ADVENTURE_2

Setup

Para realizar a segunda parte do projeto deve obter a nova versão do código disponibilizada em *https://github.com/tecnico-softeng/reference.git*. Deve portanto adicionar um novo repositório remoto:

```
$ git remote add reference https://github.com/tecnico-softeng/r
```

De seguida deve assegurar-se que o código da primeira parte foi entregue através do ramo first-deliver, mudando para esse ramo:

```
$ git checkout first-deliver
```

Pode então apagar a versão atual que possui do ramo master:

```
$ git branch -d master
```

e obter a nova versão do master a partir do repositório remoto reference:

```
$ git fetch reference
$ git checkout -b master reference/master
```

Para que o ramo em *origin* do master fique igual ao novo deve sobrepor a nova versão:

```
$ git push origin master --force
```

Deve de seguida criar o ramo develop onde o grupo irá trabalhar na segunda parte e criar esse ramo em origin:

```
$ git checkout -b develop
$ git push origin develop:develop
```

Verifique que consegue correr os testes existentes:

```
$ mvn clean test
```

Enunciado

A segunda parte versa sobre uma implementação mais elaborada do método *process* da classe *Adventure*, tirando partido do padrão de desenho *State*. Esta segunda parte promove não só o trabalho separado entre módulos, utilizando a tecnologia *JMockit*, mas também promove um desenvolvimento em que se testa primeiro (filosofia também adoptada no desenvolvimento dos novos módulos na primeira parte do projeto). Assim, deve começar por ler as classes *Adventure* e *AventureState*, tal como todas as subclasses desta última classe (em *pt.ulisboa.tecnico.softeng.broker.domain*) do código que acabou de configurar: elas servirão de ponto de partida a uma parte significativa das alterações a realizar nesta segunda parte. Deverá ainda estudar cuidadosamente o módulo *Activity*, e utiliza-lo como referência para implementar os outros módulos, dado que este já implementa alguma da funcionalidade que deverão implementar nos módulos *Car* e *Hotel*.

Em relação à primeira entrega, a sequência de eventos que o módulo *Broker* irá iniciar terá de ser alterado para:

- O módulo *Broker* solicita ao módulo *Activity* a reserva duma atividade;
- Se o passo anterior for executado com sucesso, e se a reserva for para vários dias, é solicitado a reserva de um quarto ao módulo *Hotel*;
- De seguida prossegue-se à reserva de um veículo no módulo *Car*. Nota: A reserva de um veículo é opcional e deve ocorrer após a reserva de quarto ou atividade (no caso de não haver lugar a reserva de quarto).
- Por fim, o módulo procede ao pagamento no módulo *Bank* e comunica o valor ao módulo *Tax*.

Seguindo a mesma filosofia dos módulos fornecidos, os módulos *Car* e *Tax* devem, em caso de erro não recuperável, ser capazes de lançar exceções (*CarException* e *TaxException*, respectivamente). Estas situações podem ocorrer por diversas motivos, desde não haver um veículo disponível para as datas solicitadas, até um dos envolvidos na transação comercial não ter o NIF registado no *IRS*. No módulo *Car* essas exceções têm impacto no comportamento do módulo *Broker* de uma forma semelhante ao que acontece com os módulos *Hotel* e *Activity*. Todos os módulos, com exceção do *Tax*, passam a fazer pedidos de pagamento ao banco. No que diz respeito aos módulos *Hotel*, *Activity* e *Car* os pedidos ao *Bank* e *Tax* são repetidos até terem sucesso, qualquer seja o tipo de erro. Os pedidos são armazenados e voltam a ser tentados mais tarde, como já se encontra implementado no código do módulo *Activity*. Se quando o módulo *Broker* fizer a confirmação, estado *confirmed*, não obter de algum destes módulos as confirmações de pagamento e de envio de fatura, passa para o estado *undo*.

No caso de haver erros de comunicação nos pedidos do *Broker* ao *Car* deve ser tentado **5 vezes** até que se considere o processamento da reserva do carro não possível.

No eventual cancelamento de uma transação comercial, cada módulo deve comunicar com o módulo *Tax* solicitando o cancelamento da fatura. Note que dado que a fatura já foi emitida, e que estas nunca são apagadas, os pedidos de cancelamento de fatura nunca lançam uma exceção *TaxException*. Tal como com as submissões de fatura, se houver problemas de comunicação, as anulações são armazenadas e reenviada da próxima vez que houver uma submissão de fatura ou cancelamento.

O código deve ainda ser refatorizado para que:

- Todos os produtos transacionáveis possuam um valor de custo associado. O broker deverá ainda ter uma margem de lucro. O somatório dos valores de cada um dos produtos no pacote de aventura, mais a margem de lucro do broker, é o valor que o cliente final terá de pagar.
- Todas as entidades que podem efetuar transações comerciais possuem um IBAN, pelo que todos os módulos, exceto o Tax, devem comunicar com o Bank para efetuarem transações.
- A entidade cliente é representada como uma classe no módulo Broker. Esta possui um IBAN, NIF e a idade, e passa a ser o argumento do construtor de *Adventure*.
- A entidade Broker possui dois NIFs, um de comprador e outro de vendedor e eles são únicos entre todos os brokers.

Os grupos devem começar por reunir para identificar as tarefas, associá-las ao backlog, e definir dependências de precedência entre elas, de forma a calendarizá-las no período de 3 semanas disponíveis. Por exemplo, durante a primeira semana é necessário que esteja terminada a refatorização dos módulos. Nesta entrega, os grupos devem apenas **garantir uma cobertura de 70% em todos os módulos**.

Cada grupo deve dividir-se em sub-grupos de 3/4 alunos, em que cada um deles trabalha nos módulos que não trabalhou na primeira entrega. **É obrigatório indicar no README.md quais os elementos que trabalham em cada um dos módulos, indicando o número de aluno, o nome e o username no GitHub**.

Para a entrega deverão fazer após o último commit:

```
$ git checkout develop
$ git checkout -b second-deliver
$ git tag ADVENTURE_2
$ git push origin --tags second-deliver:second-deliver
```

Esta *tag* colocada deve ter uma data anterior à data limite de entrega, dia **8 de Abril pelas 20:00**.

Durante os laboratórios, os alunos dos grupos serão avaliados com base no trabalho desenvolvido durante cada uma das semanas. Para isso é necessária a criação de um *sprint* no GitHub, usando a interface de *Project*, em que cada tarefa deve ser representada por um *Note*. Neste caso o projeto a criar deve-se chamar *Sprint Two*. Uma vez criado o projeto, escolhendo como template pré-definido "*Kanban (automatic)*", deve ser alterado o nome da coluna *To Do* para *Backlog*. Nesta coluna serão colocadas todas as tarefas a realizar, as quais serão definidas durante a reunião do grupo. Cada tarefa é criada como uma *Note*, contendo a descrição da tarefa a realizar, e deve ser convertida num *Issue*, por forma a que os commits possam ser-lhe associados. Cada aluno que vá realizar uma tarefa move o respetivo *Note* da coluna de *Backlog* para a coluna *In progress*, e atribui-se essa tarefa no *Issue* respetivo. Quando a tarefa é terminada, o commit deve ser associado ao respetivo Issue, passando este para o estado fechado. Para se associar um *commit* a um *Issue* deve colocar-se na mensagem de commit *close #n*, em que *n* é o número do *Issue*. Quando o Issue é fechado o respetivo *Note* é automaticamente passado da coluna *In progress* para a coluna *Done*. **Cada commit deve estar associado a uma única tarefa**.

Anexo

Para vossa referência, de seguida listamos os issues que consideramos essenciais para o desenvolvimento do projeto:

Broker

- Create the entity Client in broker (replace IBAN)
- Associate two NIFs to broker (seller and buyer)
- Add car interface
- Create state rent vehicle
- Move state payment to before confirmation
- Add tax interface
- Submit invoice to tax module in state payment
- Confirmation state moves to undo if some of the modules did not have payment references or did not sent invoice to tax
- Submit cancel invoice to tax in state undo

Hotel & Car & Tax

- Add NIF and IBAN **[onde aplicável]**
- Add price to a room/vehicle **[onde aplicável]**
- Add bank interface
- Add tax interface
- Submit invoice to tax module
- Processes payment in bank **[onde aplicável]**
- Submit cancel invoice to Tax
- Submit cancel payment to bank
- Manage queues of pending invoices
- Handle errors in cancel invoice to tax
- Handle errors in cancel payment to bank