Lab VI.

Objetivos

Os objetivos deste trabalho são:

- Identificar e utilizar padrões relacionados com a construção e estrutura de objetos e classes
- Aplicar boas práticas de programação por padrões em casos práticos

Nota: Para além do código no codeUA, apresente o diagrama de classes da solução final (pode usar o UMLet, por exemplo, ou um plugin Eclipse/Netbeans).

VI.1 Empresa Pst (Petiscos e Sweets)

As empresas *Sweets* e *Petiscos* estão em processo de fusão (Pst) e precisam de integrar os seus registos de pessoal. A empresa *Sweets* usa 2 classes *Employee* e *Database*, enquanto que a empresa *Petiscos* usa *Empregado* e *Registos*.

```
// Sweets
class Employee {
     private String name;
     private long emp_num;
     private double salary;

     public Employee(String name, long emp_num, double salary) {
          this.name = name;
          this.emp_num = emp_num;
          this.salary = salary;
     }
     public String getName() {
          return name;
     }
     public long getEmpNum() {
          return emp_num;
     }
     public double getSalary() {
          return salary;
     }
}

class Database {        // Data elements
     private Vector<Employee> employees; // Stores the employees

     public Database() {
          employees = new Vector<>();
     }
     public boolean addEmployee(Employee employee) {
          // Code to add employee
     }
     public void deleteEmployee(long emp_num) {
          // Code to delete employee
     }
     public Employee[] getAllEmployees() {
          // Code to retrieve collection
     }
}
```

```java
// Petiscos
class Empregado {
    private String nome;
    private String apelido;
    private int codigo;
    private double salario;

    public Empregado(String nome, String apelido, int codigo, double salario) {
        this.nome = nome;
        this.apelido = apelido;
        this.codigo = codigo;
        this.salario = salario;
    }
    public String apelido() {
        return apelido;
    }
    public String nome() {
        return nome;
    }
    public int codigo() {
        return codigo;
    }
    public double salario() {
        return salario;
    }
}

class Registos {
    // Data elements
    private ArrayList<Empregado> empregados; // Stores the employees
    public Registos() {
        empregados = new ArrayList<>();
    }
    public void insere(Empregado emp) {
        // Code to insert employee
    }
    public void remove(int codigo) {
        // Code to remove employee
    }
    public boolean isEmpregado(int codigo) {
        // Code to find employee
    }
    public List<Empregado> listaDeEmpregados() {
        // Code to retrieve collection
    }

}
```

1) Complete o código omisso nos métodos indicados nas classes *Database* e *Registos* e escreva uma função *main* para testar cada conjunto.
2) Escreva um programa que usa ambos os conjuntos de funcionários (*Database* e *Registos*) sem alterar o código legado em qualquer uma dessas classes. O programa deve implementar os seguintes métodos:
   - Um método para adicionar um empregado.
   - Um método para remover um empregado, dado o número de funcionário
   - Um método para verificar se um empregado existe na empresa, dado o número do empregado.
   - Um método para imprimir os registos de todos os funcionários.

VI.2 Gestão dinâmica de lista de contactos

Neste problema pretende-se criar um conjunto de interfaces e classes que permitam a gestão ágil de uma lista de contactos (da classe Contact). Pretende-se nomeadamente que:

a) O armazenamento da lista possa ser feito em qualquer formato (por exemplo, TXT separado por tab, CVS, JSON, XLS, binário, etc.). Não sabemos à partida que formatos poderemos vir a criar. Para resolver este problema defina a seguinte interface que deverá ser respeitada por todas as implementações de armazenamento:

```
public interface ContactsStorageInterface {
    public List<Contact> loadContacts();
    public boolean saveContacts(List<Contact> list);
}
```

b) A utilização da lista de contacto poderá ser realizada por aplicações distintas pelo que, para separar funcionalidades, deve usar a seguinte interface:

```
public interface ContactsInterface {
    public void openAndLoad(ContactsStorageInterface store);
    public void saveAndClose();
    public void saveAndClose(ContactsStorageInterface store);
    public boolean exist(Contact contact);
    public Contact getByName(String name);
    public boolean add(Contact contact);
    public boolean remove(Contact contact);
}
```

Através desta interface deverá ser possível manipular os contactos sem saber o tipo de armazenamento usado.

Desenvolva uma solução para este problema de modo a permitir usar, pelo menos, os formatos texto e binário para armazenamento. Teste a solução com um conjunto de contactos (criados na função main de forma estática, introduzidos num ficheiro de texto, ...).