1. Introdução teórica

Interfaces

Interface, no contexto de Programação, é um recurso de **Projeto Orientado a Objetos** que descreve, antes de se codificar uma determinada classe, como essa classe deverá apresentar-se a outras classes, expondo quais operações ela permitirá que essas outras classes tenham acesso.

Portanto, uma interface é como um protótipo de uma ou mais classes, e especifica quais **métodos públicos** deverão ser codificados nas classes que implementem (sigam) esse protótipo. A interface, portanto, é como um contrato que estabelece as obrigações (de codificação) das classes que aderirem a essa interface. Portanto, todos os métodos declarados em uma interface deverão ser implementados nas classes que adiram à interface.

Definir interfaces é uma fase do processo de projetar um software, antes mesmo de se saber quais classes serão efetivamente modeladas. Assim, nas fases iniciais do projeto de software, pensa-se sobre o que deverá ser realizado pelas classes (quais métodos as futuras classes deverão possuir para serem usados por outras classes e por aplicações) e não sobre como cada método deverá ser codificado internamente (já que há várias maneiras de se codificar cada método e, no início, saber quais técnicas serão usadas no código é algo ainda nebuloso).

Python não possui o conceito de interfaces. Ao contrário, Python possui a possibilidade de definir algo semelhante através do uso de Classes Abstratas (Como foi explicado no Capítulo 14 da apostila de Python).

Já em Java e em C#, uma interface descreve um conjunto de métodos que deverão ser codificados nas classes que implementarem a interface.

Uma interface é declarada através da palavra reservada **interface** e contém somente constantes e métodos abstratos. Todos os membros declarados na interface devem ser públicos e nenhum detalhe de codificação dos métodos pode ser implementado. A implementação dos métodos será feita nas classes que aderirem à interface, ou seja, o código interno de cada método previsto na interface será feito pelas classes que implementarem tal interface. Portanto, cada classe que aderir a uma interface poderá implementar os métodos previstos de maneira diferente de outras classes, usando recursos e técnicas mais adaptados a cada classe.

Uma aplicação de Interface

Pensemos em classes de manutenção de coleções de registros (dados relacionados). Existem diferentes implementações dessas classes. Por exemplo, usando vetores de objetos lidos de arquivos ou de tabelas de bancos de dados (como já estudamos), ou usando outras estruturas, como listas ligadas e árvores (que ainda estudaremos).

No entanto, todas essas implementações seguem a mesma diretriz: elas devem possuir métodos para ler e armazenar os dados, pesquisar dados, incluir novos dados, excluir dados existentes, alterar dados, listar dados, ordenar os dados, dentre outras operações. Esses métodos são sempre os mesmos. No entanto, a forma de cada estrutura de dados implementar (codificar) esses métodos sempre previstos varia muito, pois depende dos recursos e limitações de cada estrutura de dados.

No entanto, podemos usar uma interface para definir quais métodos uma classe de manutenção de coleções de registros deverá implementar, antes mesmo de decidirmos sobre qual estrutura de dados concreta usaremos nas fases posteriores do processo de projeto orientado a objetos.

Pensemos nas classes de manutenção de vetores e na classe abstrata VetorDados que estudamos no 1º semestre, em Python. Todas aquelas classes possuíam métodos semelhantes e que tiveram de ser implementados em cada uma delas. Portanto, essas classes seguem a mesma interface, que poderíamos chamar de ManterDados. Vamos lembrar dos métodos que codificamos para essas classes, e que toda classe de manutenção de vetores de objetos deveria ter:

```
void leituraDosDados(String nomeArquivo)
void gravarDados(String nomeArquivo)
Boolean existe(Tipo dadoProcurado)
void incluirNoFinal(Tipo novoDado)
void incluirEm(Tipo novoDado, int posicaoDeInclusao)
```

```
void excluir(int posicaoDeExclusao)
Tipo valorDe(int indiceDeAcesso)
void alterar(int posicaoDeAlteracao, Tipo novoDado)
void trocar(int origem, int destino)
void ordenar()
Boolean estaVazio()
Boolean estaNoInicio()
Boolean estaNoFim()
void irAoInicio()
void irAoFim()
void irAoAnterior()
void irAoProximo()
int getPosicaoAtual()
void setPosicaoAtual(int novaPosicao)
Situacao getSituacao()
void setSituacao(Situacao novaSituacao)
```

Tipo, citado acima, é a classe específica dos dados que serão armazenados no vetor de dados da classe que implementar os métodos acima.

Situacao, citada acima, é uma enumeração (ou tipo enumerado). É uma coleção de palavras que podemos usar em uma classe para tornar mais inteligível o sentido de alguma variável. No caso de uma classe de manutenção de vetor de dados, as situações pelas quais ele passa podem ser descritas pelo enumerador declarado abaixo:

```
enum Situacao {navegando, incluindo, excluindo, alterando, editando, buscando}
```

Uma interface de manutenção de dados, pode ser declarada usando os métodos acima. Observe que essa interface descreve os métodos que qualquer tipo de estrutura de armazenamento (vetor, lista ligada, árvore binária) necessitará implementar para que possa ser usada em um programa de manutenção de dados, independentemente da estrutura específica que se decidir, futuramente, para armazenar os dados. Portanto, interfaces servem como um recurso de projeto que permite unificar o vocabulário a ser usado em classes, permite unificar as ideias e estabelece uma diretriz para nortear a codificação das futuras classes. Outra vantagem de usar interfaces na organização do projeto é que, como todas as classes de manutenção de dados deverão ter os mesmos métodos públicos, pois seguem a mesma interface, é possível trocar a classe que implementa essa interface por outra classe que também a implementa, sem ter que alterar o código da aplicação, já que os métodos da nova classe têm as mesmas assinaturas que os métodos da classe substituída.

Para a manutenção de registros de Estudantes, poderíamos declarar uma interface como a que se segue:

```
public interface ManterDados {
    enum Situacao {navegando, incluindo, excluindo, alterando, editando,
                   buscando}
    void leituraDosDados(String nomeArquivo);
    void gravarDados(String nomeArquivo);
    Boolean existe(Estudante dadoProcurado);          // pesquisa binária
    void incluirNoFinal(Estudante novoDado);
    void incluirEm(Estudante novoDado, int posicaoDeInclusao);
    void excluir(int posicaoDeExclusao);
    Estudante valorDe(int indiceDeAcesso);
    void alterar(int posicaoDeAlteracao, Estudante novoDado);
    void trocar(int origem, int destino);
    void ordenar();
    Boolean estaVazio();
    Boolean estaNoInicio();
    Boolean estaNoFim();
    void irAoInicio();          // posicaoAtual fica no início do vetor
    void irAoFim();             // posiçãoAtual fica no fim do vetor
    void irAoAnterior();        // posiçãoAtual retrocede à posição anterior
    void irAoProximo();         // posiçãoAtual avança para a posição seguinte
```

```
    Situacao getSituacao();
    void setSituacao(Situacao novaSituacao)
}
```

O código acima seria gravado em um arquivo chamado ManterDados.java, e incorporado a um projeto de aplicação Java.

A partir dele, podemos desenvolver uma classe ManterEstudantes, que implementa os métodos descritos na interface e, também, outros métodos específicos dessa classe. Por exemplo:

```
import java.io.*;
import static java.lang.System.out;

public class ManterEstudantes implements ManterDados {
    int qtosDados,
        posicaoAtual;
    Estudante[] dados;
    Situacao situacao;

    public void leituraDosDados(String nomeArquivo)  throws IOException {
        try {
            posicaoAtual = 0;
            BufferedReader arquivoDeEntrada = new BufferedReader(
                    new FileReader("c:\\temp\\dadosEstudantes.txt"));
            String linhaDoArquivo = "";
            try {
                boolean parar = false;
                while (! parar) {
                    Estudante novoDado = new Estudante();
                    try {
                        if (novoDado.leuLinhaDoArquivo(arquivoDeEntrada) ) {
                            incluirNoFinal(novoDado);
                        }
                        else
                            parar = true;
                    }
                    catch (Exception erroDeLeitura) {
                        out.println(erroDeLeitura.getMessage());
                        parar = true;
                    }
                }
                arquivoDeEntrada.close();
            }
            catch (IOException erroDeIO) {
                out.println(erroDeIO.getMessage());
            }
        }
        catch (FileNotFoundException erro) {
            out.println(erro.getMessage());
        }
    }

    public void gravarDados(String nomeArquivo) throws IOException {
        BufferedWriter arquivoDeSaida = new BufferedWriter(
                new FileWriter("c:\\temp\\dadosEstudantes.txt"));

        for (int indice=0; indice < qtosDados; indice++)
            arquivoDeSaida.write(dados[indice].formatoDeArquivo());
        arquivoDeSaida.close();
    }

    public Boolean existe(Estudante dadoProcurado) { ... }
    public void incluirNoFinal(Estudante novoDado) { ... }
    public void incluirEm(Estudante novoDado, int posicaoDeInclusao) { ... }
    public void excluir(int posicaoDeExclusao) { ... }
    public Estudante valorDe(int indiceDeAcesso) { ... }
```

```
    public void alterar(int posicaoDeAlteracao, Estudante novoDado) { ... }
    public void trocar(int origem, int destino) { ... }
    public void ordenar() { ... }
    public Boolean estaVazio() { ... }
    public Boolean estaNoInicio() { ... }
    public Boolean estaNoFim() { ... }
    public void irAoInicio() { ... }
    public void irAoFim() { ... }
    public void irAoAnterior() { ... }
    public void irAoProximo() { ... }
    public int getPosicaoAtual() { ... }
    public void setPosicaoAtual(int novaPosicao) { ... }
    public Situacao getSituacao() { ... }
    public void setSituacao(Situacao novaSituacao) { ... }
}
```

O código acima foi praticamente copiado do projeto ClasseDeVetor, que estudamos em aula, e utiliza também métodos codificados na classe Estudante, como leuLinhaDoArquivo() e formatoDeArquivo(). Os métodos leituraDosDados() e gravarDados() foram copiados do programa Manutencao.Java.

Basicamente, os demais métodos comuns a essa classe a ao programa Manutencao.java, que tratam dos dados do vetor estud, poderiam ser copiados e adaptados à classe ManterEstudantes. Já os métodos que faltariam codificar, estão explicados no capítulo 13 da apostila de Python.

2. Descrição do projeto

Esse projeto deve ser desenvolvido em dupla, em linguagem Java, preferencialmente usando o IDE IntelliJ, para se manter a uniformidade na correção.

Ele deve incorporar o projeto Manutencao.java, desenvolvido em aula, com as opções de seletor já existentes, e adicionando algumas novas operações, que serão descritas abaixo.

Ao invés do vetor estud e da variável quantosEstudantes, deve-se usar uma instância (objeto) da classe ManterEstudantes, com seus métodos e atributos, para recodificar as operações do programa. Operações de inclusão, exclusão, pesquisa e outras devem ser codificadas nessa classe, ao invés de o serem diretamente na classe do programa principal. Tais operações devem ser chamadas pela instância da classe ManterEstudantes que substituir o vetor estud.

Deve-se ler um arquivo texto contendo siglas (6 caracteres) de até 15 disciplinas e armazená-las em um vetor de strings. Cada posição desse vetor conterá a sigla da disciplina correspondente à mesma posição do vetor notas da classe Estudante. Por exemplo:

	0	1	2	3	...	14
Siglas	Biolog	Matema	Portug	TecPro	...	

					notas					
	curso	ra	nome	qn	0	1	2	3	...	14
0	19	24105	Maria	4	8.5	10.0	7.2	7.5		
1	39	24110	João	4	3.5	8.0	5.6	8.0		
2	19	24203	Renata	4	4.8	7.2	6.5	5.8		
...	...	...	...	...	...	...	...	...		

As operações 2 e 4 devem ter cabeçalhos contendo as siglas das disciplinas acima de cada coluna de notas.

Deve-se adicionar a seguinte operação ao programa:

9 – Estatísticas:

- Disciplina com maior número de estudantes com aprovação
- Disciplina com maior número de estudantes com retenção
- Qual estudante teve a maior média de notas?
- Desse estudante, quais as disciplinas com maior e menor notas?
- Média aritmética dos alunos em cada disciplina
- Na disciplina com a menor média, qual foi o aluno com a maior nota?
- Na disciplina com a maior média, qual foi o aluno com a menor nota?

Essas operações **não devem** ser implementadas na classe VetorEstudantes, e sim na classe do programa principal, pois essas estatísticas são necessidades dessa aplicação específica, e não da classe de armazenamento de dados em si.