PUC Minas
Instituto de Ciências Exatas e Informática

Pontifícia Universidade Católica de Minas Gerais
Bacharelado em Engenharia de Software
Laboratório de Computação III
Prof.: Henrique Batista da Silva
Lista de Exercícios 2

Arquivos

Orientações:

- Os exercícios devem ser feitos em **linguagem Java**.
- Cada exercício deve ser feito em um arquivo .java (extensão dos arquivos da linguagem Java). Você deve entregar apenas um único arquivo compactado (.zip) no SGA contendo todos os exercícios resolvidos. **Não entregue a pasta do projeto do Eclipse, e sim apenas os arquivos ".java" do seu programa.**
- Exercícios copiados receberão nota zero.
- Não deixe a lista para a última hora. Comece o quanto antes. Assim haverá tempo para esclarecer dúvidas com o professor e na monitoria.

Parte I

Utilize as informações a seguir para criar um controle automatizado de uma clínica média. Sabe-se que essa clínica deseja ter um controle semanal (2a a 6a feira) das consultas realizadas. A cada dia podem ser realizadas no máximo três consultas para cada médico. Considere que são cadastrados apenas 3 médicos e 15 pacientes.

PACIENTE (COD_PACIENTE, NOME, ENDERECO, TELEFONE)
MEDICO (COD_MEDICO, NOME, TELEFONE)
CONSULTA (NUM_CONSULTA, DIA_SEMANA, MÊS, ANO, HORA, COD_MEDICO, COD_PACIENTE)

a. Construa, usando classes, os tipos necessários para armazenar os PACIENTES, MEDICOS e CONSULTAS.

b. Usando os tipos do item anterior, implemente um procedimento para cadastrar um paciente. Este procedimento deve garantir que não haverá mais de um paciente com o mesmo código. O nome de cada paciente deve ser lido da tela (entrada padrão). As informações dos pacientes devem ser salvas em um arquivo binário em disco chamado "pacientes.dat" (ver código no final do enunciado)

c. Faça um procedimento para cadastro dos médicos. Siga as mesmas instruções do item b (o arquivo binário deve se chamar "medicos.dat").

d. Implemente um procedimento que cadastre uma consulta como ela é descrita no enunciado do problema. Lembre-se que as consultas só podem ser marcadas de 2a a 6a feira. Lembre-se também que cada médico só pode atender dois pacientes por dia. Ao final de um dia, o sistema deve permitir salvar o histórico de todas as consultas em um arquivo binário ("consultas.dat"). Este arquivo deve manter um histórico de todas as consultas do sistema.

e. Implemente uma função que receba o nome de um médico e um determinado dia da semana; percorra os dados cadastrados e imprima na tela quantas consultas estão agendadas para este médico neste determinado dia.

f. Seu programa deve permitir a geração de relatório de consultas. Deve haver a opção para gerar, na tela e em arquivo texto ("relatório.txt") de todos as consultas do sistema. Deve haver também a opção para saber as consultas de um determinado dia somente. As informações do relatório devem conter todas as informações de uma consulta.

```
import java.io.Serializable;

public class Paciente implements Serializable {

	private int codigo;
	private String nome;

	public Paciente(int codigo, String nome) {
		super();
		this.codigo = codigo;
		this.nome = nome;
	}

	public int getCodigo() {
		return codigo;
	}
	public void setCodigo(int codigo) {
		this.codigo = codigo;
	}
	public String getNome() {
		return nome;
	}
	public void setNome(String nome) {
		this.nome = nome;
	}

}


import java.io.FileOutputStream;
```

```java
import java.io.IOException;
import java.io.ObjectOutputStream;

public class ArquivoDeDadosEscrita {

	private ObjectOutputStream output;

	public void openFile(String file) throws IOException {
		output = new ObjectOutputStream(  new FileOutputStream(file) );
	}

	public void setData(Object data) throws IOException {

		output.writeObject( data );
	}

	public void closeFile() throws IOException {
		if ( output != null )
			output.close();
	}
}
import java.io.EOFException;
import java.io.FileInputStream;
import java.io.IOException;
import java.io.ObjectInputStream;

public class ArquivoDeDadosLeitura {

	private ObjectInputStream input;

	public void openFile(String file) throws IOException {
		input = new ObjectInputStream(  new FileInputStream(file) );
	}

	public Object getData() throws IOException, ClassNotFoundException {

		Object data;

		try {
			data = input.readObject();
		}
		catch (EOFException endOfFileException) {
			return null;
		}

		return data;
	}

	public void closeFile() throws IOException {
		if ( input != null )
			input.close();
	}
}
```

```
public static void main(String[] args) throws IOException, ClassNotFoundException {
		// TODO Auto-generated method stub

		String arquivo = "paciente.bin";

		Paciente p = new Paciente(1, "Alice");

		ArquivoDeDadosEscrita escrita = new ArquivoDeDadosEscrita();

		escrita.openFile(arquivo);
		escrita.setData(p);
		escrita.closeFile();

		ArquivoDeDadosLeitura leitura = new ArquivoDeDadosLeitura();

		leitura.openFile(arquivo);
		Paciente a;

		while ((a = (Paciente)leitura.getData()) != null) {
			System.out.println( a.getCodigo() +" - "+ a.getNome() );
		}

		leitura.closeFile();


	}
```

Parte II

Modifique o exercício da parte II para que o cadastro dos médicos, pacientes e consultas seja feito usando a biblioteca do Java RandomAccessFile. Esta biblioteca permite que os registro em arquivo binário em disco sejam acessados de forma direta (sem a necessidade de ler todos os registros sequencialmente). Utilize o código a seguir como exemplo para implementar a classe de acesso a dados de Paciente, Médico e Consultas. Implemente um programa principal para teste e que permita a busca direta de médicos, pacientes ou consultas pelo número de seus respectivos registro e imprima todos os seus atributos na tela.

```
import java.io.File;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.io.RandomAccessFile;

public class ArquivoDeDadosDeAcessoAleatorioPaciente {

	private RandomAccessFile file;
	private int numReg;
	private int tamReg;
	private int tamHead;
```

```
	private static final int STRING_MAX_TAM = 30;

	public ArquivoDeDadosDeAcessoAleatorioPaciente() {
		this.file = null;
		this.numReg = -1;		// número de registro (-1: não há registros)
		this.tamReg = (Integer.SIZE / 8) + STRING_MAX_TAM + STRING_MAX_TAM +
(Integer.SIZE / 8);
		this.tamHead = 4;
	}

	public void openFile(String path) {

		File f = new File(path);

		// se arquivo no existe definie numReg como 0. Se o arquivo já existe,
mantenha
		// como -1
		if (!f.exists())
			this.numReg = 0;

		try {
			file = new RandomAccessFile(f, "rw");
		} catch (FileNotFoundException e) {
			System.out.println("File not found");
			System.exit(0);
		}

		// se numReg é igual a -1, há registros dentro do arquivo, então
verifica a
		// quantidade de registros
		if (this.numReg == -1)
			this.setNumReg();

	}

	private void setNumReg() {
		try {
			this.numReg = file.readInt();

		} catch (IOException e) {
			System.out.println("Error!");
			System.exit(0);
		}
	}

	public void openFileReadOnly(String path) {
		try {
			file = new RandomAccessFile(new File(path), "r");
		} catch (FileNotFoundException e) {
			System.out.println("File not found");
			System.exit(0);
		}

		this.setNumReg();
	}
```

```
	public int getNumReg() {
		return numReg;
	}

	public void setData(Paciente p) {
		int pos = this.tamHead + (this.numReg * this.tamReg);	// calcula ponteiro para a primeira posição vazia do arquivo

		try {
			file.seek(pos);

			file.writeInt(p.getCodigo());
			file.writeUTF(p.getNome());
			file.seek(pos + (Integer.SIZE / 8) + STRING_MAX_TAM);

			file.writeUTF(p.getSobrenome());
			file.seek(pos + (Integer.SIZE / 8) + STRING_MAX_TAM + STRING_MAX_TAM);

			file.writeInt(p.getIdade());

			file.seek(0);
			this.numReg += 1;
			file.writeInt(this.numReg);

		} catch (IOException e) {
			System.out.println("Error!");
			System.exit(0);
		}
	}

	public Paciente getData(int key) {

		if (key >= this.numReg)
			return null;

		int pos = this.tamHead + (key * this.tamReg);

		Paciente p = new Paciente();

		try {
			file.seek(pos);

			p.setCodigo(file.readInt());
			p.setNome(file.readUTF());

			file.seek(pos + (Integer.SIZE / 8) + STRING_MAX_TAM);

			p.setSobrenome(file.readUTF());

			file.seek(pos + (Integer.SIZE / 8) + STRING_MAX_TAM + STRING_MAX_TAM);

			p.setIdade(file.readInt());
```

```
			} catch (IOException e) {
				System.out.println("Error");
				System.exit(0);
			}

			return p;
		}

		public void closeFile(String path) {
			try {
				file.close();

			} catch (IOException e) {
				System.out.println("Error");
				System.exit(0);
			}
		}
}
```