School Management System

Project Documentation

- **Project Title:** School Management System
- **Course Name:** Object Oriented Programming
- **Submitted by:**
- **Submitted to:** Dr. Fahima Maghraby

2. Table of Contents

Introduction

The **School Management System** is an interactive console application designed to manage students and teachers within a school environment. It allows for adding, removing, and viewing students and teachers across classrooms and departments. This project aligns with **SDG 4: Quality Education**, aiming to enhance school management efficiency. It supports the shift towards a sustainable, paperless, and efficient educational environment, aligning with global environmental goals and Sustainable Development Goal 13: Climate Action.

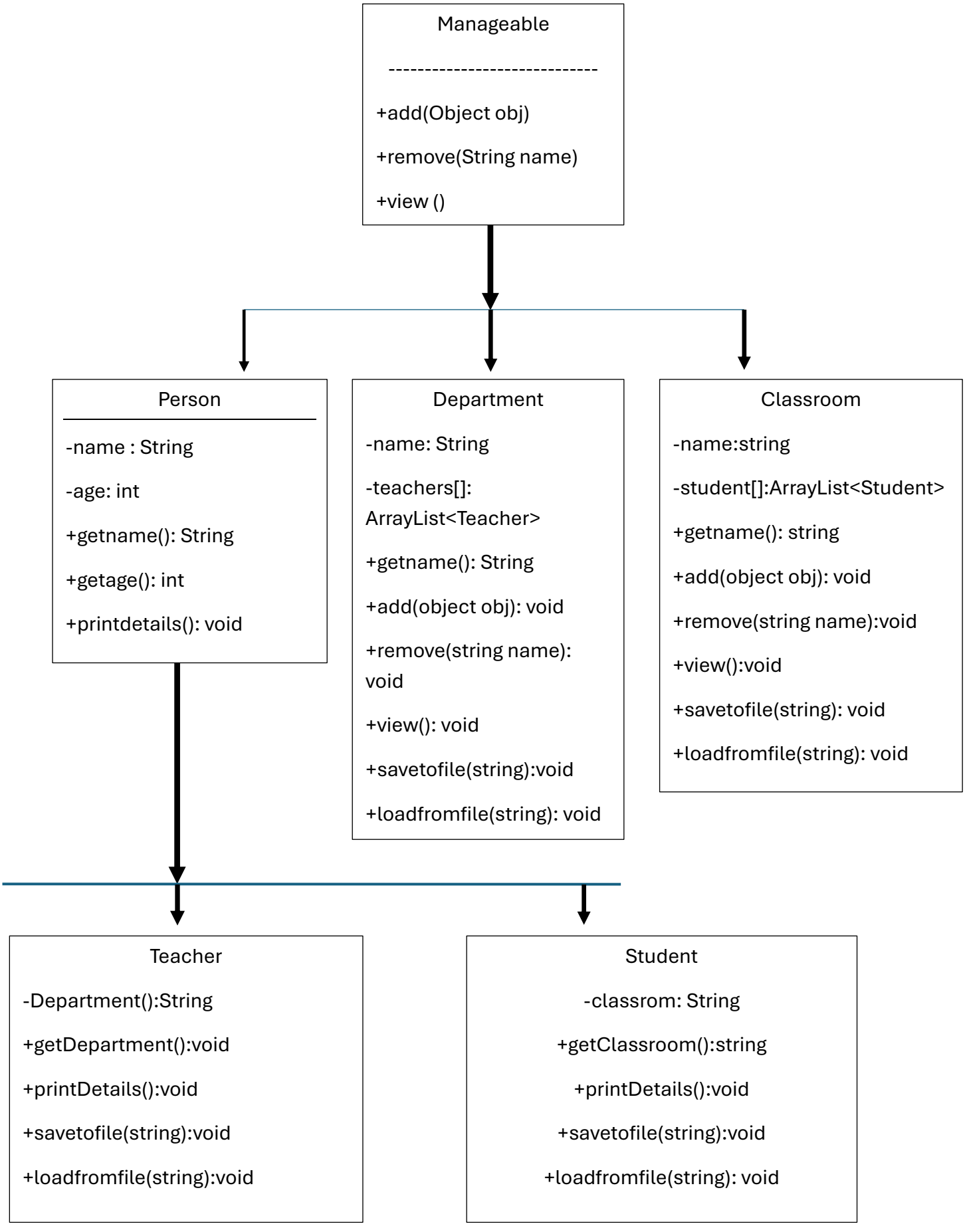
Problem Statement

Managing educational institutions manually is time consuming and error-prone. There is a need for a centralized system to simplify administrative tasks such as tracking students and teachers across different classrooms and departments.

Features of the System

- Add and remove students from classrooms
- Add and remove teachers from departments
- View all students in a specific classroom
- View all teachers in a specific department
- Save all data to files for persistence
- Load data from files on application startup

Class Diagram UML



System Design

Architecture

Encapsulation:

- **Classes like Teacher, Student, Department, and Classroom encapsulate data (such as name, age, and department) and provide methods to access or modify that data**
- **The private variables EX:(name, age, teachers, students) are encapsulated, and only getter methods EX:(getName(), getAge()) or specific methods EX:(add(), remove(), view()) are provided to interact with the data.**
- **The saveToFile() and loadFromFile() methods handle file I/O internally, hiding the implementation details from other parts of the system.**

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

public class Department implements Manageable {  17 usages
    private String name;  7 usages
    private ArrayList<Teacher> teachers;  10 usages

```

```java
import java.io.*;

public class Teacher extends Person {
    private String department;
```

```java
import java.io.*;

public class Student extends Person {
    private String classroom;
```

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

public class Classroom implements Manageable {  33 usages
    private String name;  7 usages
    private ArrayList<Student> students;  10 usages
```

```java
private void updateFile(String fileName) {  1 usage
    try (BufferedWriter writer = new BufferedWriter(new FileWriter(fileName))) {
        for (Teacher teacher : teachers) {
            writer.write(teacher.getName() + "," + teacher.getAge() + "," + teacher.getDepartment());
            writer.newLine();
        }
        System.out.println("File updated successfully.");
    } catch (IOException e) {
        System.out.println("Error updating file: " + e.getMessage());
    }
}
```

Inheritance:

The class Person is the superclass for both Teacher and Student, meaning both inherit common properties (name, age) and methods EX:(getName(), getAge()).

```java
public abstract class Person {
    protected String name;
    protected int age;

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

    public abstract void printDetails();

    public String getName() { return name; }

    public int getAge() { return age; }
}
```

Polymorphism:

- **The method printDetails() is overridden in both the Teacher and Student classes. Polymorphism allows each subclass to provide its own implementation of the method**
- **The add() method in Classroom and Department uses polymorphism by accepting an Object and then checking its type to determine if it should add a Student or Teache**

```java
    @Override
    public void printDetails() {
        System.out.println("Student: " + name + ", Age: " + age + ", Classroom: " + classroom);
    }
```

```java
@Override
public void printDetails() {
    System.out.println("Teacher: " + name + ", Age: " + age + ", Department: " + department);
}
```

```java
@Override
public void add(Object obj) {
    if (obj instanceof Teacher) {
        Teacher teacher = (Teacher) obj;

        for (Teacher existingTeacher : teachers) {
            if (existingTeacher.getName().equalsIgnoreCase(teacher.getName())) {
                System.out.println("Teacher " + teacher.getName() + " already exists in department.");
                return;
            }
        }

        teachers.add(teacher);
        System.out.println("Added teacher: " + teacher.getName() + " to department: " + name);
        teacher.saveToFile("teachers.txt");
    }
}
```

```java
public void add(Object obj) {
    if (obj instanceof Student) {
        Student student = (Student) obj;

        // Check if the student already exists before adding (case insensitive)
        for (Student existingStudent : students) {
            if (existingStudent.getName().equalsIgnoreCase(student.getName())) {
                System.out.println("Student " + student.getName() + " already exists in classroom.");
                return; // Avoid adding duplicates
            }
        }

        students.add(student);
        System.out.println("Added student: " + student.getName() + " to classroom: " + name);
        student.saveToFile("students.txt");
    }
}
```

Abstraction:

- **The Person class provides an abstract method printDetails() which is implemented by its subclasses Teacher and Student**
- **The Manageable interface abstracts the management methods (add(), remove(), view()) and provides a common contract for classes like Classroom and Department to follow, without needing to know the specific details of the implementation**

```java
public interface Manageable {
    void add(Object obj);
    void remove(String name);
    void view();
}
```

```java
    public abstract void printDetails();

    public String getName() { return name; }

    public int getAge() { return age; }
}
```

Composition:

- **The Department and Classroom classes both contain ArrayList objects (teachers and students) This means that a department or classroom is composed of multiple teachers or students, showing the "has-a" relationship, a core concept of composition.**
- **Methods like add(), remove(), and view() in these classes work with the ArrayList to manage the contained objects.**

```java
public class Classroom implements Manageable {  33 usages
    private String name;  7 usages
    private ArrayList<Student> students;  10 usages

    public Classroom(String name) {  33 usages
        this.name = name;
        this.students = new ArrayList<>();
        loadFromFile("students.txt"); // Load students when the classroom is created
    }
```

```java
public class Department implements Manageable {  17 usages
    private String name;  7 usages
    private ArrayList<Teacher> teachers;  10 usages

    public Department(String name) {  17 usages
        this.name = name;
        this.teachers = new ArrayList<>();
        loadFromFile("teachers.txt");
    }
```

Constructor and Initialization:

- **Constructors are used in the Department, Classroom, Teacher, and Student classes to initialize objects with specific values like name, age, and department/classroom.**
- **In the Department and Classroom constructors, file loading occurs to populate the list of teachers or students, ensuring the objects are properly initialized from the start.**

Error Handling:

- **The use of try-catch blocks in methods like saveToFile(), loadFromFile(), and updateFile() provides basic error handling, ensuring that the program does not crash when an error occurs while reading or writing files.**

```
71
72      public void saveToFile(String fileName) {  no usages
73          try (BufferedWriter writer = new BufferedWriter(new FileWriter(fileName, true))) {
74              for (Teacher teacher : teachers) {
75                  writer.write(teacher.getName() + "," + teacher.getAge() + "," + teacher.getDepartment());
76                  writer.newLine();
77              }
78              System.out.println("Department data saved to file.");
79          } catch (IOException e) {
80              System.out.println("Error writing to file: " + e.getMessage());
81          }
82      }
83
84      private void loadFromFile(String fileName) {  1 usage
85          try (BufferedReader reader = new BufferedReader(new FileReader(fileName))) {
86              String line;
87              while ((line = reader.readLine()) != null) {
88                  String[] data = line.split(",");
89                  if (data[2].equalsIgnoreCase(name)) {
90                      boolean exists = false;
91                      for (Teacher teacher : teachers) {
92                          if (teacher.getName().equalsIgnoreCase(data[0])) {
93                              exists = true;
94                              break;
95                          }
96                      }
```

Main Classes

Person

Attributes: name, age.

Abstract method: printDetails().

Teacher

Extends Person and adds to department.

Student

Extends Person and adds to classroom.

Classroom

Manages a list of students and handles file operations.

Department

Manages a list of teachers and handles file operations.

File Operations

Data is saved and loaded from teachers.txt and students.txt to maintain persistence**.**

The **Main** class provides a menu with the following options:

- Add or remove a student from a classroom.
- Add or remove a teacher from a department.
- View all students or teachers in a specific classroom or department.

Screenshots of Test Cases

screenshots of the following test cases with their outputs:

- **Adding a student to a classroom.**

```
            switch (choice) {
                case 1:
                    System.out.println("Enter student name:");
                    String studentName = scanner.nextLine();
                    System.out.println("Enter student age:");
                    int studentAge = scanner.nextInt();
                    scanner.nextLine();
                    System.out.println("Enter student classroom:");
                    String studentClassroom = scanner.nextLine();
                    Student student = new Student(studentName, studentAge, studentClassroom);
                    if (studentClassroom.equals("A1")) {
                        classroomA1.add(student);
                    } else if (studentClassroom.equals("A2")) {
                        classroomA2.add(student);
                    } else {
                        System.out.println("Invalid classroom.");
                    }
                    break;
```

- **Removing a student from a classroom**

```
                case 2:
                    System.out.println("Enter student name to remove:");
                    String studentToRemove = scanner.nextLine();
                    System.out.println("Enter classroom:");
                    String removeClassroom = scanner.nextLine();
                    if (removeClassroom.equals("A1")) {
                        classroomA1.remove(studentToRemove);
                    } else if (removeClassroom.equals("A2")) {
                        classroomA2.remove(studentToRemove);
                    } else {
                        System.out.println("Invalid classroom.");
                    }
                    break;
```

- **Adding a teacher to a department**

```

                case 3:
                    System.out.println("Enter teacher name:");
                    String teacherName = scanner.nextLine();
                    System.out.println("Enter teacher age:");
                    int teacherAge = scanner.nextInt();
                    scanner.nextLine();
                    System.out.println("Enter department:");
                    String teacherDepartment = scanner.nextLine();
                    Teacher teacher = new Teacher(teacherName, teacherAge, teacherDepartment);
                    if (teacherDepartment.equals("math")) {
                        mathDepartment.add(teacher);
                    } else if (teacherDepartment.equals("english")) {
                        englishDepartment.add(teacher);
                    } else {
                        System.out.println("Invalid department.");
                    }
                    break;
```

- **Removing a teacher**

```
76
77              case 4:
78                  System.out.println("Enter teacher name to remove:");
79                  String teacherToRemove = scanner.nextLine();
80                  System.out.println("Enter department:");
81                  String removeDepartment = scanner.nextLine();
82                  if (removeDepartment.equals("math")) {
83                      mathDepartment.remove(teacherToRemove);
84                  } else if (removeDepartment.equals("english")) {
85                      englishDepartment.remove(teacherToRemove);
86                  } else {
87                      System.out.println("Invalid department.");
88                  }
89                  break;
90
```

- **Viewing all teachers in a department**

```
103                 case 6:
104                     System.out.println("Enter department to view teachers:");
105                     String departmentToView = scanner.nextLine();
106                     if (departmentToView.equals("math")) {
107                         mathDepartment.view();
108                     } else if (departmentToView.equals("english")) {
109                         englishDepartment.view();
110                     } else {
111                         System.out.println("Invalid department.");
112                     }
113                     break;
114
```

- **Viewing all students in a classroom**

```
90
91                  case 5:
92                      System.out.println("Enter classroom to view students:");
93                      String classroomToView = scanner.nextLine();
94                      if (classroomToView.equals("A1")) {
95                          classroomA1.view();
96                      } else if (classroomToView.equals("A2")) {
97                          classroomA2.view();
98                      } else {
99                          System.out.println("Invalid classroom.");
100                     }
101                     break;
102
```

Conclusion

The **School Management System** demonstrates the practical application of object-oriented principles to solve real-world problems. It efficiently automates school administrative tasks while ensuring data persistence and ease of use