
Relatório Final

Trabalho de Grupo

# Project Manager

Enunciado 02

**Formação**

*"ReProgama a tua careira"*

**Elementos**

Ana Filipa Estiveira

Filipa Gonçalves

Gonçalo Carvalho

José Oliveira

Data de Entrega : 10 de Novembro de 2014

# Índice

# 1. Introdução

No âmbito da avaliação do módulo *M1- Programação e Algoritmia*, da formação Randstad *Reprograma a tua carreira,* foi-nos proposto a elaboração e implementação de um trabalho em grupo. O trabalho intitulado *Project Manager* foi seleccionado de modo aleatório de entre quatro trabalhos propostos pela *ChallengeIT*. Os elementos do grupo foram escolhidos de forma análoga entre os formandos.

O trabalho consiste na realização de um aplicativo, a pedido de um dado cliente, para determinar os custos associados ao desenvolvimento de projectos. Sabe-se que os elementos que definem um projecto são o local de trabalho, a equipa de consultores e subprojectos. O aplicativo terá as funções de adicionar elementos a um projecto, obter custos dos projectos e mostrar na consola todos os elementos de um projecto. As dúvidas são colocadas ao cliente durante o desenvolvimento do aplicativo.

## 1.1. Estrutura Do Relatório

A estrutura do relatório inicia-se com a apresentação do problema e consequentemente a sua interpretação. De seguida aborda a metodologia de trabalho utilizada, bem como o planeamento das tarefas de modo a cumprir os prazos pré-estabelecidos com o cliente. Por fim faz uma abordagem técnica do desenvolvimento do aplicativo e apresenta os resultados obtidos.

# 2. Apresentação do Problema:

O nosso cliente pretende criar uma aplicação que seja capaz de:

- Adicionar diferentes elementos a um projecto (local de trabalho, equipa de consultores e subprojectos);
- Determinar os custos associados ao desenvolvimento de projectos;
- Mostrar os diferentes elementos que compõem o projecto

Segundo o cliente *"um local de trabalho é caracterizado pelo seu nome, localização e preço. Um consultor é definido pelo seu nome, o seu preço por hora e o número de horas de trabalho que realizou. Uma equipa tem um nome e é composta por um conjunto de consultores, onde um deles é o líder de equipa. A diferença entre um consultor e um líder é que o líder ainda recebe um prémio. O custo da equipa é a soma do custo de cada um dos consultores. Um projeto tem um nome e é gerido por um consultor que não está inserido em nenhuma equipa. O custo total de um projeto corresponde à soma do custo de cada um dos seus elementos."*

Um outro requisito exigido é referente a apresentação dos resultados do aplicativo, isto é a apresentação "dos *elementos deve ser feita pela ordem alfabética do nome do elemento, enquanto a apresentação dos consultores da equipa deve ser feita de forma crescente pelo seu custo por hora."*

## 2.1. Esquematização do enunciado

Após a interpretação em grupo dos dados supracitados e esclarecimentos de dúvidas com o cliente, conseguimos inferir o seguinte:

Os custos envolvidos no desenvolvimento de um projeto dependem dos seguintes elementos:

- Local de trabalho;
- Equipa de consultores;
- Subprojectos.

- *Projecto:*
  - Um projeto tem um nome;
  - É gerido por um consultor que não está inserido em nenhuma equipa.
    - Os Managers dos subprojectos não correspondem ao manager do projeto.
    - Qualquer manager tem um valor de prémio associado, quer seja manager de um projeto ou subprojecto.
  - Um projeto contém subprojetos.

        - Um subprojecto é, em si, um projeto.
    - O custo total de um projeto corresponde à soma do custo de cada um dos seus elementos.

- *Equipa de Consultores*
    - Tem um nome;
    - É composta por um conjunto de consultores e um deles é o líder de equipa.
    - O custo da equipa é a soma do custo de cada um dos consultores.
    - Os elementos de um projecto são o conjunto composto pelos elementos principais (desse projecto) e pelos elementos de todos os subprojectos e assim sucessivamente. Podem existir elementos na equipa principal que não estão em nenhum subprojecto, e elementos num subprojecto que não se encontram na equipa principal (i.e. do projecto).

- *Consultor*
    - É definido pelo seu nome, o seu preço por hora e o número de horas de trabalho que realizou.
    - A diferença entre um consultor e um líder é que o líder ainda recebe um prémio. O prémio é um valor fixo por projecto, ou seja, recebe o mesmo prémio independentemente do projecto em causa.
    - O valor do consultor não varia consoante o projeto.
    - Quando um consultor é adicionado a uma equipa é indicado o número de horas que irá trabalhar.
    - Os consultores, e consequentemente os gestores apenas podem estar afetos a um projeto. Não podem participar  em mais do que um projeto em simultâneo.

- *Local de trabalho:*
    - Caracterizado pelo seu nome, localização e preço.
    - Um projeto tem um local de trabalho. Se contiver outros projetos, cada um deles terá naturalmente o seu local de trabalho, igual ou diferente do projeto em que está inserido.**

- *Aplicação deve permitir:*
    - Criar um projeto no qual se pode:
        - Adicionar os diferentes elementos;
        - Obter o custo do projeto;
        - Mostrar os diferentes elementos que compõem o projeto.
            - Elementos do projecto ordenados por ordem alfabética
            - Consultores da equipa ordenados de forma crescente pelo seu custo por hora.

# 3. Método de trabalho

Aquando do início do projecto, decidimos que a leitura linear, interpretação, a recolha e esquematização da informação do enunciado 02 – Project Manager e consequente discussão sobre a forma de implementação mais adequada seria feita por toda a equipa para que todos compreendessem o problema e pudessem contribuir para a identificação dos pontos-chave e esquematização do modelo a ser implementado.

A abordagem à resolução do problema passa por:

- Desenvolvimento de diagrama de domínio UML;
- Implementação do código / teste unitários;
- Aplicativo final;
- Relatório final;
- Apresentação de PowerPoint.

O trabalho foi efectuado praticamente em simultâneo, com contribuições valiosas de cada elemento do grupo, devido às diferentes valências e níveis de conhecimento. Surgiram ao longo de todo o processo dúvidas, algumas delas expostas ao "cliente" e outras devido a questões de procedimento ou abordagem, que foram expostas e resolvidas pelo grupo.

## 3.1 – Distribuição De Trabalho

Ao longo de todo o processo foram distribuídas tarefas distintas a cada elemento do grupo, contudo procurou-se sempre que todos os elementos participassem e estivessem a par de todas as vertentes.

A formanda Filipa Gonçalves teve uma participação importante nas tarefas associadas à parte Office, na implementação do código, na verificação dos testes, nas discussões dos métodos, abordagens para a solução do problema e no desenvolvimento do plano de tarefas em Project Libre

A formanda Filipa Estiveira teve uma participação importante nas tarefas associadas à parte Office, quanto à implementação do código, participou numa fase inicial, não se sentindo preparada para uma fase mais avançada da discussão do mesmo.

O formando José Oliveira teve uma participação importante na implementação do código, nos testes e nas discussões dos métodos e abordagens para a solução do problema e na realização do relatório final.

O formando Gonçalo Carvalho teve uma participação importante na implementação do código, nos testes e nas discussões dos métodos e abordagens para a solução do problema.

## 3.2- Recursos utilizados

Visto os formandos não estarem geograficamente perto uns dos outros, foi decidido que o sistema de trabalho seria em sala de formação, com o horário da formação destinado para este trabalho e o restante seria feito em regime *homeworking* via Skype, com tarefas pré-destinadas a cada um e devidamente distribuídas, podendo sempre partilhar informação on-line.

Sempre que surgiu uma dúvida, e sempre que possível, essa dúvida foi exposta aos colegas do grupo, ao "cliente", aos formadores (que tanto estavam em sala, como no Skype), ou aos colegas dos outros grupos.

Foi utilizado como repositório de versões do nosso código, o GitHub git@github.com:zegreg/ProjectManager.git

## 3.3 – Test driven development (TDD)

O método utilizado para o desenvolvimento do código foi o test ***driven*** development (TDD). Os testes unitários foram implementados de modo a dar resposta aos requisitos essenciais exigidos pelo cliente. Estes testes permitiram-nos orientar para os resultados esperados e optimizar o código principal.

Os testes desenvolvidos foram:

- *gestTotalCost_test()*. Este método verifica um dos requisites exigidos, a obtenção dos custos de um projecto. O custo de um projecto é a soma total dos custos dos seus elementos (local, Consultores, Equipa e Manager).

- *shouldNotAcepetTheSameConsultant_test()*. Este teste pretende garantir que uma equipa de um projecto, não poderá aceitar duplicação de consultores.

- *getTotalCost_withNewProject_test()*. Um projecto pode ter subprojectos que também apresentam custos. Portanto o custo de um projecto é o somatório dos seus custos mais os custos de subprojectos, caso existam. Este teste pretende garantir que a adição de um subprojecto a um projecto resulta, também no aumento dos custos totais do projecto

- *shouldNotAcepetSubProjectsWithSameName_test()*. Um projecto não poderá ter duplicação de subprojectos.

- *shouldRemoveASubProject_test()*. Uma das ferramentas do utilizador será a capacidade de adicionar e remover elementos. Este teste pretende garantir a remoção de subprojectos de um projecto.

- *shouldRemoveAConsultants_test()*. Este teste pretende garantir a remoção de um consultor de um projecto.

- *shoulSortByNameSubProjects_test()*. Este teste pretende garantir a ordenação dos subprojectos por ordem alfabética.

- *shoulSortByConsultsCostPerHourInATeam_test()*. Este teste pretende garantir a ordenação por ordem crescente de custo de mão de obra (custo/hora).

- *findSubProjectByName_test()*. Este teste pretende garantir a pesquisa por nome dos projectos.

## 3.4 -Planeamento de tarefas

A estratégia delineada pelo grupo de modo a cumprir os prazos está representada na figura 1.

Versão final

| | Nome | Duração | Início | Fim |
|---|---|---|---|---|
| 1 | Leitura e interpretação do enunciado | 0,8 dias? | 05-11-2014 10:00 | 06-11-2014 10:00 |
| 2 | Recolha e esquematização da informação do enunciado | 0,6 dias? | 05-11-2014 10:00 | 05-11-2014 17:00 |
| 3 | Discussão sobre a forma de implementação mais adequada | 2,4 dias? | 05-11-2014 10:00 | 10-11-2014 10:00 |
| 4 | Identificação das classes e métodos | 2,1 dias? | 05-11-2014 11:00 | 07-11-2014 17:00 |
| 5 | Identificação dos Pontos de fragilidade na implementação esc | 2 dias? | 05-11-2014 14:00 | 10-11-2014 9:00 |
| 6 | Pedido de esclarecimento ao cliente | 0,1 dias? | 05-11-2014 16:00 | 05-11-2014 17:00 |
| 7 | Elaboração do Modelo UML preliminar | 0,3 dias? | 05-11-2014 14:00 | 05-11-2014 17:00 |
| 8 | Elaboração e envio do relatório preliminar | 0,1 dias? | 05-11-2014 16:00 | 05-11-2014 17:00 |
| 9 | planeamento de tarefas | 0,2 dias? | 06-11-2014 9:30 | 06-11-2014 11:30 |
| 10 | Desenvolvimento e implementação do código | 1,75 dias? | 06-11-2014 9:30 | 10-11-2014 11:00 |
| 11 | Desenvolvimento e implementação de testes associados | 1,75 dias? | 06-11-2014 9:30 | 10-11-2014 11:00 |
| 12 | Pedido de esclarecimento ao cliente | 0,1 dias? | 06-11-2014 16:00 | 06-11-2014 17:00 |
| 13 | Documentação do código | 1,4 dias? | 06-11-2014 14:00 | 10-11-2014 11:00 |
| 14 | Elaboração do Modelo UML final | 0,3 dias? | 08-11-2014 8:00 | 10-11-2014 11:00 |
| 15 | Estruturação do relatório Final | 0,6 dias? | 07-11-2014 16:00 | 10-11-2014 14:00 |
| 16 | Elaboração dos Slides de Apresentação do Trabalho | 1,4 dias? | 06-11-2014 16:00 | 10-11-2014 14:00 |
| 17 | Entrega do Projeto em Java | 0,1 dias? | 10-11-2014 13:30 | 10-11-2014 14:30 |
| 18 | Entrega do relatório Final | 0,1 dias? | 10-11-2014 13:30 | 10-11-2014 14:30 |
| 19 | Preparação da Apresentação oral do trabalho | 0,2 dias? | 10-11-2014 12:00 | 10-11-2014 15:00 |

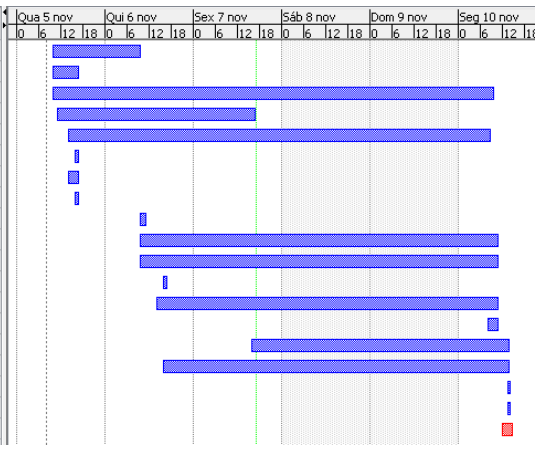


**Figura 1- Cronograma de actividade**

O planeamento sofreu alterações à medida que o trabalho foi desenvolvido e as dúvidas com o cliente esclarecidas. Demorámos mais tempo do que o esperado na identificação da melhor metodologia de implementação para o problema, e na identificação dos seus pontos de vulnerabilidade.

## 3.5 -Modelo de domínio, em UML

Versão final:

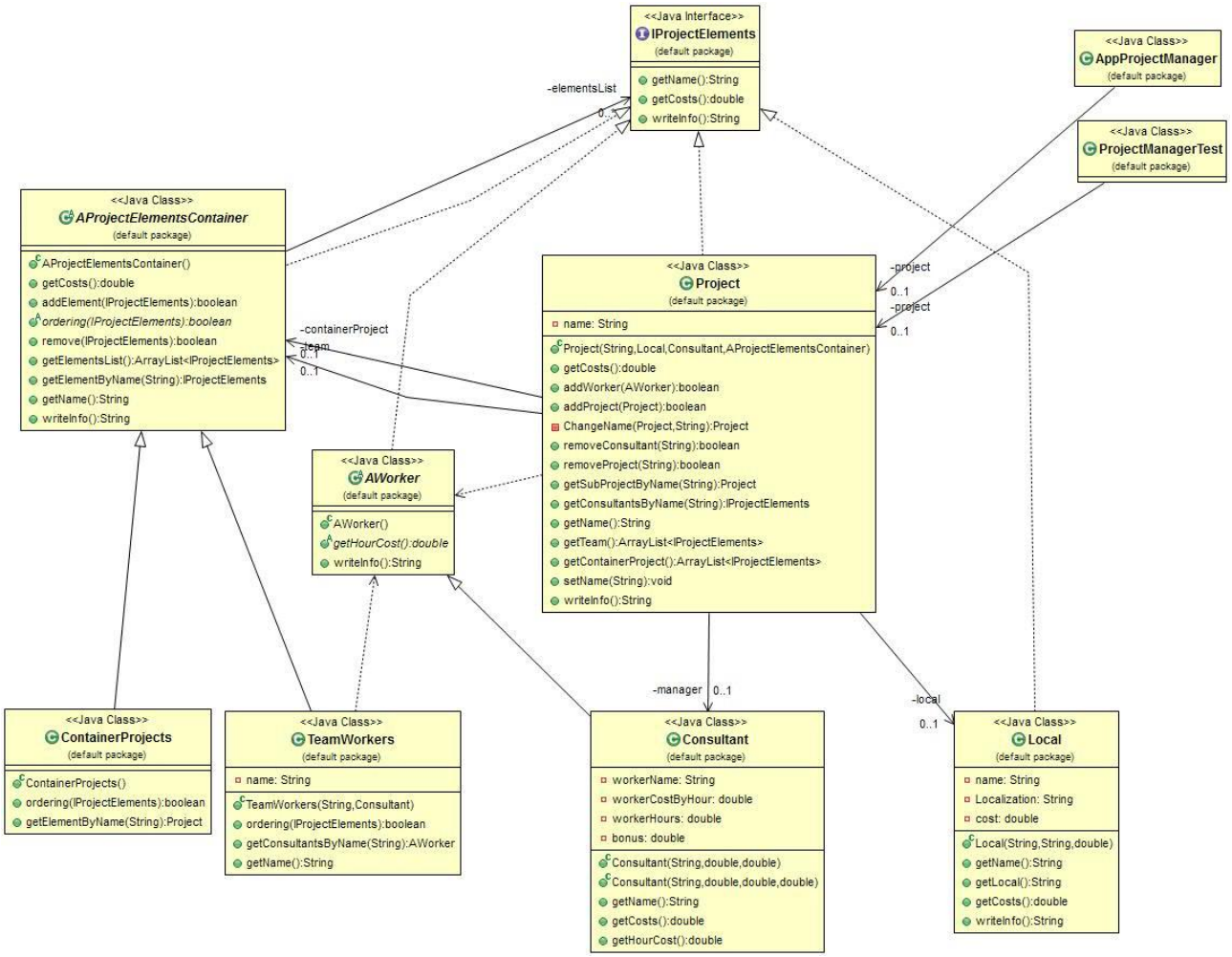


Figura 2- Representação do diagrama de domínio UML.

# 4. Especificação dos componentes / Classes

Com base na informação recolhida do enunciado tivemos necessidade de criar as seguintes classes com as seguintes características:

**Interface IProjectElements**

Todos os elementos de um projecto têm um nome e um custo associado, assim decidimos também criar esta interface, cujos métodos permitem obter o nome (*getName()*), custo (*getCost()*) e escrever a informação *(writeInfo())* relativa a cada uma das instâncias das classes que a implementam.

Esta interface é implementada nas classes Classe abstrata *AProjectElementsContainer*, Classe *Local*, Classe Abstrata *AWorker* e pela *Classe Project*.

**Classe *abstrata AProjectsElementsContainer***

Esta classe abstrata implementa a interface IProjectElements. Todos os seus elementos são arrayLists onde diferentes elementos são armazenados. No enunciado é-nos dito que um projecto é constituído por diferentes elementos sendo que um deles é também um projecto que pode conter outros projectos e assim sucessivamente. Dada a natureza do problema, o primeiro esquema que nos surgiu foi um esquema em árvore o que nos direcionou logo para a possível utilização de um TreeSet ou um TreeMap para armazenar os diferentes projectos e seus elementos. A ideia do TreeSet foi abandonada porque podia haver repetição de consultores, ou seja um consultor podia estar no Project principal e num subprojecto. O TreeMap seria mais adequado, uma vez que utiliza uma chave que poderia facilitar a pesquisa dos diferentes elementos nos diferentes níveis, contudo não dominamos ainda este tipo de contentor logo poderiam advir daí alguns problemas de implementação. Escolhemos utilizar um *ArrayList* por assim podermos obter, remover ou reescrever um elemento que se encontra em determinada posição, a possibilidade de obter o tamanho do *arrayList* e pela necessidade de ordenar os seus elementos de diferente forma conforme a sua natureza/tipo.

Todos os elementos têm nome e custo associado que resulta da soma do custo de todos os elementos que armazena que podem ser obtidos utilizando os métodos *getName()* e *getCost()*, respectivamente. Os seus elementos podem ser ordenados da forma desejada para cada um dos tipos (como pede o enunciado), utilizando o método *Ordering()* e podem ser adicionados (*AddElement()*), removidos (*Remove()*) ou encontrados através do seu nome dentro dos objectos desta classe (*GetElementByName()*).

Foram ainda criados os métodos *getElementsList()* obtém a lista de elementos desejada procurando-a pelo seu nome e que pode ser usada para efeitos de impressão, ordenação ou procura de elementos em particular na lista, e o *WriteInfo()* que cria uma String que resulta da concatenação da informação armazenada em cada elemento do arrayList.

**Classe Abstrata AWorker**

Esta classe abstrata implementa a interface IProjectElements. Todos os seus elementos têm nome e custo associado. Esta classe abstrata foi criada para permitir a criação de outros permitir a criação de outros objectos com características semelhantes mas que não sejam do tipo Consultant, ou seja, se um dia mais tarde for necessário integrar elementos num projecto que não sejam consultores, será apenas necessário criar uma classe derivada desta referente a esses novos objectos.

**Classe Project**

Havendo esta relação de dependência entre o projecto e o *Local de trabalho, a equipa de consultores e subprojectos e, tendo eles características diferentes, inicialmente pensámos em criar estas quatro classes, Project, Local, TeamWorkers e Subproject,* cujas instâncias representam cada um dos elementos do projeto. Contudo desde logo surgiu-nos a dúvida se os subprojectos eram, também eles, projectos de "menor dimensão" e, como tal, com as mesmas características do projecto. Após esclarecimento do cliente, que referiu que um projeto contém projetos, ou seja, um subprojecto é, em si, um projeto, decidiu-se que não haveria uma classe dedicada aos objectos do tipo Subproject, já que estes são objectos do tipo Project.

As instâncias da classe Project representam um projecto. Um projeto tem um nome, é gerido por um consultor (*Manager*) que não está inserido em nenhuma equipa e tem elementos do tipo *Local* e *ConsultantTeam.*

**Classe TeamWorkers**

As equipas de trabalho são instâncias da classe TeamWorkers que têm um nome e são compostas por um conjunto de consultores, sendo um deles o líder de equipa (leader).

Às instâncias desta classe, podem ser adicionados elementos utilizando o método *Ordering().* Uma vez que, segundo o enunciado os elementos da equipa deveriam ser ordenados por ordem crescente de preço por hora, quando queremos inserir um *worker* na equipa o mesmo é logo posicionado na lista ordenada de acordo com o seu vencimento.

**Classe Consultant**

Esta classe estende a classe *AWorker,* logo as suas instâncias são definidas pelo seu nome e o seu preço/ hora.

As suas instâncias representam Consultores que são definidos pelo seu nome, o seu preço/ hora e o número de horas de trabalho que realizam em determinado projecto. Uma vez que são parte integrante da TeamWorker, decidimos criar uma classe, Consultant, cujas instâncias representam

Consultores. Ao longo do enunciado é referida a existência de dois tipos de consultores, o consultor com as características acima apresentadas e o líder/Gestor que além destas características, tem a particularidade de receber um bónus.

**Classe ContainerProjects**

Os projectos são instâncias da classe ContainerProjects que é um contentor de objectos do tipo Project.

Os objectos do tipo Project podem ser adicionados e ordenados por ordem alfabética pelo método *Ordering()*.

**Classe Local**

As instâncias desta classe representam o local de trabalho de cada projecto e são caracterizadas pelo nome, localização e preço do local de trabalho do projecto.

# 5. Aplicativo centrado no utilizador

Para uma utilização mais intuitiva do aplicativo, e pensando nas possíveis necessidades de utilização, criamos a classe AppProjectManager. Esta classe tem diversos comandos que vão definir métodos que irão chamar os métodos da Classe Project.

Os comandos são:

- *"subproject"* - Adicionar subprojectos;
- *"consultant"* - Adicionar consultores;
- *"remove project"* - Remover projecto;
- *"remove consultant"* - Remover consultores;
- *"costs"* - Mostrar os custos do Projecto;
- *"info"* - Mostrar elementos do Projecto;
- *"exit"* – Fim da aplicação

Os método da *AppProjectManage* são:

- *showCommand();*- Mostra na consola todos os comandos que estão disponíveis ao utilizador.
- *readCommand();*- Lê os comandos que são inseridos na consola.

- *readProject();* - Lê da consola a informação necessária para criar um novo objecto do tipo Project. Este método retorna um novo objecto do tipo Project caracterizado pelo seu nome, manager, local de trabalho e equipa.

- *readLeader();*- Lê da consola a informação necessária para criar um novo objecto do tipo *Consultant* com as características de um líder. Este método retorna um novo objecto do tipo *Consultant* caracterizado pelo seu nome, preço por hora, número de horas de trabalho e bónus (porque é o líder/manager).

- *readProjectLocation();*- Lê a informação necessária para criar um novo objecto do tipo Local que representa o local de trabalho associado a cada projecto. Este método retorna um novo objecto do tipo Local caracterizado pelo seu nome, localização e custo.

- *readConsultantTeam();*- Lê a informação necessária para criar um novo objecto do tipo *TeamWorkers* que representa a equipa de trabalho associada a cada projecto. Este método retorna um novo objecto do tipo *TeamWorkers* caracterizado pelo seu nome e líder.

- *readConsultant();*- Lê a informação necessária para criar um novo objecto do tipo *Consultant* que representa a os consultores que podem ser associados às equipas de trabalho ou como Manager de um determinado projecto. Este método retorna um novo objecto do tipo *Consultant* caracterizado pelo seu nome, preço por hora e número de horas de trabalho.

- inputDouble();- Verifica se os valores introduzidos na consola são números, como é esperado.

A leitura dos dados do teclado é feita através da Classe Scanner do package java.util que recebe como parâmetro uma InputStream (System.in). O objecto do tipo Scanner é declarado no construtor do *AppProjectManager*.

É no método execute do *AppProjectManager*, onde se inicia a leitura dos comandos pela chamada do método *readCommand()* que retorna uma String do método *nextLine()* do Scanner. Enquanto receber *input* válidos do teclado, é feita a leitura, ou seja, como os comandos são do tipo *final* e utilizados como *key* num *switch*, se esse *key* não existir a sua leitura é interrompida.

Passemos então à explicação de cada um dos comandos da aplicação:

- *"subproject"* - Este comando permite a criação de um subprojecto de um projecto previamente criado, seja ele o projecto inicial, quer seja de um subprojecto deste. Nele é pedido ao utilizador que insira na consola o nome do projecto onde queremos criar o subprojecto. É feita uma pesquisa para verificar se o nome do projecto inserido existe utilizando o método *getSubProjectByName()* da classe *Project*; caso não exista é retornado um aviso *"The project/subproject does not exists!"*; caso exista, é usado o método *readProject()* para pedir ao utilizador que insira a informação do novo subprojecto. É então feita a verificação se o subprojecto já existe utilizando novamente o método

*getSubProjectByName()* da classe *Project* e, caso exista, é retornado um aviso *"That worker has already been added to this project!"*. Caso não exista, o subprojecto é criado e é perguntado ao utilizador se pretende criar mais algum subprojecto. Este comando é executado até que a resposta seja diferente de "y".

- *"consultant"* - Este comando permite a adição de um consultor à equipa de um projecto previamente criado, seja ele o projecto inicial, quer seja de um subprojecto deste. Nele é pedido ao utilizador que insira na consola o nome do projecto onde queremos inserir o consultor. É feita uma pesquisa para verificar se o nome do projecto inserido existe utilizando o método *getSubProjectByName()* da classe *Project*; caso não exista é retornado um aviso *"The_project/subproject_does_not_exists!"*; caso exista, é usado o método *readConsultant()* para pedir ao utilizador que insira a informação do novo consultor. É então feita a verificação se o consultor já existe nessa equipa de trabalho ou se se trata do Manager do projecto utilizando o método *addWorker()* da classe *Project* que adiciona o consultor, apenas se nenhuma destas situações ocorrer. Caso este consultor já pertença à equipa, é retornado um aviso *"That_worker_has_already_been_added_to_this_project!"*. Caso ainda não exista, o consultor é adicionado ao projecto e é perguntado ao utilizador se pretende adicionar mais algum consultor. Este comando é executado até que a resposta seja diferente de "y".

- *"remove project"* - Este comando permite que seja removido algum subprojecto de um projecto já existente. Para isso é pedido ao utilizador que insira na consola o nome do projecto de onde queremos remover determinado subprojecto e o nome do subprojecto a remover. É feita uma pesquisa para verificar se o projecto inserido existe utilizando o método *getSubProjectByName()* da classe *Project*. Caso o projecto não exista é lançada uma exceção e é mostrado um aviso *"The_project/subproject_does_not_exists!"*; caso exista, é verificado se o subprojecto em causa pertence/existe neste projecto e, em caso afirmativo, remove-o da lista.

- *"remove consultant"* - Este comando permite remover um consultor de uma equipa de trabalho de um projecto previamente criado, seja ele o projecto inicial, quer seja de um subprojecto deste. Nele é pedido ao utilizador que insira na consola o nome do projecto de onde queremos remover o consultor e o nome do consultor em causa. É feita uma pesquisa para verificar se o nome do projecto inserido existe utilizando o método *getSubProjectByName()* da classe *Project*; caso não exista é lançada uma exceção e é mostrado um aviso *"The_project/subproject_does_not_exists!"*; caso exista, é verificado se o consultor em causa pertence à equipa desse projecto e, caso afirmativo, remove-o da lista.

- *"info"* - Este comando permite a escrita num ficheiro *.txt de toda a informação de um projecto/subprojecto, ou seja a informação referente ao Manager, local de trabalho, equipa e todos os seus subprojectos. O nome do ficheiro é o nome do projecto/subprojecto em causa. Se não encontrar o projecto lança exceção.

- *"costs"* - Este comando permite ver os custos associados a um determinado projecto. Nele é pedido que o utilizador que insira na consola o nome do projecto em causa. É feita uma pesquisa para verificar se o nome do projecto inserido existe utilizando o método *getSubProjectByName()* da classe *Project* e caso exista imprime a informação dos custos referente aos elementos desse projecto através do metido getCost() de cada um deles. Se não encontrar o projecto lança exceção.

- *"end"*- Este comando força a saída da aplicação.

# 6. Resultados obtidos

1. <u>O aplicativo inicia com a criação de um Projecto e todos os seus elementos.</u>

New Project
- Project Name: *rePrograma*
- Project Manager
Name: *Filipa G.*
Payment per hour: *20*
Working hours: *300*
Bonus: *200*
- Project Location
Workplace name: *Cave*
Workplace location: *Saldanha*
Workplace cost: *500*
- Project Team
Team Name: Os farruscos
- Team Leader
Name: Filipe
Payment per hour: 20
Working hours: 300
Bonus: 200

2. <u>Posteriormente é mostrado os comandos na consola</u>

Type a Command (mostra os comandos do utilizador):

- Subproject (Criar um subproject)
- Consultant (Criar um consultor)
- remove project( Remover um projecto)
- remove consultant (Remover um consultor)
- costs (Mostrar  na consola os custos totais de um projecto)
- info (Imprimir para .txt a informação correspondente a um projecto)
- exit (Terminar o aplicativo)

3. Utilização dos comandos para alteração do projecto criado

**Info** (mostrar os elementos inseridos no Projecto rePrograma)

Write all information of the project/subproject with the name: rePrograma

**Output:**
rePrograma, cost: 12900.0€
Local: Cave, Saldanha, cost: 500.0€
Manager: Filipa G., payment per hour:20.0€, cost: 6200.0€
Team: Filipe, payment per hour:20.0€, cost: 6200.0€

Nota, o custo total do projecto rePrograma = ao custo do Local + custo do Manager + custo da Equipa.

4. Adição de subprojectos ao projecto e ordenação por ordem alfabética dos subprojectos e ordem crescente de mão de obra (custo/hora)

**rePrograma**, cost: 51900.0€
Local: Cave, Saldanha, cost: 500.0€
Manager: Filipa G., payment per hour:20.0, cost: 6200.0€
Team: Filipe, payment per hour:20.0, cost: 6200.0€

**rePrograma - a tua carreira**, cost: 26050.0€
Local: sala ao lado, Saldanha, cost: 300.0€
Manager: José, payment per hour:20.0, cost: 8150.0€
Team: Lara, payment per hour:**10.0**, cost: 5000.0€
Hugo, payment per hour:**15.0**, cost: 4600.0€
Daniel, payment per hour:**20.0**, cost: 8000.0€

**rePrograma - o teu dia**, cost: 12950.0€
Local: a sala atrás do vidro, Saldanha, cost: 200.0€
Manager: Filipa E., payment per hour:20.0, cost: 8150.0€
Team: Eva, payment per hour:15.0, cost: 4600.0€
Gonçalo Carvalho

Nota, todos os subprojectos adquirem como prefix o nome do projecto.
O custo total do reprograma = ao custo do subprojecto rePrograma – a tua carreira (26050.0€) + o subproject reprograma- o teu dia (12950.0€) = 51900.0€

# Discussão e Conclusão

O desenvolvimento deste aplicativo foi bastante educativo, tanto na consolidação dos conhecimentos adquiridos neste 1º módulo de programação, como na partilha e recolha de informação por parte do grupo. Foi interessante perceber a dinâmica de trabalho de grupo, e a riqueza conseguida pela conjugação de ideias à realização do problema em causa. O enunciado fornecido, aparentemente de fácil compreensão, não apresenta uma solução objectiva, e suscitou algumas dúvidas, descobertas durante as nossas discussões em grupo. O planeamento inicial foi em grande parte cumprido, com exceção do tempo necessário para a identificação dos pontos de fragilidade na implementação escolhida. Tínhamos presente que *"…o pior estado de um mau programa é funcionar"* e por isso a nossa concentração estava na concepção e não nos resultados.

As dúvidas foram surgindo à medida que avançávamos no desenho esquemático do projecto. Sabíamos que um bom desenho inicial, era crucial para um bom resultado junto ao cliente. Logo de início deparamo-nos com o elemento subprojecto, ficámos na dúvida se os subprojectos eram vistos como projetos de menor dimensão ou seriam vistos apenas como "tarefas" que a equipa tem de desempenhar e apenas terão um determinado custo associado. Foi-nos dito que um subprojecto é um projecto, logo implementa todos os elementos de um projecto. Um dos pontos fulcrais do desenho foi quando o cliente nos informou que um subprojecto pode ter subprojectos. Outras dúvidas surgiram:

- Um mesmo projeto pode estar a ser desenvolvido em vários locais de trabalho?
- O valor hora de cada consultor é atribuído consoante o projeto/subprojeto?
- A equipa do projecto corresponde ao total de elementos constituintes de cada equipa dos subprojectos, ou pode haver elementos da equipa que não participam em qualquer subprojectos?
- Os Managers podem ter a seu cargo um e um só projeto? Ou podem ser gestores de projectos e dos seus subprojectos?
- Os Managers dos subprojectos correspondem ao manager do projecto?
- O Manager recebe algum prémio como o líder da equipa?
- O prémio de líder é valor/h, é valor fixo por projecto ou percentagem sobre o preço por hora?
- Os consultores de uma equipa trabalham todos as mesmas horas?

Os aspectos a melhorar e/ou validar passaria pela utilização de um Map em vez de uma List, uma vez que a utilização de uma key poderia facilitar a pesquisa dos diferentes elementos. Os nomes utilizados para as variáveis podiam ser mais expressivos, apesar da documentação produzida em @Javadoc e aos comentários explicativos. Também pensámos na elaboração de um manual de utilização para o cliente e um user interface, que facilitaria a compreensão e a correcta utilização do aplicativo. No

aplicativo ficou por desenvolver o comando *Edit* que permitiria a edição dos elementos de um projecto, fazendo as alterações do estado do mesmo.

O nosso projecto pretendeu ter uma visão generalista, de modo que pudessem ser feitas possíveis extensões ao modelo. Com esta implementação podemos fazer inserção de outros trabalhadores que não sejam consultores nas equipas, criando uma classe derivada de *AWorker*.

BIBLIOGRAFIA


- The Java Programming Language, 4th Edition, by Ken Arnold, James Gosling, David Holmes.
- Slides do 1º Modulo sobre PROGRAMAÇÃO E ALGORITMIA da ChallengeIT.
-Consulta das API no site https://docs.oracle.com/javase/7/docs/api/.