Project guide

We are going to implement an application that will produce reports of different types. We are going to have Person- and Company-classes and also PersonReport- and CompanyReport-classes. But we will be prepared to

- Produce reports to different medias (Screen and File). These will be handled by different "Reporters"
- Produce reports formatted in various manners (Tabular and Html). This will be done by different "Formatters"

We will implement our solution in a manner we can later on implement new reports, medias and formattings into our solution as easily as possible, hiding complexities into the base-classes. However, we will reach that target in smaller steps.

Step 1

Let's start by implementing a ScreenReporter-class that knows how to print tabular data to screen. You have to be able to define the column titles for the report and then print the actual data. In the main the following should work:

```
ScreenReporter rep=new ScreenReporter();
rep.addColumn("Name",20);
rep.addColumn("Age",5);
rep.printColumns();
rep.printData("John Wayne");
rep.printData(82);
rep.printData("Ronald Reagan");
rep.printData(93);
```

And produce output like:

```
Name                Age
John Wayne          82
Ronald Reagan       93
```

So basically, you need to implement addColumn- and printData-methods to the ScreenReporter-class.

Internally addColumn just stores information passed to the method (column title and width) to an ArrayList. It might be convenient to create an extra class ColumnDef that just holds title and width for single column.

Couple of versions of printData are needed taking different types of parameters (String and int are needed so far). You also need a counter (which column is being printed) so that you can print the linefeed after last column.

Figure out (simplest possible) mechanism to make each column as wide as requested in addColumn.

OK, now we notice that we are going to repeatedly report Person-data. Let's create Person-class with fields name (String) and age (int). Encapsulate fields with accessors to create a JavaBean. Also create a constructor

with whom you may set the name and age. (Alternatively name and birthday when age should be a calculated property with just a getter). Feel free to add couple extra fields too.

Also create PersonReport-class so that, in the main, you may:

```
PersonReport pr=new PersonReport();
pr.addData(new Person("John Wayne",82));
pr.addData(new Person("Ronald Reagan",92));
pr.doReport();
```

And it should produce similar report to the one above.

The PersonReport-class will hold a reference to the Reporter in a field. Likewise it will maintain the list of persons. The constructor of PersonReport should specify the colums

Also create Company-class with fields name (String), phone (String), email (String) and CompanyReport-class in similar manner the PersonReport was implemented.

```
CompanyReport cr=new CompanyReport();
cr.addData(new Company("Coders Unlimited","555-234234","info@coders.net"));
cr.addData(new Company("Testers united","555-123123","info@testers.com"));
cr.doReport();
```

Should produce:

```
Company             Phone       Contact email
Coders Unlimited    555-234234  info@coders.net
Testers united      555-123123  info@testers.com
```

Step2

Create a FileReporter-class. To start with you may make an exact copy of ScreenReporter implementation. But instead of writing to System.out you should open a file into which you write.

- Constructor takes filename as parameter
- Implement beginReport that opens the file, store out-reference to a field
- endReport that closes the file

It might be most convenient to open the file in the following manner:

```
out=new PrintWriter(new FileWriter(fileName));
```

The Report-classes should call beginReport before printing actual data and endReport after printing. Finally you should be able to just change the Reporter-object the Reports use into FileReporter.

OK, now the code is clumsy. Both reporters contain a huge amount of similar code. First implement:

```
public interface Reporter {
```

```
    void beginReport();
    void endReport();

    void addColumn(String title, int width);
    void printData(String data);
    void printData(int data);
}
```

Then implement base class ReporterBase for both ScreenReporter and FileReporter. It should implement the Reporter-interface. Figure out how much of the common functionality you can place into the base-class. Remember that the base class can be marked abstract. In fact, the final implementation of ScreenReporter should resemble the following (FileReporter is only slightly more complicated)

```
public class ScreenReporter extends ReporterBase{
    protected PrintWriter getWriter(){
        return new PrintWriter(System.out);
    }

    protected void closeWriter(PrintWriter pw){
    }
}
```

In CompanyReport and PersonReport change the reference type of the field into Reporter. Modify the constructor of both classes so that it can take the Reporter as parameter. And you should be able to:

```
    ScreenReporter repScreen=new ScreenReporter();
    FileReporter repFile=new FileReporter("report.txt");

    PersonReport pr=new PersonReport(repScreen);
    pr.addData(new Person("John Wayne",82));
    pr.addData(new Person("Ronald Reagan",92));
    pr.doReport();

    CompanyReport cr=new CompanyReport(repFile);
    cr.addData(new Company("Coders Unlimited","555-234234","info@coders.net"));
    cr.addData(new Company("Testers united","555-123123","info@testers.com"));
    cr.doReport();
```

Final touch on this step. Create common base class for all Reports so that the above main still works but the implementation of Report-classes looks something like this:

```
public class PersonReport extends ReportBase<Person> {

    public PersonReport(Reporter rep){
        super(rep);
    }

    protected void addColumns(Reporter reporter){
        reporter.addColumn("Name",20);
```

```
        reporter.addColumn("Age",5);
    }


    protected void printData(Reporter reporter, Person p){
        reporter.printData(p.getName());
        reporter.printData(p.getAge());
    }
}
```

Get the idea? Creating of new Report-classes should be as easy as possible. Most of the stuff is handled by the base class.

Step 3

Report knows what data to report. Reporter knows the media to which data is to be reported. Report can work with any Reporter and Reporter can work with any Report. But we still need different formats. Let us introduce Formatter concept.

```
public interface Formatter {
    String begin(List<ColumnDef> defs);
    String end(List<ColumnDef> defs);
    String row(List<ColumnDef> defs,List<String> data);
}
```

Reporters should know the formatter. When the reporter is instantiated, the Formatter is given as constructor parameter. When printing the title row the Reporter queries the string from the formatter and just displays it. When printing a data row the Reporter again queries the entire row from the formatter and just displays it. After printing the data the Reporter queries from the formatter what should be printed at the end of the report.

First implement TabularFormatter. It should format each row in similar manner to the original solution. Just change the Reporter to use the Formatter instead of formatting the data itself. You should be able to:

```
        TabularFormatter tf=new TabularFormatter():
        ScreenReporter repScreen=new ScreenReporter(tf);
        FileReporter repFile=new FileReporter(tf,"report.txt");
```

Then implement HtmlFormatter. It should produce something like follows:

```
<table>
<tr><td>Name</td><td>Age</td></tr>
<tr><td>John Wayne</td><td>82</td></tr>
<tr><td>Ronald Reagan</td><td>92</td></tr>
</table>
```

Step 4

Now extending our reporting solution has become flexible, but the main is a mess. We have to instantiate all kinds of classes before we can actually do the reporting. Wouldn't it be nice if we could just:

```
Report pr=Report.create(Person.class, Report.TABULAR);
pr.addData(new Person("John Wayne",82));
pr.addData(new Person("Ronald Reagan",92));
pr.doReport();

Report cr=Report.create(Company.class, Report.HTML,"report.txt");
cr.addData(new Company("Coders Unlimited","555-234234","info@coders.net"));
cr.addData(new Company("Testers united","555-123123","info@testers.com"));
cr.doReport();
```

To accomplish the above you need to create a base class to ReportBase, ie ReportBase<T> extends Report<T>. The create-methods implemented are so called Factory Methods. They hide the complexities of instantiating the concrete report, but also they hide the actual report-implementation. The user of the report doesn't know the actual class implementing the Person report or company report.

Add couple of features:

- Sort data before printing it. The report user (the main) should decide the order of the data
- Filter the data before printing it. The report user should decide the criteria used