Algorithms and Data Structures

Functional Interfaces and Streams in a Project Planning System(PPS)

Assignment-4

Version: 6 November 2020

Introduction

In this assignment you will demonstrate the use of the Java Functional Interface and its application in combination with Streams and Collectors as provided by the Java Collection Framework. You will build a small, simplified Project Planning System (PPS) which tracks data about projects and employees who have been assigned to work on those projects. You will implement code to build the data structure and extract information from it.

A starter project is provided along with this assignment. It provides some // TODO annotations in the PPS, Employee and Project classes hinting where code is missing. These classes also include methods to import sample data from XML files. *You should not need to worry or change anything in those XML methods. Also, no changes are required in the classes of the 'utils' package*.

Sample XML files are provided in the resources folder in combination with an pps.xsd schema file. You can edit, validate and use your own xml sample data if you wish.

Some unit tests are provided to test parts of your solution, but you should not expect them to cover all possible mistakes in a solution. So it is wise to add some additional unit-tests!

Specification of the PPS.

In Figure 2 you find the UML class diagram specifying the PPS. For sake of simplicity, no visibility indicators are included for attributes or methods and Java Collection interface types are specified for relevant data types.

A Project is uniquely identified by a project code. The start and end dates specify the first and the last working day of the project. Employees are expected to only work on weekdays and not on Saturday or Sunday. Other free days (holidays etc.) are not considered in this assignment.

For every project we register the committed work hours per day per Employee. We assume that an employee who is assigned to the project will work the same number of hours on every project working day between its start and its end. (You may consider this the average project task load of an employee.) Employees may work on multiple projects in the same period of time. If an employee has to work more than 8 hours on one day in total, he or she has to work overtime.

Employees are uniquely identified by a number. For every employee we have recorded an hourly wage. This is the charge rate per hour that the employee works on a project. This rate is the same for all projects. Junior employees have an hourly wage <= 26.

You can calculate a project's total manpower budget (= total employee cost / wage) from:

a) The total number of working days between project start and project end,
b) the hours per day that each employee works for the project and
c) the hourly wage of each assigned employee.

Every project has one project manager. For every employee we track the projects that this employee is managing. That way you can calculate the total budget responsibility of each employee.

The assignment.

The core of the assignment is to complete the method

0. PPS.printPlanningStatistics()

such that it provides the answer to seven, key questions about the data in the PPS. In Figure 1 you find an example of expected output of PPS.printPlanningStatistics(). We are particularly interested in your results with the input file 'HVA2018_e50_p100.xml'. The main program Main01_PPS is already setup to produce your results…

In order to calculate those outcomes, you should complete six corresponding methods:

```
1. PPS.calculateAverageHourlyWage()
2. PPS.calculateLongestProject()
3. PPS.calculateMostInvolvedEmployees()
4. PPS.calculateTotalManpowerBudget()
5. PPS.calculateManagedBudgetOverview(employeeFilter)
6. PPS.getFulltimeEmployees()
7. PPS.calculateCumulativeMonthlySpends()
```

Successful solutions of above methods will certainly depend on correct functioning of the Project and Employee classes. You first need to complete and fix the basics of these classes and also complete three of their methods:

```
8. Project.addCommitment(employee, hoursPerDay)
9. Project.calculateManpowerBudget()
10.Employee.calculateManagedBudget()
```

We also ask you to complete the

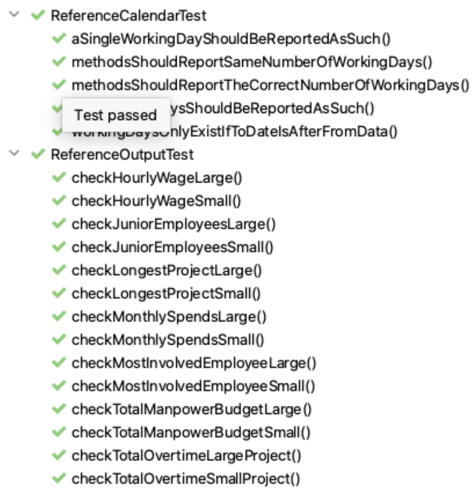
```
11.PPS.Builder()
```

local class which provides a builder pattern for small PPS instances using method-chaining.
This class is verified by one of the unit tests.

Unit tests.

The unit tests that are part of the provided start-project are by no means complete. They serve as a starting point which you are encouraged to extend with more tests, since that can lead to a higher grade. On the right you see a part of an extended test-set that will be used during the grading of your solution. The names of the test methods might inspire you when adding extra unit tests.

Obviously, we have designed the unit tests to be independent of any correct solution approach, but as we have not tested our unit tests against your solution yet, we cannot be 100% sure… So, if you suspect a mistake, please do verify with your teacher.



```
✓ ReferenceCalendarTest
    ✓ aSingleWorkingDayShouldBeReportedAsSuch()
    ✓ methodsShouldReportSameNumberOfWorkingDays()
    ✓ methodsShouldReportTheCorrectNumberOfWorkingDays()
    ✓ Test passed ysShouldBeReportedAsSuch()
    ✓ workingDaysOnlyExistIfToDateIsAfterFromData()
✓ ReferenceOutputTest
    ✓ checkHourlyWageLarge()
    ✓ checkHourlyWageSmall()
    ✓ checkJuniorEmployeesLarge()
    ✓ checkJuniorEmployeesSmall()
    ✓ checkLongestProjectLarge()
    ✓ checkLongestProjectSmall()
    ✓ checkMonthlySpendsLarge()
    ✓ checkMonthlySpendsSmall()
    ✓ checkMostInvolvedEmployeeLarge()
    ✓ checkMostInvolvedEmployeeSmall()
    ✓ checkTotalManpowerBudgetLarge()
    ✓ checkTotalManpowerBudgetSmall()
    ✓ checkTotalOvertimeLargeProject()
    ✓ checkTotalOvertimeSmallProject()
```

Grading criteria.

You are expected to use functional interfaces and streams in your solution. In each of the above 10 incomplete methods of PPS, Project and Employee you are expected to use at least one of:

A. .stream()
B. A reference to a method using the '::' notation
C. A lambda expression using '->' notation
D. .collect()
E. .merge()
F. .mapToInt()
G. .filter()

Overall, across your solution, each of the above features A – G should occur at least once.

The code snippets of all seven completed PPS methods should be included in your report, and clarified.

Subject to these grading criteria, the following scale will be applied:

Sufficient: Five out of the seven PPS methods are correctly implemented and explained.

Good: Six out of the seven PPS methods are correctly implemented and explained, leveraging correct implementations of all three methods of Product and Employee.

Excellent: All seven PPS methods are correctly implemented and explained, leveraging correct implementations of all three methods of Product and Employee.

Sample result:

Here you find the expected output of PPS.printPlanningStatistics() for the sample input file 'HVA2018_e10_p25.xml'.

```
Project Statistics of 'HvA2018_e10_p25.xml' in the year 2018
10 employees have been assigned to 25 projects:

1. The average hourly wage of all employees is 48,90.
2. The longest project is 'Funiture replacements - WBH-05(P100685)' with 113
available working days.
3. The follow employees have the broadest assignment in no less than 12
different projects:
[Thomas E. HANSEN(100016)]
4. The total budget of committed project manpower is 671970
5. Below is an overview of total managed budget by junior employees (hourly
wage <= 26):
{Barbara D. PAYNE(100011)=45950}
6. Below is an overview of employees working at least 8 hours per day:
[Emily T. HERRERA(100039)]
7. Below is an overview of cumulative monthly project spends:
{JANUARY=29395, FEBRUARY=54228, MARCH=82158, APRIL=98839, MAY=86291,
JUNE=66420, JULY=87180, AUGUST=80574, SEPTEMBER=43428, OCTOBER=30908,
NOVEMBER=12549}
```

Figure 1: Expected results of sample HVA2018_e10_p25

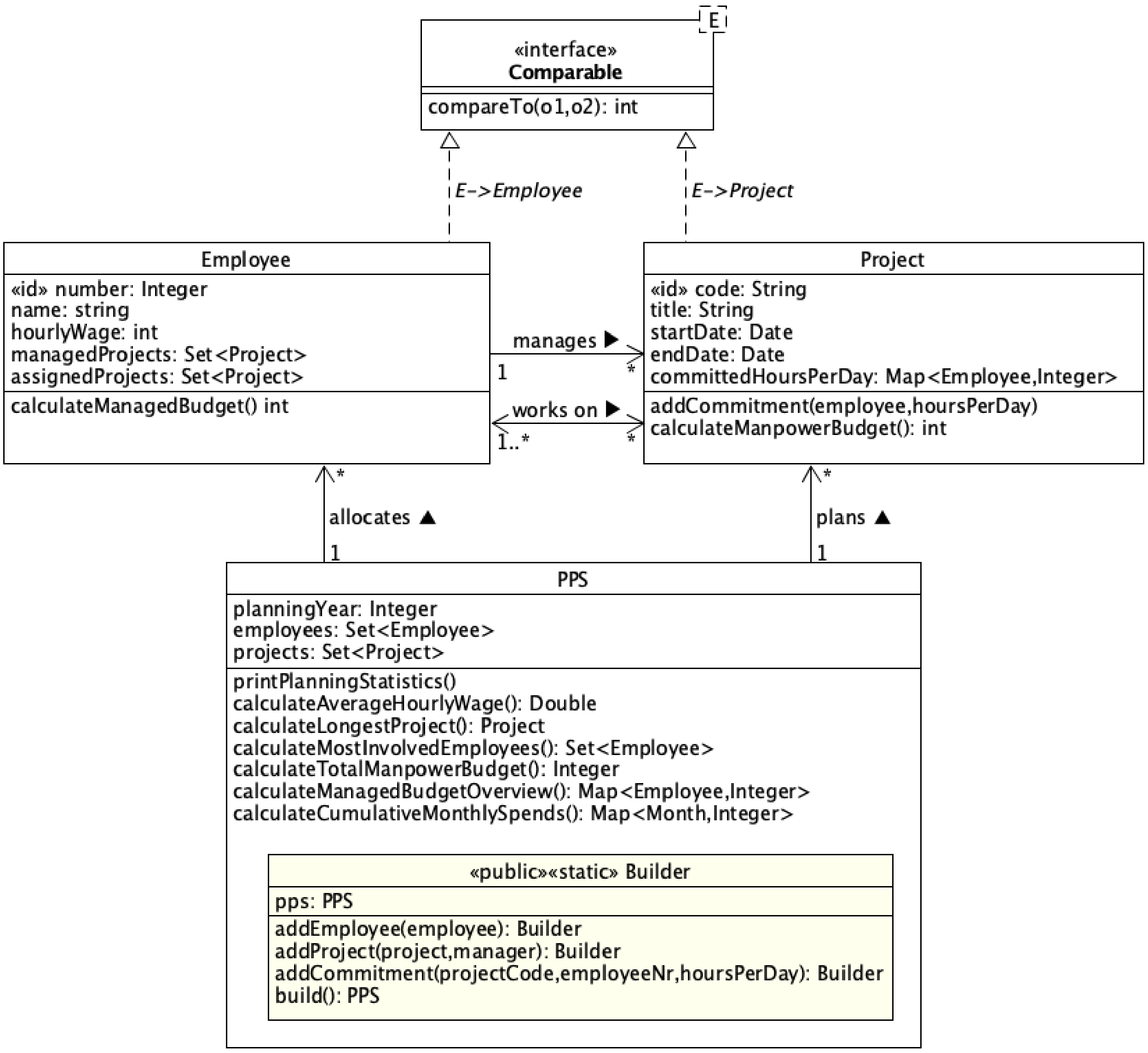

E
«interface»
Comparable
compareTo(o1,o2): int
E->Employee
E->Project
Employee
«id» number: Integer
name: string
hourlyWage: int
managedProjects: Set<Project>
assignedProjects: Set<Project>
calculateManagedBudget() int
Project
«id» code: String
title: String
startDate: Date
endDate: Date
committedHoursPerDay: Map<Employee,Integer>
addCommitment(employee,hoursPerDay)
calculateManpowerBudget(): int
manages ▶
1
*
works on ▶
1..*
*
allocates ▲
*
1
plans ▲
*
1
PPS
planningYear: Integer
employees: Set<Employee>
projects: Set<Project>
printPlanningStatistics()
calculateAverageHourlyWage(): Double
calculateLongestProject(): Project
calculateMostInvolvedEmployees(): Set<Employee>
calculateTotalManpowerBudget(): Integer
calculateManagedBudgetOverview(): Map<Employee,Integer>
calculateCumulativeMonthlySpends(): Map<Month,Integer>
«public»«static» Builder
pps: PPS
addEmployee(employee): Builder
addProject(project,manager): Builder
addCommitment(projectCode,employeeNr,hoursPerDay): Builder
build(): PPS

Figure 2: PPS class diagram