Java Project

In Brief...

For this Java project, you will create a Java program for a school. The purpose is to create a report containing one or more classrooms. For each classroom, the report will contain:

- The room number of the classroom.
- The teacher and the subject assigned to the classroom.
- A list of students assigned to the classroom including their student id and final grade.

Instructions

Application Structure

Projects should be organized into libraries or packages of classes that fall into general categories. This project can be divided into two packages:

1. school which contains JavaBeans that naturally fall into that category.
2. util which contain general purpose classes.

In your project directory create the listed subdirectories. They will be the packages used for the JavaBeans, interfaces, and utilities required:

- util
- school

The util Package Files

Your class files for your course supplied the `KeyboardReader` class. Copy the KeyboardReader.java into the util package.

In the util package, create the `Displayable` interface. The interface should declare one method as follows:

```
public abstract String display()
```

The following shows the directories and files you should have at this point:

- Project Directory
  - util package
    - KeyboardReader.java
    - Displayable.java
  - school package

The school Package Files

Object-Oriented programs become more maintainable, flexible, and saleable when programmers use inheritance, encapsulation, polymorphism, and abstraction. The following descriptions of the classes in the school package should leverage these principles.

Create the Person JavaBean

In the school package, create the Person JavaBean. Make it an **abstract** class. Declare the following instance variables:

- `String firstName`
- `String lastName`

Include the **getter** and **setter** methods for each variable. Use the **camel case** naming convention for JavaBean methods and variables. Include a method named `getFullName()` that returns both names concatenated into a **String** with a space between the first and last names.

Create the Teacher JavaBean class

Create the `Teacher` class in the school package. It inherits the `Person` abstract class and

implements the `Displayable` interface. It defines only one variable as follows:

- `String subject`

Include the **getter** and **setter** method for the variable. Use the **camel case** naming convention for JavaBeans. Provide a **no argument constructor**. Provide another constructor that uses the following parameters to initialize the variables:

- `String firstName`
- `String lastName`
- `String subject`

Override the `display()` method. It should return a **String** containing the teacher's full name using the `getFullName()` method defined in `Person` and the subject taught as follows:

```
Roger Sakowski teaches English.
```

Create the Student JavaBean Class

Create the `Student` class in the school package. It inherits the `Person` abstract class and implements the `Displayable` interface. It defines two variables:

- `int studentId`
- `int finalGrade`

Include the **getter** and **setter** methods for the variables. Use the **camel case** naming convention for JavaBeans. Override the `display()` method. It should return a **String**containing the student's id, the student's full name using the `getFullName()` method defined in `Person`, and the student's final grade as follows:

```
Student ID: 1    John Doe Final Grade: 90
```

Create the Classroom JavaBean Class

Create the `Classroom` class in the school package. It implements the `Displayable` interface. It defines three instance variables:

- `int roomNumber`

- `Displayable teacher` (note that the `Teacher` instance is downcast to the `Displayable` interface)
- `ArrayList<Displayable> students` (note that the `Student` instances in the list are downcast to the `Displayable` interface)

Provide a **no argument constructor**. Provide another constructor that uses the following parameters to initialize the variables:

- `int roomNumber`
- `Displayable teacher`
- `ArrayList<Displayable> students`

The packages and files you have created so far should look like the following:

- Project Directory
  - util package
    - KeyboardReader.java
    - Displayable.java
  - school package
    - Person.java
    - Teacher.java
    - Student.java
    - Classroom.java

Programming Logic

The PrintReports class

A well designed program depends on source code that not only does the job, but does it in a highly maintainable and efficient way. There should never be blocks of duplicate code and methods should be simple and designed to one thing. The **PrintReports** class will contain most of the programming logic for this project. Organization of methods and their responsibilities are the focus of this section.

Create PrintReports

In your project directory create the `PrintReports` class.

- Project Directory

    - **PrintReports.java**
    - util package
        - KeyboardReader.java
        - Displayable.java
    - school package
        - Person.java
        - Teacher.java
        - Student.java
        - Classroom.java

It will define the `main()` method.

Create support methods signatures

`PrintReports` should define the following methods using the listed method signatures:

- `public Displayable enterClassroom()`
- `public Displayable enterTeacher()`
- `public Displayable enterStudent()`
- `void report(ArrayList<Displayable>)`

You can leave them as skeleton code for now. We will cover the logic they should contain in turn.

Working with non-static methods

Note that the methods are not **static**. One way to escape the **static** requirement `main()` imposes is to use this approach:

```
public static void main(String[] args) {
        new PrintReports();
}

public PrintReports(){
        // Your code goes here
}
```

The public PrintReports() Constructor

In a `do…while` loop collect the data need to create a `Classroom` object using the `enterClassroom()` method. You should be able to create any number of `Classroom`objects. Prompt the user so he or she can enter another `Classroom` or quit the loop. Store the `Classroom` objects in an `ArrayList<Displayable>` collection.

The public Displayable enterClassroom() Method

Using `KeyboardReader`, prompt the user for a room number. Save it as an **int**. The room number must be **100 or greater**. If the user enters a lower number, he or she should be prompted again until an acceptable number is entered.

Call `enterTeacher()` to obtain an instance of a teacher and store it as a `Displayable` object.

In a `do…while` loop, call `enterStudent()` to obtain a Student as a Displayable object and store it in an `ArrayList<Displayable>` collection. Prompt the user so he or she can enter another student or quit the loop.

The public Displayable enterTeacher() Method

The method should prompt the user using **KeyboardReader** for their first and last name as well as the subject they teach. Create an instance of **Teacher** using that data and return the object as an instance of **Displayable**.

The public Displayable enterStudent() method

Prompt the user for the student id, first and last names, and their final grade. Using that data, create a `Student` instance. A student's id must be greater than 0. A student's final grade must be between 0 and 100. Return the Student object as a `Displayable` object.

The void report(ArrayList<Displayable>) Method

In a `for` loop, iterate through the `ArrayList<Displayable>` collection containing the downcast `Classroom` objects.

Call the `display()` method defined in `Classroom`. It should report the room number.

It should call the `display()` method in the `teacher` variable to report the teacher assigned to the classroom.

In a for loop it should iterate through the `ArrayList<Displayable>` collection of `Student` objects calling the `display()` method for each one.

The Report Example

The following is an example of output produced by the `report()` method. For brevity, it only demonstrates one classroom containing one student. Your program should allow you to create multiple classrooms and multiple students per classroom.

```
First You Need To Create A Classroom
Enter Room Number: 101

Now You Need To Enter A Teacher For The Classroom.
Enter Teacher First Name: Sam
Enter Teacher Last Name: Huston
Enter Subject Taught: English

Now You Need To Add Students For The Classroom
Enter Student First Name: Sally
Enter Student Last Name: Jones
Enter Student ID: 1
Enter Student Final Grade: 90

Enter Another Student? (Y/N): n (y should prompt for a new student)

Enter Another Classroom? (Y/N): n (y should prompt for a new classroom)

-------------------------------------------------------------
Room Number: 101
Sam Huston teaches English
Student ID: 1   Sally Jones     Final Grade: 90
-------------------------------------------------------------
```

Grading:

Source Code: 5%

- Project compiles without errors or warnings: 5%

Packages: 10%

- Project uses packages: 5%
- PrintReports uses import statements: 5%

Encapsilation: 10%

- Person, Student, Teacher, and Classroom classes declare private variables: 5%

- Person, Student, Teacher, and Classroom classes declare public getter and setter methods: 5%

Inheritance: 5%

- Student and Teacher extends Person: 5%

Abstraction: 15%

- Displayable interface declares display method: 5%
- Person is an abstract class: 5%
- Student and Teacher implements Displayable: 5%

Polymorphism: 20%

- Student and Teacher overrides display method: 5%
- Person, Student, Teacher, and Classroom classes declare private variables: 5%
- Student and Classroom are stored in ArrayList<Displayable> collections: 5%
- Teacher is stored in a Displayable variable: 5%

Implementation: 35%

- PrintReport contain the main method: 5%
- The main method is the only member of the class declared as static: 5%
- User is able to enter multiple Classroom instances: 5%
- User is able to enter multiple Student instances for each Classroom: 5%
- Classroom room number must be greater than 100: 5%
- Program should output a final report of each Classroom: 5%
- Program should output a final report listing Students per Classroom: 5%

Rules

This project is meant for you to use your own skills and knowledge. This means that we expect the work to be your own work. We also expect that you will want to look some stuff up. Please feel free to use your course manual and the course content to help you along. You may also use the Internet as a source of help, especially for looking up documentation and errors.

Note that the instructor is not a resource during this project. The purpose of the project is to evaluate how well you can do without access to the instructor.

Submitting Project

- After you have completed all of the requirements of the project, create an archive (zip file) of your entire project "java-dev" folder. You can use 7-zip, or WinRAR, or WinZip, or whatever compression tool you prefer to create the zip file. We recommend 7-zip as it is free.
- Email the file to project@webucator.com (mailto:project@webucator.com).
- If you are having trouble sending the project attachment via email we suggest using a file sharing service such as Dropbox, Box.net or Google Drive.
- Please allow 10 business days for us to review the exam submission.
- You must complete the project before the expiration date of your course (12 months after you started). We estimate it will take about 20 hours. Be sure to leave yourself enough time.