

Ficha Prática #05

5.1 Objectivos

1. Praticar a utilização de **Diagramas de Sequência** e de **Diagramas de Classe**;
2. Relacionar estes diagramas com a implementação que eles representam.

5.2 Exercícios

Para os exercícios abaixo propostos analise os enunciados e responda às questões criando os respectivos diagramas.

5.2.1 Compras online

Considere o seguinte extracto de código Java (o método `comprados(String bi)` calcula uma lista com todos os bilhetes comprados por um dado comprador). Note que a lista `res` é passada por referência no método `addBilhetes(List<String> res)`.

```
public class Compras {  
    private String nome = "";  
    private Map<String, Comprador> compradores; //idComprador->Comprador  
    ...  
  
    public List<String> comprados(String bi) {  
        List<String> res = null;  
        boolean existe = this.compradores.containsKey(bi);  
        if (existe)  
            res = this.calcula(bi);  
        return res;  
    }  
}
```

```
public List<String> calcula(String bi) {
    Comprador c = this.compradores.get(bi);
    List<String> res = new ArrayList<String>();
    c.addBilhetes(res);
    return res;
}
...
}

public class Comprador {
    private List<String> bilhetes;
    ...

    public void addBilhetes(List<String> res) {
        String o;
        int i=0;
        int tam = this.bilhetes.size();
        while(i < tam) {
            o = this.bilhetes.get(i);
            res.add(o);
            i++;
        }
    }
    ...
}
```

Relativamente ao código apresentado:

1. Analise o código e apresente o correspondente **Diagrama de Classes**.
2. Escreva um **Diagrama de Sequência** que descreva o comportamento do método `comprados(String bi)` (o modelo deverá descrever a lógica da solução e não necessariamente todo o código Java que foi escrito).
3. Considere agora que no método `addBilhetes(List<String> res)` o ciclo `while` é substituído por:

```
res.addAll(this.bilhetes);
```

Refaça o **Diagrama de Sequência** da pergunta anterior, agora com a nova versão do método.

5.2.2 Sistema de Avaliação de Trabalhos

Considere o excerto de código Java que a seguir se apresenta:

```
interface Identificavel {
    int getID();
}

abstract class Pessoa {
    private String nome;
    abstract void setNome(String n);
}

class Aluno extends Pessoa implements Identificavel {
    private Grupo m_g;
    private int numAluno;
    private int notaTeo;
    private int bounsPrat;
    void regista(Grupo g) {...};
}

class Grupo {
    private String cod;
    private int nota;
    private List<Entrega> entregas;
    void addEntrega(Entrega e) {...}
}

class Entrega implements Identificavel {
    private Date data;
    private int nota_docente;
    private Aluno avaliador;
    private int nota_avaliador;
    private String comentarios;
}

class Docente extends Pessoa implements Identificavel{
    private int cod;
}

class SGT {
    private Docente responsavel;
    private List <Docente> docentes_praticas;
    private TreeMap <Integer,Aluno> alunos;
    private List <Grupo> grupos;
```

```
int getNotaAluno(int codAluno) {...}  
void registaEntrega(Entrega e, String codGrupo) {...}  
boolean validaAvaliadores() {...}  
}
```

Relativamente ao código apresentado:

1. Analise o código e apresente o correspondente **Diagrama de Classes**, procurando ser o mais exaustivo possível na identificação dos relacionamentos entre as classes.
2. Desenhe **Diagramas de Sequência** para os seguintes métodos:
 - (a) `int getNotaAluno(int codAluno)` — O método deverá calcular a nota de um aluno, sabendo que a nota teórica e prática valem 60% e 40% da nota final, respectivamente.
 - (b) `void registaEntrega(Entrega e, String codGrupo)` — O método deverá registar uma entrega no grupo indicado, caso ainda não exista uma entrega para essa data.
 - (c) `boolean validaAvaliadores()` — O método deverá verificar que nenhum aluno seja avaliador do seu próprio grupo.