persons);
                    break;
                case 4:
                    removePerson(scanner, persons);
                    break;
                case 5:
                    saveToFile(persons);
                    System.exit(0);
                default:
                    System.out.println("Invalid choice. Please try again.");
            }
        }
    }
```



```java
    private static void addStudent(Scanner scanner, List<Person> persons) {
        System.out.print("Enter name: ");
        String name = scanner.nextLine();
        System.out.print("Enter age: ");
        int age = scanner.nextInt();
        scanner.nextLine(); // Consume newline
        System.out.print("Enter address: ");
        String address = scanner.nextLine();
        System.out.print("Enter student ID: ");
        String studentId = scanner.nextLine();
        System.out.print("Enter major: ");
        String major = scanner.nextLine();

        Student student = new Student(name, age, address, studentId, major);
        persons.add(student);
    }

    private static void addTeacher(Scanner scanner, List<Person> persons) {
        System.out.print("Enter name: ");
        String name = scanner.nextLine();
        System.out.print("Enter age: ");
        int age = scanner.nextInt();
        scanner.nextLine();
        System.out.print("Enter address: ");
        String address = scanner.nextLine();
        System.out.print("Enter employee ID: ");
        String employeeId = scanner.nextLine();
        System.out.print("Enter subject: ");
        String subject = scanner.nextLine();

        Teacher teacher = new Teacher(name, age, address, employeeId, subject);
        persons.add(teacher);
    }
```



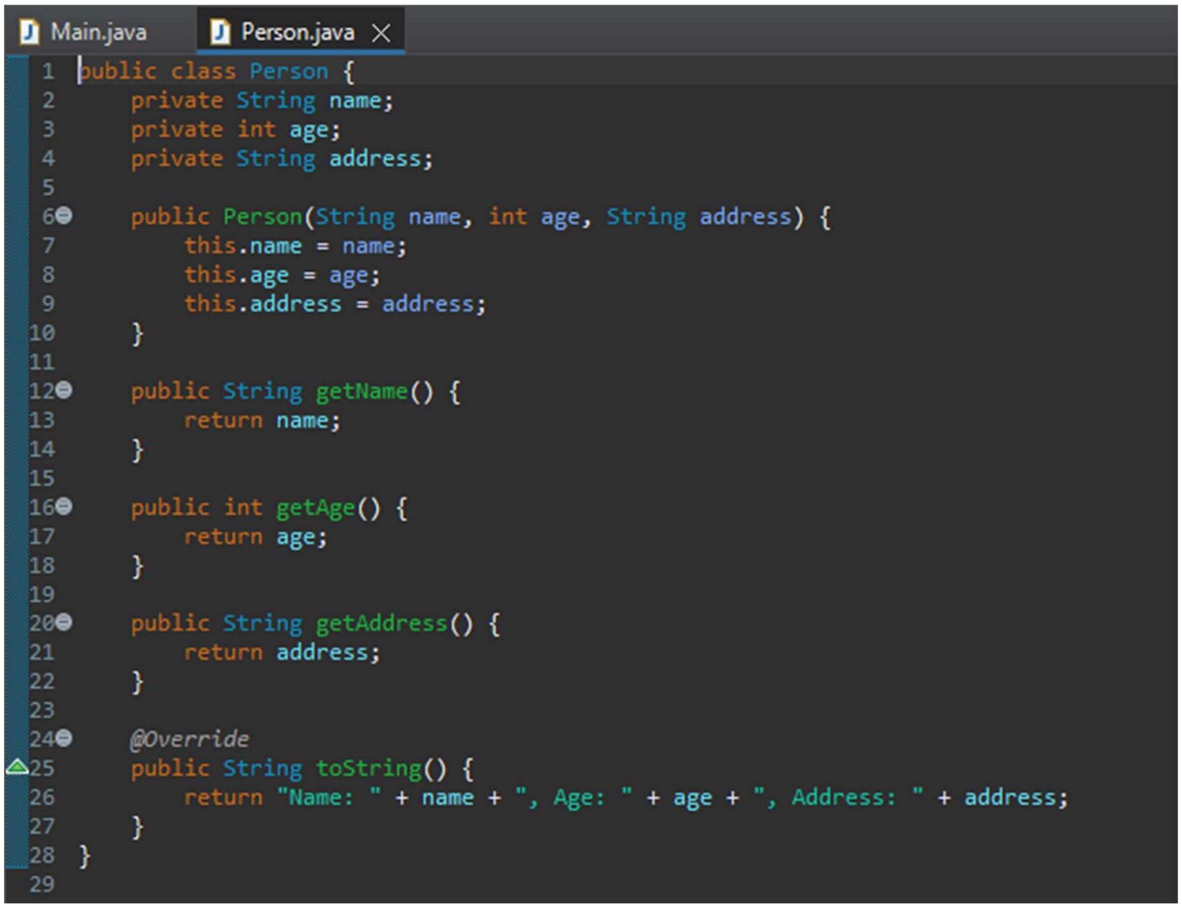
```java
    private static void viewPersons(List<Person> persons) {
        if (persons.isEmpty()) {
            System.out.println("No persons found.");
        } else {
            for (Person person : persons) {
                System.out.println(person);
            }
        }
    }

    private static void removePerson(Scanner scanner, List<Person> persons) {
        System.out.println("Enter the name of the person to remove:");
        String name = scanner.nextLine();
        Person personToRemove = null;

        for (Person person : persons) {
            if (person.getName().equalsIgnoreCase(name)) {
                personToRemove = person;
                break;
            }
        }

        if (personToRemove != null) {
            persons.remove(personToRemove);
            System.out.println(name + " has been removed.");
        } else {
            System.out.println("Person not found.");
        }
    }
```



```java
    private static void saveToFile(List<Person> persons) {
        try (BufferedWriter writer = new BufferedWriter(new FileWriter(FILE_NAME))) {
            for (Person person : persons) {
                writer.write(person.toString());
                writer.newLine();
            }
        } catch (IOException e) {
            System.err.println("Error saving to file: " + e.getMessage());
        }
    }

    private static void loadFromFile(List<Person> persons) {
        try (BufferedReader reader = new BufferedReader(new FileReader(FILE_NAME))) {
            String line;
            while ((line = reader.readLine()) != null) {
                if (line.contains("Student ID:")) {
                    String[] parts = line.split(", ");
                    String name = parts[0].split(": ")[1];
                    int age = Integer.parseInt(parts[1].split(": ")[1]);
                    String address = parts[2].split(": ")[1];
                    String studentId = parts[3].split(": ")[1];
                    String major = parts[4].split(": ")[1];
                    persons.add(new Student(name, age, address, studentId, major));
                } else if (line.contains("Employee ID:")) {
                    String[] parts = line.split(", ");
                    String name = parts[0].split(": ")[1];
                    int age = Integer.parseInt(parts[1].split(": ")[1]);
                    String address = parts[2].split(": ")[1];
                    String employeeId = parts[3].split(": ")[1];
                    String subject = parts[4].split(": ")[1];
                    persons.add(new Teacher(name, age, address, employeeId, subject));
                }
            }
        } catch (IOException e) {
            System.err.println("Error loading from file: " + e.getMessage());
        }
    }
}
```

2 . Person.java

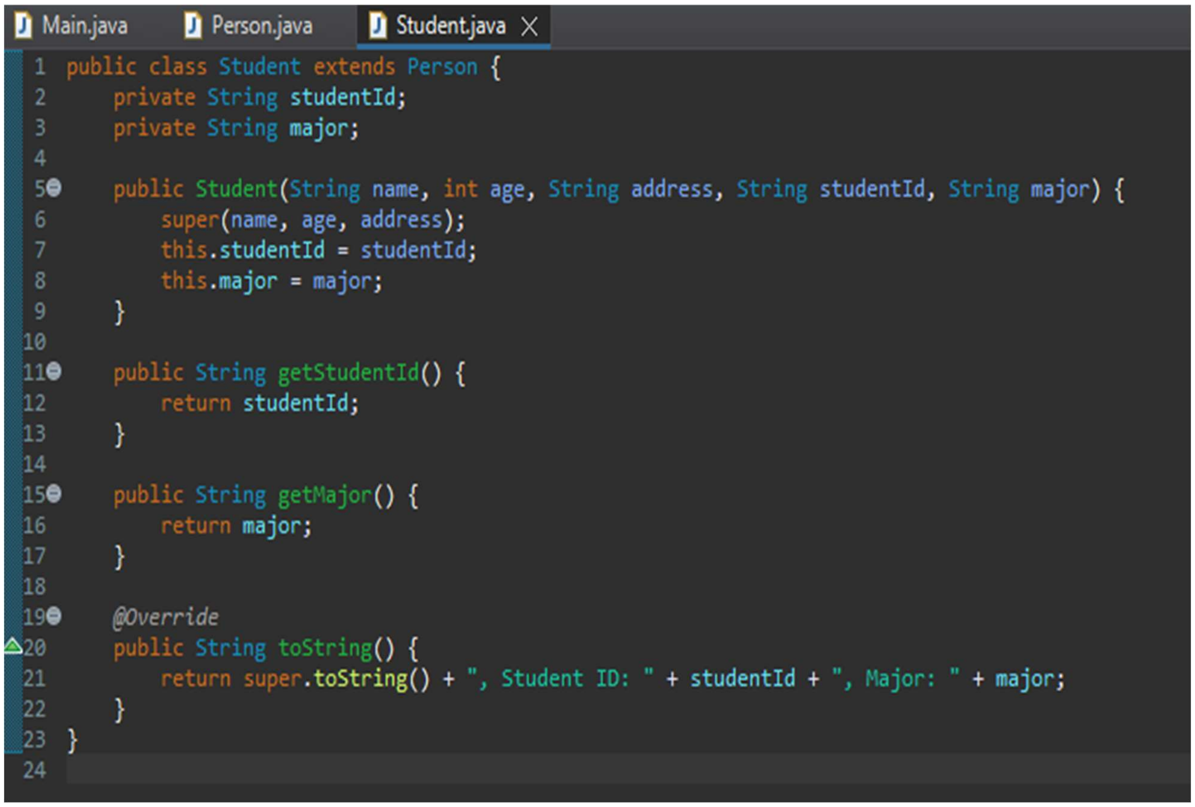
The Person class is a simple Java class designed to encapsulate the basic attributes and behaviors of a person. This class includes three private instance variables: name, age, and address, each of which represents a fundamental characteristic of a person.



```java
public class Person {
    private String name;
    private int age;
    private String address;

    public Person(String name, int age, String address) {
        this.name = name;
        this.age = age;
        this.address = address;
    }

    public String getName() {
        return name;
    }

    public int getAge() {
        return age;
    }

    public String getAddress() {
        return address;
    }

    @Override
    public String toString() {
        return "Name: " + name + ", Age: " + age + ", Address: " + address;
    }
}
```

3 . Student.java

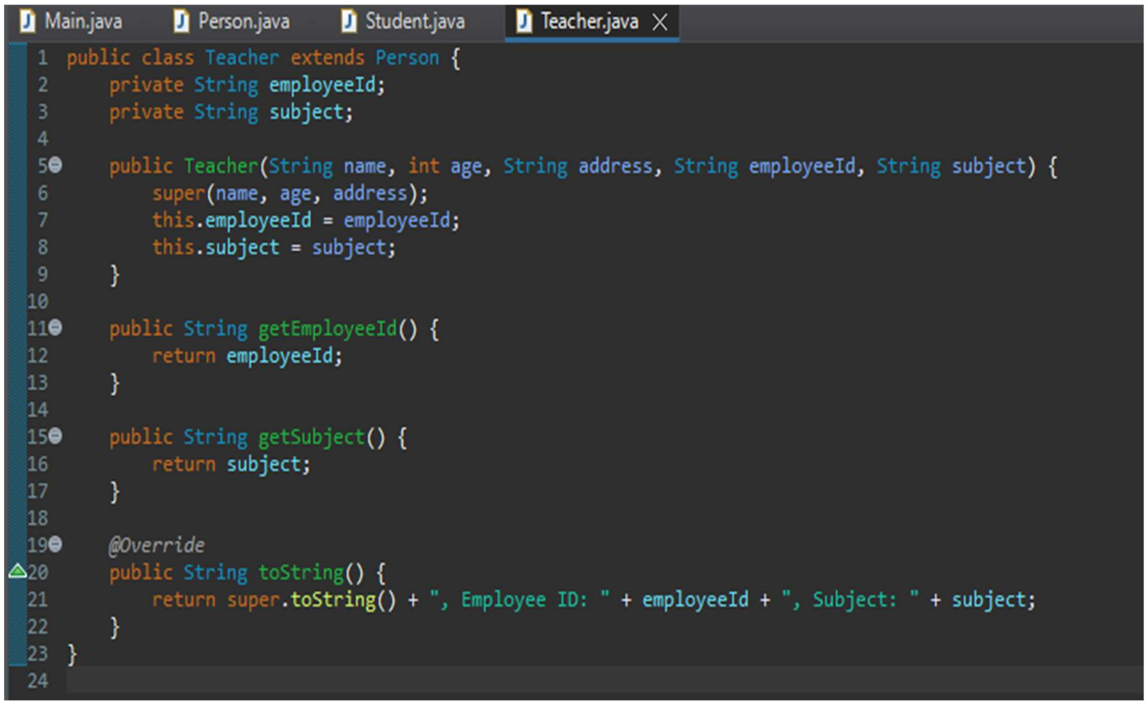
The Student class extends the Person class to model a student, adding specific attributes and behaviors relevant to a student. It inherits the basic characteristics of a person (name, age, address) and introduces additional properties unique to students



```java
public class Student extends Person {
    private String studentId;
    private String major;

    public Student(String name, int age, String address, String studentId, String major) {
        super(name, age, address);
        this.studentId = studentId;
        this.major = major;
    }

    public String getStudentId() {
        return studentId;
    }

    public String getMajor() {
        return major;
    }

    @Override
    public String toString() {
        return super.toString() + ", Student ID: " + studentId + ", Major: " + major;
    }
}
```

4 . Teacher.java

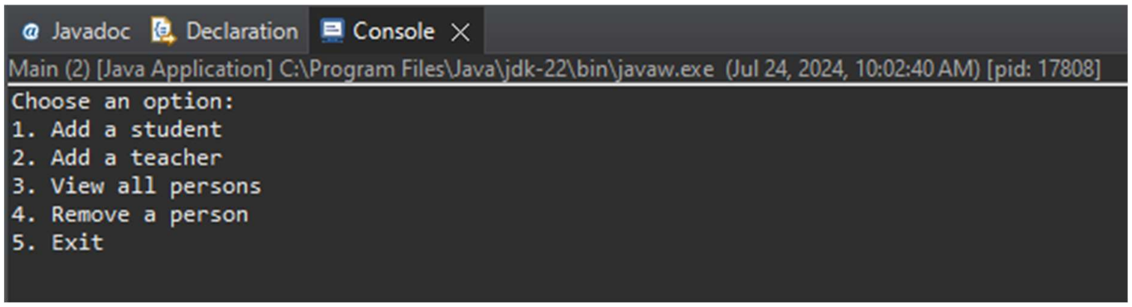
The Teacher class extends the Person class to model a teacher, adding attributes and behaviors specific to a teacher. It inherits the basic characteristics of a person (name, age, address) and introduces additional properties unique to teachers.



```java
public class Teacher extends Person {
    private String employeeId;
    private String subject;

    public Teacher(String name, int age, String address, String employeeId, String subject) {
        super(name, age, address);
        this.employeeId = employeeId;
        this.subject = subject;
    }

    public String getEmployeeId() {
        return employeeId;
    }

    public String getSubject() {
        return subject;
    }

    @Override
    public String toString() {
        return super.toString() + ", Employee ID: " + employeeId + ", Subject: " + subject;
    }
}
```

- **Code Demonstration**

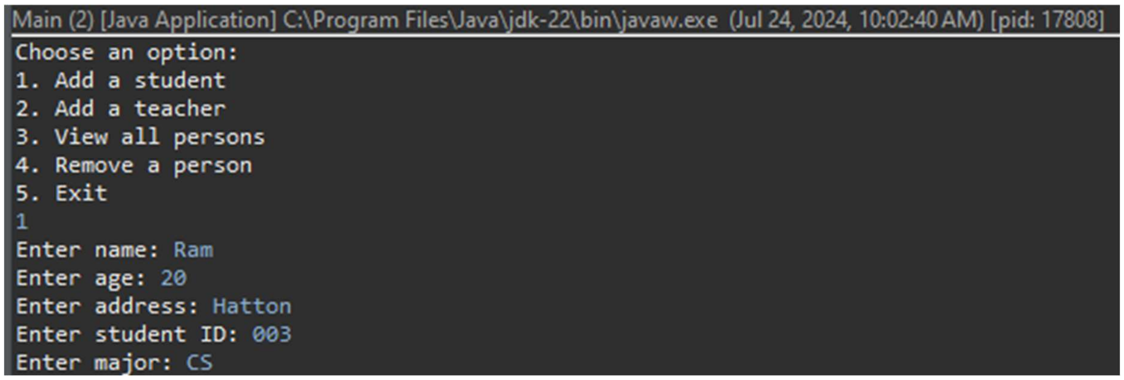
01 – Output



Javadoc Declaration Console

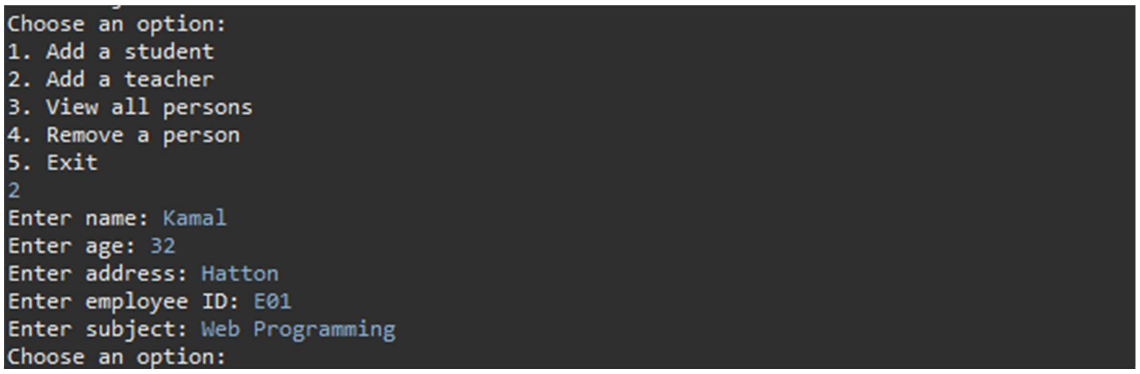
```
Main (2) [Java Application] C:\Program Files\Java\jdk-22\bin\javaw.exe (Jul 24, 2024, 10:02:40 AM) [pid: 17808]
Choose an option:
1. Add a student
2. Add a teacher
3. View all persons
4. Remove a person
5. Exit
```

02 – Add a Student



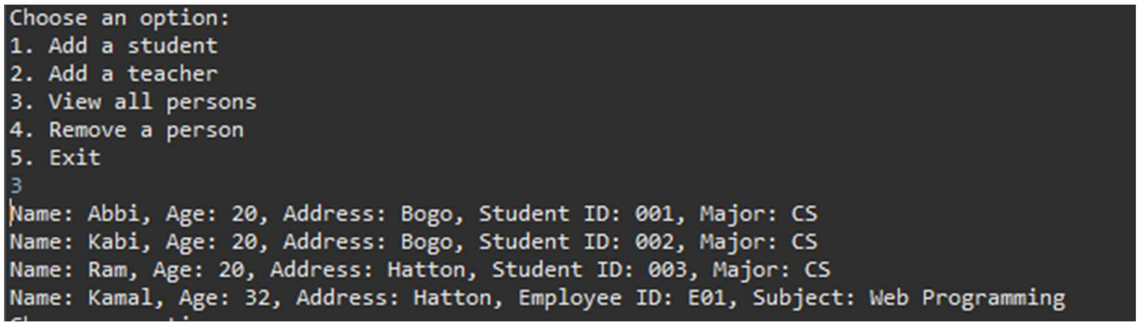
```
Main (2) [Java Application] C:\Program Files\Java\jdk-22\bin\javaw.exe (Jul 24, 2024, 10:02:40 AM) [pid: 17808]
Choose an option:
1. Add a student
2. Add a teacher
3. View all persons
4. Remove a person
5. Exit
1
Enter name: Ram
Enter age: 20
Enter address: Hatton
Enter student ID: 003
Enter major: CS
```

03 – Add a Teacher



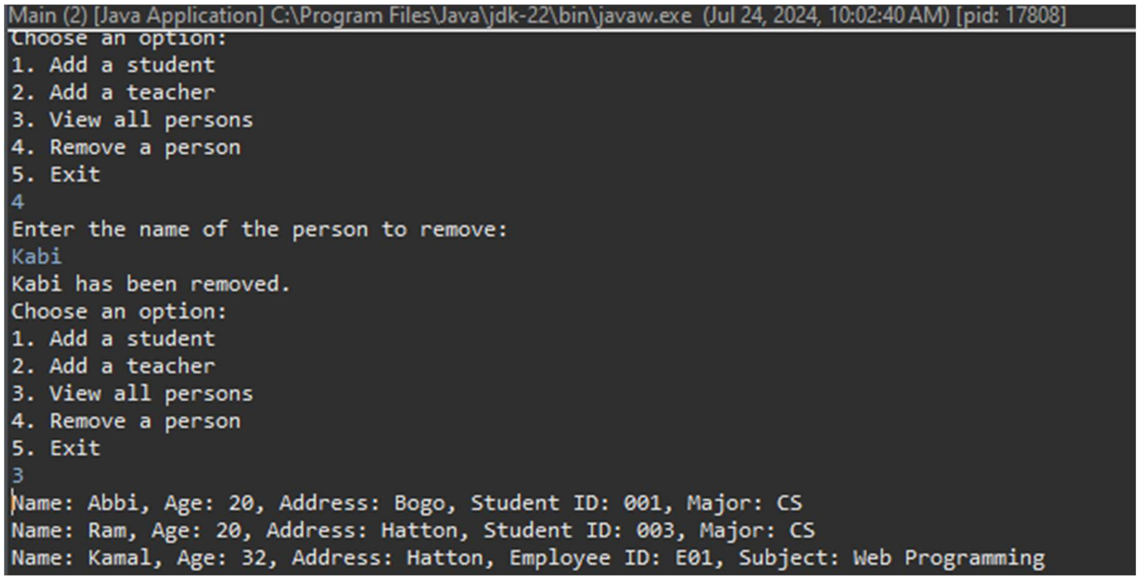
```
Choose an option:
1. Add a student
2. Add a teacher
3. View all persons
4. Remove a person
5. Exit
2
Enter name: Kamal
Enter age: 32
Enter address: Hatton
Enter employee ID: E01
Enter subject: Web Programming
Choose an option:
```

03 – View all Persons



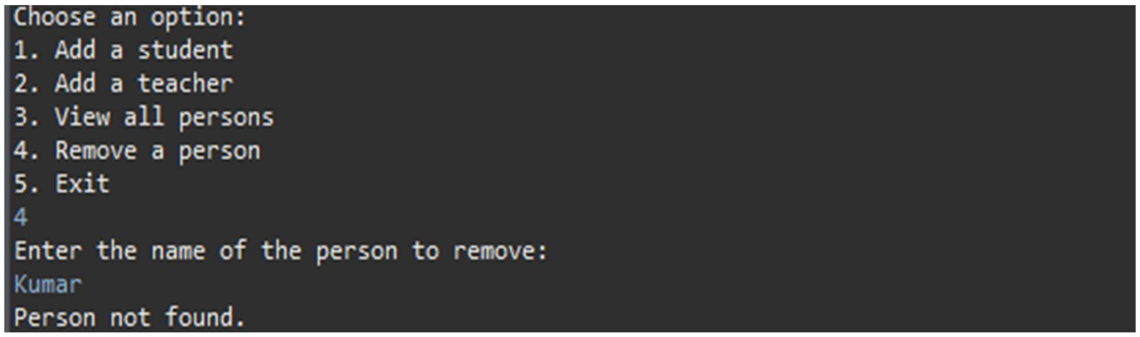
```
Choose an option:
1. Add a student
2. Add a teacher
3. View all persons
4. Remove a person
5. Exit
3
Name: Abbi, Age: 20, Address: Bogo, Student ID: 001, Major: CS
Name: Kabi, Age: 20, Address: Bogo, Student ID: 002, Major: CS
Name: Ram, Age: 20, Address: Hatton, Student ID: 003, Major: CS
Name: Kamal, Age: 32, Address: Hatton, Employee ID: E01, Subject: Web Programming
```

04 – Remove a Person



```
Main (2) [Java Application] C:\Program Files\Java\jdk-22\bin\javaw.exe (Jul 24, 2024, 10:02:40 AM) [pid: 17808]
Choose an option:
1. Add a student
2. Add a teacher
3. View all persons
4. Remove a person
5. Exit
4
Enter the name of the person to remove:
Kabi
Kabi has been removed.
Choose an option:
1. Add a student
2. Add a teacher
3. View all persons
4. Remove a person
5. Exit
3
Name: Abbi, Age: 20, Address: Bogo, Student ID: 001, Major: CS
Name: Ram, Age: 20, Address: Hatton, Student ID: 003, Major: CS
Name: Kamal, Age: 32, Address: Hatton, Employee ID: E01, Subject: Web Programming
```

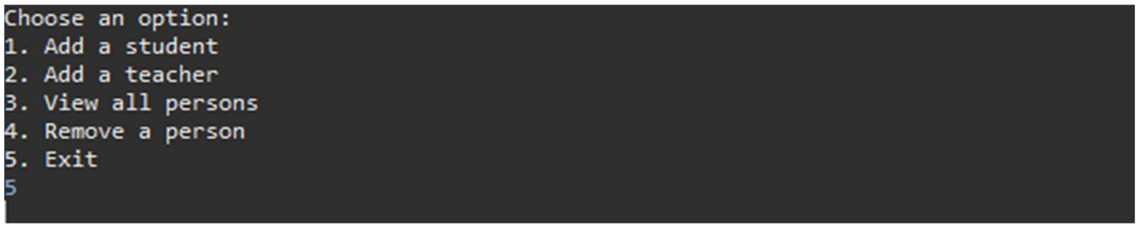
Wrong input will deliver a message called “**Person not found**”



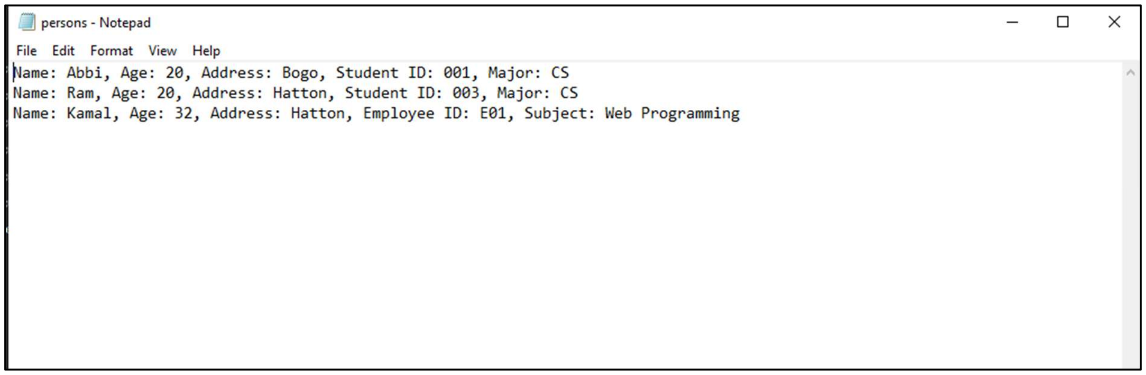
```
Choose an option:
1. Add a student
2. Add a teacher
3. View all persons
4. Remove a person
5. Exit
4
Enter the name of the person to remove:
Kumar
Person not found.
```

05 – Exit

> This operation will exit the program and your data will be recorded in text file called "**Persons.txt**".



```
Choose an option:
1. Add a student
2. Add a teacher
3. View all persons
4. Remove a person
5. Exit
5
```



```
Name: Abbi, Age: 20, Address: Bogo, Student ID: 001, Major: CS
Name: Ram, Age: 20, Address: Hatton, Student ID: 003, Major: CS
Name: Kamal, Age: 32, Address: Hatton, Employee ID: E01, Subject: Web Programming
```

• Conclusion

The School Management System project successfully demonstrates fundamental OOP principles using Java. It showcases the power of OOP in organizing and managing complex systems through **inheritance**, **polymorphism**, **encapsulation**, and **abstraction**.