Engenharia de Aplicações

Licenciatura em Engenharia Informática

GIT e Boas práticas de desenvolvimento de software

Objetivos das aulas PLs da semana 1

- Aplicar e analisar os conceitos associados ao uso do sistema distribuído de controlo de versões GIT
- Aplicar boas práticas de desenvolvimento de software

OBSERVAÇÕES GERAIS

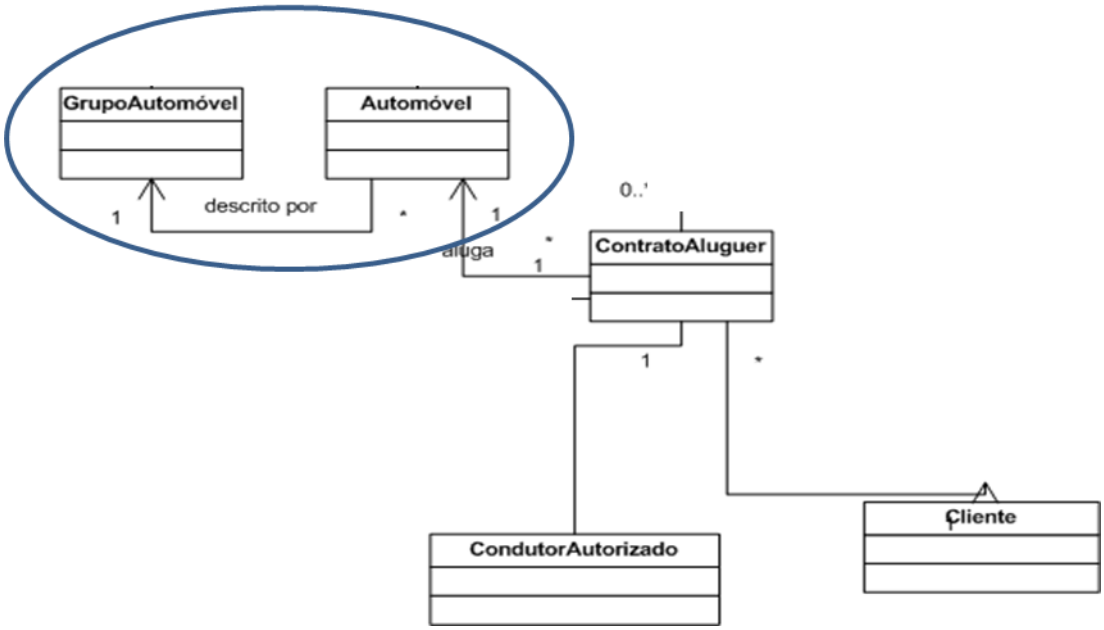
1.Para o desenvolvimento da aplicação DEMO_ORM a realizar nas duas primeiras semanas os ***estudantes vão trabalhar em grupos de 2, aqui designados por Aluno A e Aluno B****, que devem:*

- *Ter a sua conta institucional (@isep.ipp.pt) no bitbucket para criar um repositório para o projeto que vão desenvolver*
- *Na sua máquina local configurar adequadamente o Git para que os commits fiquem identificados adequadamente*
- *Consultar frequentemente, na página de EAPLI no moodle, a documentação disponível.*

Contextualização do domínio da aplicação DEMO_ORM

A empresa "Rent a Car" pretende uma aplicação a ser usada no seu negócio de aluguer de automóveis. Cada automóvel tem informação de matrícula, ano de fabrico, ano de aquisição pela empresa, cor, cilindrada, e grupo de automóvel a que pertence. Os grupos de automóveis descrevem automóveis independentemente da marca com a seguinte informação nome do grupo, nº de portas, preço por dia e classe (que pode ser: utilitário, de luxo ou comercial).
Um cliente pode alugar um automóvel realizando um contrato de aluguer que é, para um determinado período de tempo, determinado grupo de automóvel e automóvel. No contrato também deve ser indicado quais os condutores autorizados. Os dados dos clientes incluem o nome, endereço, telefone e email, e observações.
Os dados dos condutores registados no contrato incluem nome, endereço e detalhes sobre a sua carta de condução (número e data de validade).
O valor do aluguer é calculado aquando da elaboração do contrato.

Considere o seguinte modelo de domínio da aplicação de “Rent a Car”.



O projeto a desenvolver, DEMO-ORM, lidará exclusivamente com as entidades Grupo Automóvel e Automóvel e visará a implementação de uma aplicação com persistência de dados numa base de dados H2 usando ORM, disponibilizando as seguintes funcionalidades:

- **Registar Grupo Automóvel**
- **Listar Grupos Automóveis**
- **Pesquisar Grupo Automóvel por nome, classe,…**
- **Registar Automóvel**
- **Listar Automóveis**
- **Pesquisar Automóvel por matrícula, grupo automóvel**

GIT e Boas práticas de desenvolvimento de software

Objetivos Específicos:

- Aplicar e analisar os conceitos associados ao uso do sistema distribuído de controlo de versões Git no desenvolvimento do projeto DEMO_ORM (Aplicação Java usando Maven)

Passos a executar

(1) **(Aluno A)** Aceder ao Bitbucket com as suas credenciais e criar um repositório Git, de nome ***DEMO_ORM_2019_TTT_XXX_YYY*** *(TTT deverá ser a turma, XXX e YYY deverão ser os nº dos alunos, ex:, DEMO_ORM_2019_2DA_1167854_1143257)*, privado com as seguintes configurações avançadas:

   a. Project Management : issues e wiki, language : Java

(2) "Put some bits in your bucket". Aceitar as opções:

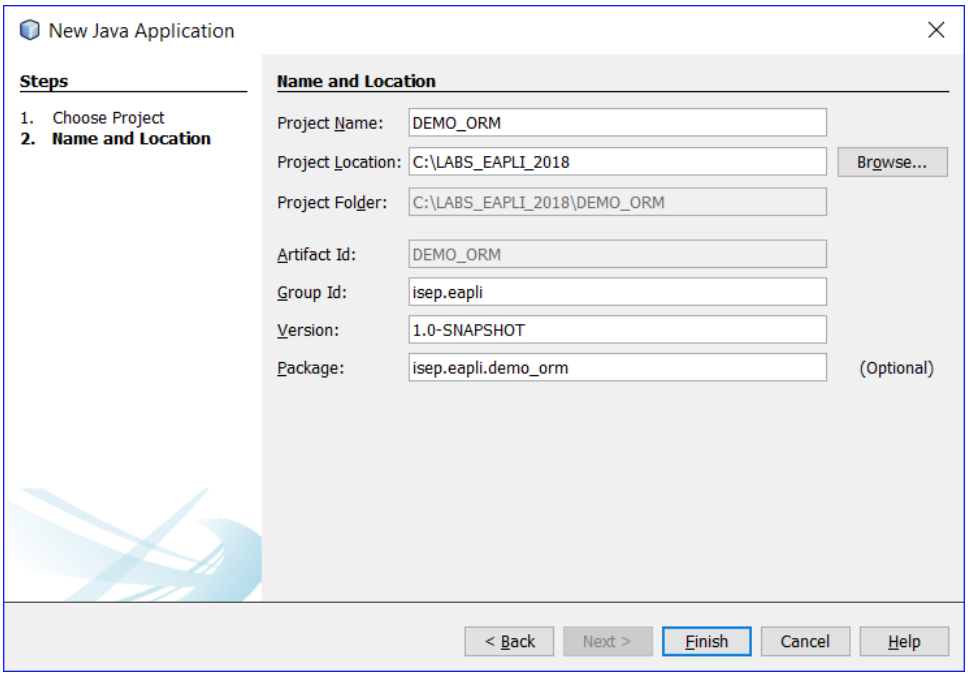
   a. Criar um ficheiro .gitignore (para posteriormente ser preenchido) fazer o Commit

   b. Criar o ficheiro README.md e fazer o Commit

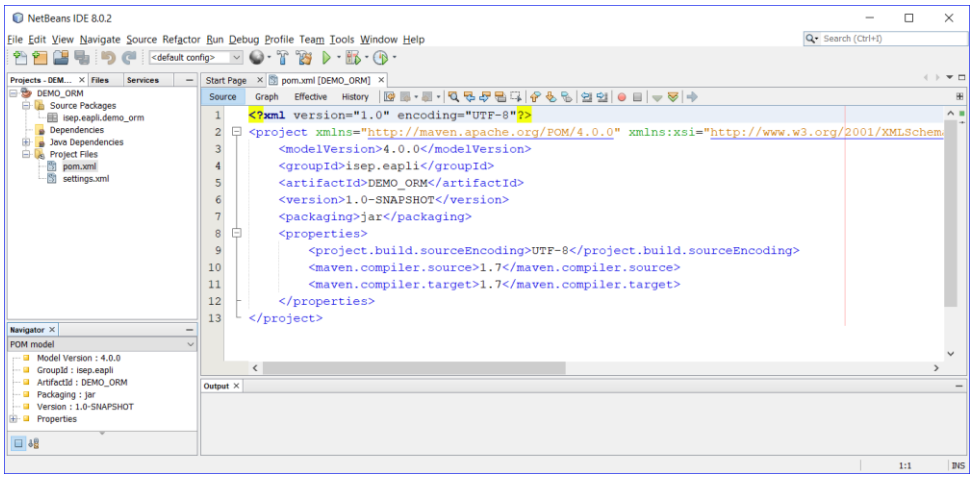
(3) Adicionar a esse repositório o outro colega e o professor das PLs como colaboradores com permissão de escrita WRITE

(4) **(Aluno B)** Na sua máquina local, usando o IDE NetBeans faz o clone do repositório usando o URL do repositório gerado no bitbucket. Deve aceitar a opção do Netbeans para criar um novo projeto que deve ser uma aplicação Java usando Maven

   a. New Project > Categories: Maven > Java Application Next



5. Consultar
   a. https://pt.wikipedia.org/wiki/Apache_Maven
   b. https://platform.netbeans.org/tutorials/nbm-maven-quickstart.html
6. Analisar o ficheiro pom.xml



7. No package isep.eapli.demo_orm criar a classe DemoORM que conterá o método main( )

8. Criar o ficheiro .gitignore com o seguinte conteúdo

```
# Maven
target/
# NetBeans specific
nbproject/private/
build/
nbbuild/
dist/
nbdist/
nbactions.xml
nb-configuration.xml
```

9. Compilar o projeto
   Build with Dependencies

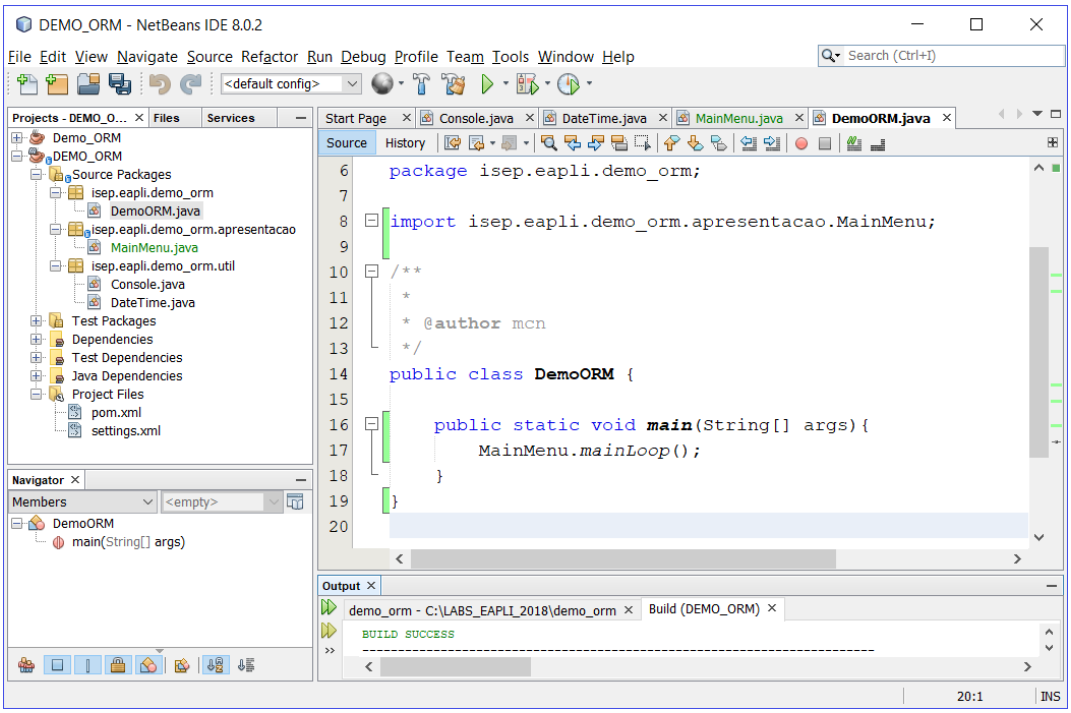
10. Fazer o commit do projeto com a mensagem “Criado o projeto maven com o esqueleto da classe principal DemoORM. Incluído o ficheiro .gitignore”. Sincronizar com o repositório remoto.

11. Os alunos conjuntamente analisam o estado do projeto

12. **(Aluno A)** Fazer o clone do repositório e criar um package isep.eapli.demo_ORM.util e nele coloque as classes utilitárias Console e DateTime incluídas no ficheiro util.zip. Fazer o commit e sincronizar com o repositório remoto.

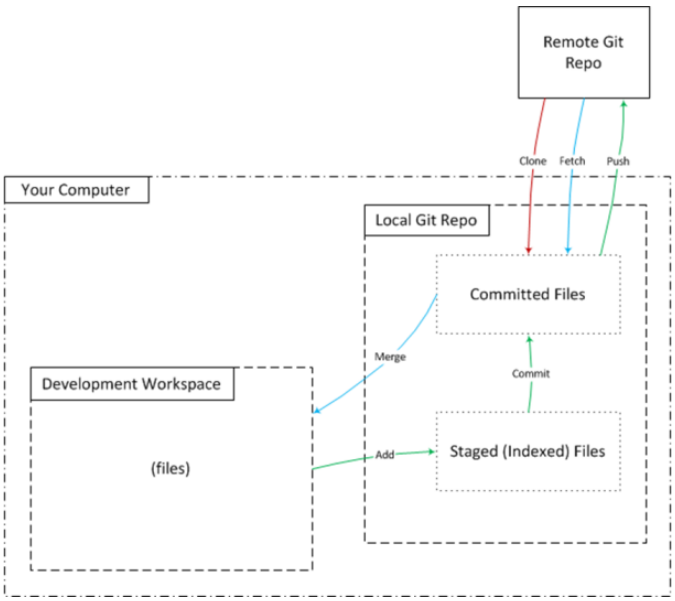
13. **(Aluno B)** Analisar a classe MainMenu incluída no file MainMenu.zip para implementar uma classe com o mesmo conteúdo neste projeto no package isep.eapli.demo_ORM.apresentacao. Se não houver qualquer erro fazer o commit e sincronizar com o repositório remoto.
    ATENÇÃO: Antes de iniciar qualquer alteração no código deve sincronizar o repositório local com o remoto (i.e. fazer pull).

14. **(Aluno A)** Implementar o método main(), fazer o commit e sincronizar com o repositório remoto.



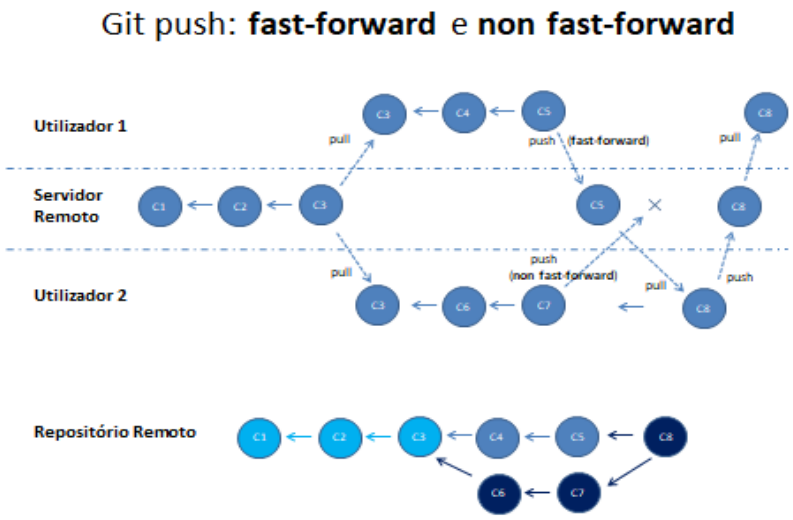
EM EQUIPA RESPONDAM ÀS SEGUINTES QUESTÕES:

1. O projeto DEMO_ORM é uma aplicação Java que usa Maven. O que é o Maven?
2. Qual o objetivo do pom.xml?
3. O que entende por um DVCS? Quais as vantagens da sua utilização?
4. Analise e sintetize os principais conceitos associados ao DVCS Git:



   a. Repositório Remoto, Área de trabalho, Repositório Local
   b. Staged (Indexed) Files e Committed Files
   c. Clone, Push, Pull = Fetch + Merge

5. Qual o objetivo do ficheiro .gitignore?
6. Em que consiste o clone de um repositório?
7. Em que consiste o fork de um repositório?
8. Analise a figura abaixo e a descrição da respetiva sequência de ações. Explique sinteticamente a ocorrência das situações fast_forward e non_fast_forward?



Sequência de ações:

1. Utilizador 1 faz pull do repositório remoto – obtém os commits até ao 3.
2. Utilizador 2 faz pull do repositório remoto – obtém os commits até ao 3.
3. Utilizador 1 faz alterações e depois um commit – cria commit 4.
4. Utilizador 1 faz alterações e depois um commit – cria commit 5.
5. Utilizador 2 faz alterações e depois um commit – cria commit 6.
6. Utilizador 2 faz alterações e depois um commit – cria commit 7.
7. Utilizador 1 faz push e tem sucesso – é um fast-forward.
8. Utilizador 2 faz push mas a operação é recusada – é um non fast-forward.
9. Utilizador 2 faz pull do repositório remoto (fetch + merge) – git cria automaticamente o novo commit 8.
10. Utilizador 2 faz push e tem sucesso – é um fast-forward.
11. Utilizador 1 faz pull do repositório remoto – obtém os commits até ao 8.

Repositório Remoto:

Azul claro – estado inicial
Azul – após passo 7: push do Utilizador 1
Azul escuro – após passo 10: push do Utilizador 2

Utilização do sistema GIT (continuação)

Objetivos Específicos

- Desenvolver software de modo colaborativo – utilização do Git
- Gerir conflitos de alterações com código partilhado
- Definir as tarefas criando "*issues*"

Boas práticas na utilização de sistemas de controlo de versões

- Pull changes/Update before editing
- Commit often
- One commit – one issue
- Write meaningful commit messages
- Don't commit broken code
- Review the merge before commit
- Setup change notifications
- Read Diffs from other developers
- Don't use dropbox or other file sharing
- Don't delete files thru filesystem

Aspetos a ter em consideração na resolução das seguintes tarefas

- Em cada passo a executar, os alunos deverão compreender os fluxos e atualizações de dados nos repositórios remotos e locais bem como nas diretorias de trabalho (de acordo com o diagrama anteriormente apresentado).
- A ordem de execução dos *issues* é determinante para testar os diferentes cenários.
- Os conflitos não são um problema, são apenas uma consequência do trabalho colaborativo por isso não devem ser evitados mas sim resolvidos quando necessário.

Passos a executar

Os Alunos A e B vão iniciar simultaneamente os issues #1 e #2

1. Fast Forward

a) **Aluno A:** #1 – Criar classe GrupoAutomovel no package de domínio e associar respetiva classe de testes; fazer o commit + push

```
public class GrupoAutomovel{
}
```

Texto commit: Classe GrupoAutomovel criada.

> *Tudo funcionará normalmente pois o repositório remoto pode ser atualizado a partir do repositório local – fast forward já que a história do repositório remoto pode ser avançada com a história do repositório local*

2. Non Fast-Forward

b) **Aluno B:** #2 – Criar classe Automovel no package de domínio e associar respetiva classe de testes; fazer commit + pull + push

```
public class Automovel{
}
```

Texto commit: Classe Automovel criada.

> *Ao tentar fazer push será avisado que o repositório remoto contém alterações que não existem no repositório local.*
> *Deve fazer pull para trazer para o repositório local essas alterações*
> *Como as alterações são em partes **distintas** do código o GIT fará o merge automático*
> *Ao fazer push novamente terá sucesso*

3. Merge/Review

a) **Aluno A:** #3 – *Implementar o método para alterar o atributo número de portas da classe GrupoAutomovel*; fazer o commit + pull + push

Texto commit: Criação do método ... da Classe GrupoAutomovel .

b) **Aluno B:** #4 – *Implementar o método para alterar o atributo classe de GrupoAutomovel; fazer o commit*

Texto commit: Implementação do método … da classe Automovel**.**

Aluno B: *#5 – Implementar o método toString()*; commit +pull + push

Texto commit: *Implementação do método toString da classe GrupoAutomovel***.**

> *Como as alterações são em partes **comuns** do código o GIT não fará o merge automático, sendo necessário resolver "merge" ;finalmente necessitará de realizar commit do merge (manual) realizado.*

a) **Aluno B:** #6 – *Implementar o método para alterar o número de kms da classe Automovel*; fazer o commit + pull + push

Texto commit: Criação do método ... da Classe Automovel .

b) **Aluno A:** #7 – *Implementar o método matricula que retorna o valor do atributo matrícula da classe Automovel; fazer o commit*

Texto commit: Implementação do método matricula da classe Automovel**.**

***Aluno A:** #8 – Implementar o método toString()*; commit +pull + push

Texto commit: *Implementação do método toString da classe Automovel*.

> *Como as alterações são em partes **comuns** do código o GIT não fará o merge automático, sendo necessário resolver "merge" ; finalmente necessitará de realizar commit do merge (manual) realizado.*