UNIVERSIDADE FEDERAL DE UBERLÂNDIA

Disciplina: **Programação Orientada a Objetos 1**
Curso: **Sistemas de Informação**
Professora: **Elaine Ribeiro Faria**

Trabalho Final da Disciplina - Parte 2

Tema: Implementação dos requisitos em Java

Instruções:
1- Faça a codificação seguindo o enunciado usando a linguagem Java.
2- Crie um arquivo txt chamado IntegrantesGrupo.txt contendo:
Nome e nro de matrícula dos integrantes do grupo
3- Envie pelo Microsoft Teams um arquivo. Zip contendo o código fonte (arquivos. Java do seu projeto) e o arquivo .txt

Data da entrega: 13/10/21 às 23:59
Avaliativa: **SIM**

Perguntas:

1- Implemente em Java, usando técnicas de encapsulamento **todo** o seu **Sistema de Clínica Médica**, que foi modelado na Parte 1 do trabalho. É muito importante que o diagrama seja corrigido antes de iniciar a implementação. A seguir um exemplo:

```
public boolean setNome(String nome){
     if (nome.length() > 0) {
          this.nome = nome; return true;
     }
     else {
          return false;
     }
}
```

   a. Para validação de cpf, implemente o algoritmo real de validação de cpf (que é encontrado facilmente na Internet, implementado nas mais diversas linguagens).
   b. Para o atributo que indica o estado civil de uma pessoa deve-se usar um atributo do tipo byte:
   - 0 - para representar o estado civil solteiro
   - 1 - para representar o estado civil casado
   - 2 - para representar o estado civil divorciado
   Na hora de exibir o estado civil de uma pessoa, deve ser exibido: “Solteiro”, “Casado” ou “Divorciado” de acordo com o valor armazenado.

   Observação sobre encapsulamento: Note que não é possível saber como este atributo está implementado, ou seja, o tipo do atributo. As demais classes do sistema apenas têm acesso à interface que dá acesso a este atributo, sem manipulá-lo diretamente. Com isto, obtemos transparência em relação à implementação interna das classes,

seus tipos de dados, etc. A transparência de implementação deve ser sempre perseguida na programação orientada a objetos.

2- Considerando o uso de construtores:
   a. Crie um construtor para a classe médico que receba como parâmetro o(s) plano(s) de saúde que ele atende, assim como seu CPF.
   b. Crie um construtor default (sem parâmetros) explicitamente para a classe Médico.
   c. Crie um construtor para a classe Consulta de forma que este receba como parâmetro o paciente e o médico. Desta forma, obriga-se que haja a associação da consulta com o paciente e o médico, que já foram previamente cadastrados no sistema.
   d. Crie construtores para cada uma das classes do problema de forma a atribuir valor aos atributos da classe com os parâmetros recebidos. Toda classe deve ter pelo menos um construtor criado por você.

3- Considerando os casos de herança, implemente todas as situações de herança descritas nos requisitos
   a. Lembre-se que existe um relacionamento de herança entre a classe funcionário (superclasse) e as subclasses médico e demais tipos de funcionários.
   b. Lembre-se também de implementar a herança entre os dois tipos de pacientes (com e sem plano).
   c. Uma herança possível de ser implementada é a que representa as pessoas do sistema e tem como subclasses pacientes e funcionários. Implemente também esse herança.
   d. Implemente um atributo estático para armazenar o nro de consultas já realizadas no mês. Crie também um método que permita zerar o nro de consultas do mês.
   e. Implemente um atributo estático para armazenar o valor que representa qual o limite de consultas que deve ser atingido por mês para os funcionários ganharem uma bonificação.
   f. Implemente um atributo estático na classe outros funcionários para armazenar o valor a ser pago de gratificação, caso o limite de consultas for atingido.
   g. Como estes atributos são encapsulados (private), crie os métodos get e set para manipulá-los. Os métodos devem ser estáticos.
   h. Implemente um método calcular Salario específico para médicos, calculado por meio da soma dos valores das consultas que o médico faz no mês. Para isso, será necessário criar um atributo SomaConsultasMes para o médico. Cada vez que ele executa uma consulta, o valor da consulta é acrescida ao SomaConsultasMes. O valor da consulta depende do tipo de paciente em questão. Crie também um método que permita zerar o valor da SomaConsultaMes.
   i. Implemente um método calcular Salario específico para outros funcionários, calculado pela soma do salário fixo mais a gratificação, caso o nro de consultas limite for atingido.

4- Considerando a realizar, crie um método realizar consulta, que altere os campos necessários da classe consulta, que atualize a data da última consulta de um cliente e que atualize os campos necessários de outras classes, como por exemplo médico e outros funcionários

5- Classes Abstratas
   a. Implemente a classe Funcionário como abstrata.
   b. Implemente o método calcularSalario como abstrato.
   c. Invente um requisito no problema que justifique a criação de um método abstrato e sua implementação nas subclasses. Este requisito pode usar as classes já existentes ou sugerir a criação de novas classes. Não esqueça de descrever o requisito

6- Classes para manter os dados
   a. Crie classes especiais capazes de manter os dados gerados no sistema. Por manter, entende-se: cadastrar, consultar e excluir. Para isto, crie as classes " DadosPacientes " , " DadosFuncionarios " , "DadosConsulta", etc.. Estas classes serão responsáveis por encapsular o acesso a cada tipo de dado específico. Para isto, estas classes devem implementar um *ArrayList* privado para armazenar a informação, e métodos públicos para permitir o acesso à informação (inserir um novo objeto, buscar um objeto, excluir um objeto, etc.). Os objetos de dados deverão ser criados apenas uma vez na classe principal e utilizados ao longo da execução da aplicação. **Exemplo:**

```
class DadosPacientes{
  private ArrayList<Pacientes> vetPac = new ArrayList<Paciente>();
    public void cadastrar(Paciente c) {
              this.vetPac.add(c); //ADICIONA  O PACIENTE NO ARRAY
              System.out.println("Total de pacientes: ");
              System.out.println(this.vetPac.size());
    }
    public void listar(){
         for (Paciente objeto: this.vetPac) {
                   objeto.mostrarDados();
                   //método mostrarDados();
         }
    }
    //este método retorna o objeto Paciente caso encontrado, ou null,
    caso não encontrado
    public Paciente buscar(String cpf) {//pode-se usar também int
         Paciente c = null;
         for ( Paciente objeto: this.vetPac) {
              if (objeto.getCPF().equals(cpf)) {
                   c = objeto;
                   break;
              }
         }
         return c;
    }
    //este método usa o método buscar já implementado
    public boolean excluir(String cpf){
         Paciente c = this.buscar(cpf);
         if (c != null) {
              this.vetPac.remove(a);
              return true;
```

```
            }
            else {
                 return false;
            }
      }
}
```

b. Acrescente ao seu programa uma funcionalidade para salvar os ArrayLists de dados em um arquivo binário. Crie um arquivo para cada ArrayList.

c. Invente um requisito no sistema que justificasse a criação de uma interface no sistema. Não esqueça de descrever o requisito.