TABLE OF CONTENTS

PRINTED SOURCE CODE

Printed Source Code reside inside these following directories:

- PrintedSourceCode_PDF
- PrintedSourceCode_TXT

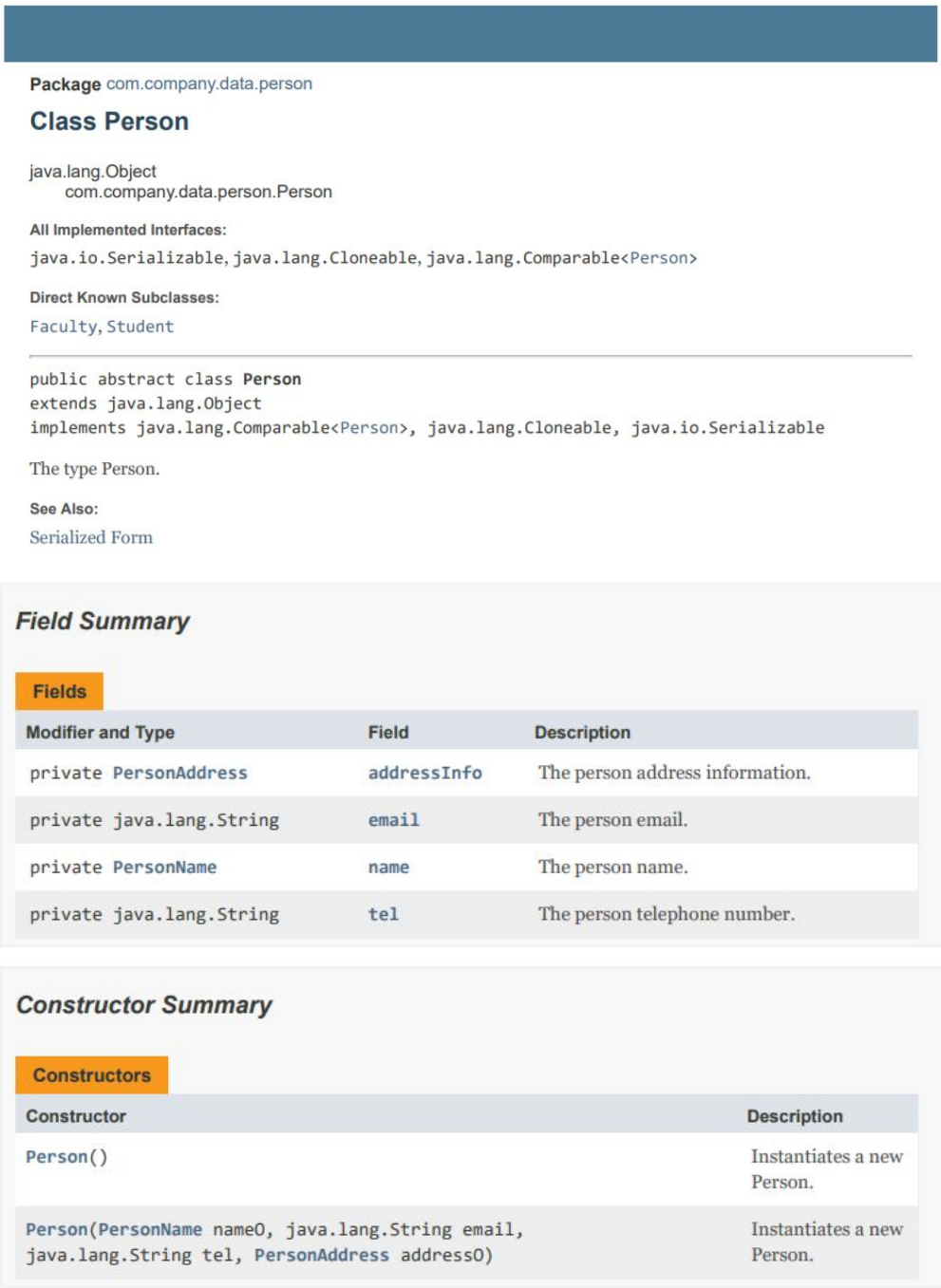
(Printed) source code are organized in a way that demonstrates the use of packages in the project:

```
C:.
    CourseSchedulerMain.pdf

---comparators
        CourseComparator.pdf
        FirstMidLastNameComparator.pdf
        LastMidFirstNameComparator.pdf
        SessionComparator.pdf

---data
    ---course
            Course.pdf
            CourseDirectory.pdf
            Session.pdf

    ---person
            Faculty.pdf
            Person.pdf
            PersonAddress.pdf
            PersonDirectory.pdf
            PersonName.pdf
            Student.pdf

---inputFiles
        1A_session_id_input.txt
        1B_session_id_input.txt
        1C_session_id_input.txt
        1D_session_id_input.txt
        4A_session_id_input.txt
        info_course_input.txt
        info_faculty_input.txt
        info_student_input.txt
        package-info.java
        S001_wish.txt
        S002_wish.txt
        S003_wish.txt
        S004_wish.txt
        S005_wish.txt
        S006_wish.txt
        S007_wish.txt
        S008_wish.txt
        S009_wish.txt
        S010_wish.txt
        TestRun.txt
```

DETAILED JAVADOC DOCUMENTATION

Sample page of the javadoc documented class: Person Class

Snippet of the page:



Package com.company.data.person

Class Person

java.lang.Object
    com.company.data.person.Person

All Implemented Interfaces:

`java.io.Serializable, java.lang.Cloneable, java.lang.Comparable<Person>`

Direct Known Subclasses:

`Faculty, Student`

```
public abstract class Person
extends java.lang.Object
implements java.lang.Comparable<Person>, java.lang.Cloneable, java.io.Serializable
```

The type Person.

See Also:

Serialized Form

Field Summary

Fields

Modifier and Type	Field	Description
`private PersonAddress`	`addressInfo`	The person address information.
`private java.lang.String`	`email`	The person email.
`private PersonName`	`name`	The person name.
`private java.lang.String`	`tel`	The person telephone number.

Constructor Summary

Constructors

Constructor	Description
`Person()`	Instantiates a new Person.
`Person(PersonName name0, java.lang.String email, java.lang.String tel, PersonAddress address0)`	Instantiates a new Person.

PROJECT REPORT

Your high-level approach

Answer these starter questions:

- What does this program do?
    - Schedule courses for
        - Students to enroll/unenroll courses they want to take
        - Teachers to assign/unassign courses for themselves
- What are the key elements?
    - Courses, sessions
    - Students, Teachers
- How do these elements relate, what are their relationships?
    - Each course has a list of their sessions
    - Each session has a teacher, a list of students enrolled in the session
    - Students and Teachers both have basic personal information, added with their corresponding specialized information; can be grouped together.

Your algorithms

- Use of Inheritance: Both Student and Teacher extend from Person
- Aggregation, Composition, Association:
    - Composition: Person vs. PersonName and PersonAddress
    - Aggregation: Student/Faculty vs. Course and Session
    - Association:
        - PersonDirectory vs. Person, Student, Faculty
        - CourseDirectory vs. Course
- Utilize dynamic arrays using ArrayList to keep track of Persons, Students, Faculties, Courses, Sessions; Use StringBuilder for their mutable (modifiable) feature.
- Exception Handling: NullPointerException, IndexOutOfBoundsException, FileSystemNotFoundException
- Use interface (Comparable, Clonable, Serializable) and override Object methods:
    - toString, equals, compareTo, hashCode, clone
- Use Comparator to enable sorting Person, Course, Session by names.
- Unique ids are determined as the following:
    - Student: Prefix “S” + # of students in the system. (S001 for first student).
    - Faculty: Prefix “F” + # of faculties in the system. (F001 for first faculty).
    - Session: Prefix courseCode + # of sessions in course (1A001 for the first session).

Key design decisions you made and why

UML Class Diagram:

- Break the code down for smaller/specific task/implementation:
    - PersonName Class
    - PersonAddress Class
- Inheritance: Both Student and Teacher extend from Person
- Create PersonDirectory and CourseDirectory to keep track of persons and courses
- Use of package for organization

What you learned by doing this exercise

- I've learned to use Inheritance, Encapsulation, Polymorphism
- I've learned to use interfaces (Comparable, Comparator, Clonable, Serializable)
- I've learned to use ArrayList, StringBuilder, FileOutputStream, Exception Handling
- I've learned to organize code into packages
- I've learned to use Javadoc and enhance my understanding of UML Diagrams
- I've learned to make my own input files for testing
- I've learned how to design code/class so that they can simplify and work together
- I've learned to "customize" my program because there was no expected output file.

Rate the exercise from 1 to 10 and explain why

I would give it an 8/10 because the project does a good job testing my understanding and ability to use everything I have learned during the semester.

The project is big but manageable, the only thing I found quite tedious is making our own input files. Creating the input files was very time consuming and very error-prone even though we are only doing a small number of what the actual prompt is asking. If there is a better way to design and create the test input files, I would like to be introduced to it.

INPUT AND OUTPUT DOCUMENTATION

Input Documentation

Input files in the current working program include
File format: Information are separated using regex "_" (underscore)

- All student information: info_student_input.txt

```
firstName_middleName_lastName_email_telephone_street_city_state_zipCode_
studentId_dateOfBirth_currentGPA_attendStartDate
```

- All faculty member information: info_faculty_input.txt

```
firstName_middleName_lastName_email_telephone_street_city_state_zipCode_
facultyId_dateHired_tenuredStatus
```

- All course information: info_course_input.txt

```
department_code_description_courseId_minStudent_maxStudent
```

- The number of sessions with session id to schedule for each course:
  - 1A_session_id_input.txt, 1B_session_id_input.txt, …

```
session1Id_session2Id_session3Id_session4Id_session5Id
```

- All of the courses each student wishes to take by course id:
  - S001_wish.txt, S002_wish.txt, …

```
course1Id_course2Id_course3Id_course4Id_course5Id
```

Output Documentation

Output files are displayed as a readable text file with heading of what to be outputted. Each output file adheres to their corresponding requirements:

- ScheduledCourseSessions.txt:
    - Each course that was scheduled (full details)
    - Each session for that course (i.e. session id)
    - The full name and id number of the instructor for the session
    - The number of students in the session
    - For each student in the session, the full name and id number.

- UnscheduledCourseSessions.txt:
    - Each course that was not scheduled (i.e. not even one session scheduled) along with the minimum number of students that needed to be in the course.

- Faculty.txt:
    - Print all details for each faculty member and list the full course details of each course and session he/she has scheduled along with the session id and number of students in that session

- ScheduledStudents.txt:
    - For each student show all details and list the session id, course id, and course description for the classes taken.

- UnscheduledStudents.txt:
    - List full details for each student that has no classes to take.

TEST CASES - UNIT TESTING

1. TestCase1_LoginAs:
    - This test case tests if the user can successfully log in as an Admin, Faculty, Student and view all the corresponding functionalities of their role.

2. TestCase2_LoginAsAdmin:
    - This test case tests if an Admin can successfully use all of their features.

3. TestCase3_LoginAsFaculty:
    - This test case tests if a Faculty can successfully use all of their features.

4. TestCase4_LoginAsStudent:
    - This test case tests if a Student can successfully use all of their features.

5. TestCase5_TestForOutputFiles:
    - This test case tests if the output files are outputted and are corrected with the specific inputs via Student Login and Faculty Login.