

GCC178 - Práticas de Programação Orientada a Objetos

Arquivos e Serialização

Luiz Henrique de Campos Merschmann
Departamento de Computação Aplicada
Universidade Federal de Lavras

luiz.hcm@ufla.br

Na Aula de Hoje



Arquivos em Java

Tipos de Arquivos

Temos basicamente dois tipos de arquivo:

- ▶ Arquivos de texto:
 - ▶ São formados por caracteres alfanuméricos e símbolos, os quais podem ser facilmente lidos abrindo-se o arquivo em um editor de texto simples.
- ▶ Arquivos binários:
 - ▶ São formados por uma sequência de registros (ou objetos).
 - ▶ Podem ser lidos por programas que entendam o conteúdo específico do arquivo (ex.: arquivos de imagem).

Classes para Manipulação de Arquivos

A Java API inclui o pacote `java.io`, que:

- ▶ Contém várias classes para suportar operações de entrada/saída de dados.
- ▶ Possui classes que se dividem em duas categorias:
 - ▶ Classes que lidam com arquivos de texto, conhecidas como leitores/escritores (*readers/writers*).
 - ▶ Classes que lidam com arquivos binários, conhecidas como tratadoras de fluxo (*stream handlers*).

Classes para Manipulação de Arquivos de Texto

Para gravar dados em um arquivo de texto:

- ▶ Usaremos a classe *FileWriter*, cujo construtor recebe o nome do arquivo a ser gravado.
 - ▶ O nome do arquivo pode ser do tipo *String* ou um objeto *File*.
- ▶ Usaremos os métodos *write* e *close* da classe *FileWriter*.

Para ler dados de um arquivo de texto:

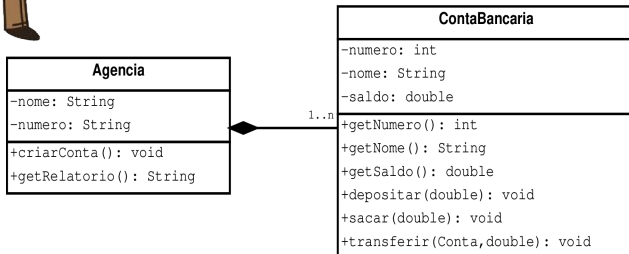
- ▶ Usaremos as classes *BufferedReader* e *FileReader*.
 - ▶ Embora a classe *FileReader* possua um método para ler um único caractere, ela não contém um método para ler uma linha.
 - ▶ Como a classe *BufferedReader* possui um método para ler uma linha, o objeto *FileReader* é empacotado em um objeto *BufferedReader*.
- ▶ Usaremos os métodos *readLine* e *close* da classe *BufferedReader*.

Exemplo: Arquivos de Texto



Vamos mostrar um exemplo de escrita/leitura em arquivo texto de dados de contas de uma agência bancária.

Para isso, consideraremos a modelagem apresentada no diagrama de classe a seguir. Observe que uma Agência possui uma lista de contas e que cada conta possui vários atributos.



Arquivo Texto: Escrita

```
import java.io.*;
public void salvarDadosContas(String nomeArquivo){
    try{
        FileWriter arq = new FileWriter(nomeArquivo);

        for(ContaBancaria conta: contas){
            arq.write(conta.getNumero() + "," + conta.getNome() + "," + conta.getSaldo() + "\n");
        }
        arq.close();
    }
    catch(IOException e){
        System.out.println("Falha ao salvar no arquivo " + nomeArquivo);
    }
}
```

Não podemos esquecer de **importar** a classe *FileWriter* do pacote **java.io**.

Criando um objeto da classe *FileWriter* passando como parâmetro o nome do arquivo a ser salvo.

Uma vez que toda saída tenha sido gravada, devemos fechar o arquivo.

O método *write* recebe uma string. Aqui estamos escrevendo os dados de cada conta bancária em uma linha do arquivo.

Arquivo Texto: Leitura

Neste exemplo, iremos ler de um arquivo os dados de diversas contas bancárias e armazená-los no `ArrayList` `contas`.

Utilizaremos as classes `BufferedReader` e `FileReader`, que pertencem ao pacote `java.io`.

```
import java.io.*;
public void carregarDadosContas(String nomeArquivo){
    try{
        BufferedReader arq = new BufferedReader(new FileReader(nomeArquivo));

        String linha = arq.readLine();

        while(linha != null){
            String[] campos = linha.split(",");

            ContaBancaria conta = new ContaBancaria(Integer.parseInt(campos[0]), campos[1]);
            conta.depositar(Float.parseFloat(campos[2]));

            contas.add(conta);

            linha = arq.readLine();
        }
        arq.close();
    } catch(FileNotFoundException e){
        System.out.println("Impossível abrir o arquivo " + nomeArquivo);
    } catch(IOException e){
        System.out.println("Problema na leitura do arquivo " + nomeArquivo);
    }
}
```

Criando um objeto da classe que lerá o arquivo passando como parâmetro o nome do arquivo a ser lido.

O método `readLine` retorna uma string com a linha lida do arquivo.

O método `split` divide a *string* em *substrings* de acordo com o **separador** passado como parâmetro.

Instanciando um objeto `ContaBancaria` com dados lidos de uma linha do arquivo.

Utilizamos os **métodos estáticos** `parseInt` e `parseFloat` para converter as *strings* em *int* e *float*, respectivamente.

A variável `linha` receberá valor **null** quando o arquivo chegar ao fim.

Uma vez que a leitura foi finalizada, devemos fechar o arquivo.

Arquivo

```
=====
1234, Luiz, 100
2345, José, 250
3456, João, 350
4567, Pedro, 50
```

Arquivos em Java

Observação:

- ▶ Devemos criar códigos de leitura e escrita em um arquivo compatíveis entre si.
 - ▶ Ou seja, primeiro devemos definir o formato dos dados no arquivo, para depois implementar os códigos de leitura e escrita com base nesse formato.

Serialização

Introdução



A **serialização** permite que **objetos inteiros** sejam lidos e gravados em uma **única operação**.

Ou seja, **transformamos** um objeto em um **array de bytes** para:

- Salvá-lo em um **arquivo**, enviá-lo via **rede** etc.

E podemos também realizar o **processo inverso (desserializar o objeto)**, ou seja, a partir do arquivo ou da rede trazemos o objeto novamente para memória.



Esse é um recurso importante, pois **evita** termos que **ler e gravar objetos campo por campo** (como fizemos com arquivos de texto).

Requisitos



Para poder participar da serialização, cada **classe** que terá seus **objetos serializados/desserializados** precisa **implementar** a interface ***Serializable*** definida no pacote **java.io**.

```
public class MinhaClasse implements Serializable { ... }
```

Vale observar que a interface *Serializable* não define nenhum método, ou seja, ela **serve** apenas para **indicar** para a JVM que os **objetos** daquela classe **podem ser serializados**.

É recomendado que as classes que implementam *Serializable* tenham uma constante que indique a sua versão. Exemplo:

```
public class Pessoa implements Serializable {  
    private static final long serialVersionUID = 1L;  
    private String nome;  
}
```

Considere que você tenha serializado um objeto e, depois disso, tenha alterado a sua classe (p.ex, inserido atributo). Quando for desserializar o objeto, ele não será compatível com a classe alterada. Essa constante ajuda a JVM identificar essa situação e reportar um erro específico.

Serialização e Arquivos Binários



Como já dissemos, podemos serializar objetos para salvá-los em arquivos.

Nesse caso, uma vez que os **objetos serializados** correspondem a **arrays de bytes**, estaremos manipulando **arquivos binários**.

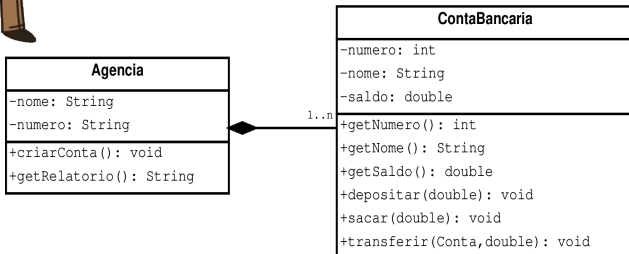
Vale lembrar que, de forma geral, não é possível ver claramente o conteúdo de um arquivo binário usando um editor de texto comum.

Exemplo de Serialização



Vamos mostrar um exemplo de serialização salvando/recuperando dados de contas bancárias de uma agência bancária.

Para isso, consideraremos a modelagem apresentada no diagrama de classe a seguir. Observe que uma Agência possui uma lista de contas e que cada conta possui vários atributos.



Arquivo Binário: Escrita



Para **salvamos** dados em um **arquivo binário** podemos usar as **classes** `FileOutputStream` e `ObjectOutputStream` e os métodos `writeObject` e `close` da última classe.

Observe neste código que estamos salvando o objeto `Agencia` todo, ou seja, os dados de todas as contas da agência.

```
public static void salvarEmArquivo(Agencia ag, String arquivoDestino){  
    try{  
        ObjectOutputStream oos = new ObjectOutputStream(  
            new FileOutputStream(arquivoDestino));  
        oos.writeObject(ag);  
        oos.close();  
    }  
    catch(IOException e){  
        System.out.println(e.getMessage());  
    }  
}
```


Arquivo Binário: Leitura



Para **recuperarmos** dados de um **arquivo binário** podemos usar as **classes** *FileInputStream* e *ObjectInputStream* e os métodos *readObject* e *close* da última classe.

```
public static Agencia lerDoArquivo(String arquivoOrigem){  
    Agencia ag = null;  
    try{  
        ObjectInputStream ois = new ObjectInputStream(  
            new FileInputStream(arquivoOrigem));  
        ag = (Agencia)ois.readObject();  
        ois.close();  
    }  
    catch(Exception e){  
        System.out.println(e.getMessage());  
    }  
    return ag;  
}
```

Arquivo Binário x Arquivo Texto

Ao longo do curso vimos como salvar dados em **arquivo binário** e em **arquivo texto**.

Afinal, qual deles é melhor utilizar?

Arquivos binários apresentam algumas **vantagens**:

- Permitem salvar de forma rápida e simples um ou mais objetos.
- São mais rápidos para serem lidos/escritos e os arquivos ficam menores.

No entanto, têm **desvantagem** quando a classe sofre alterações.

Arquivos texto têm algumas **desvantagens**:

- Dão mais trabalho para serem manipulados (você precisa tratar cada atributo do seu objeto).
- São mais lentos para serem lidos/escritos e os arquivos ficam maiores.

No entanto, quando as classes sofrem alterações com frequência, **pode ser benéfico** você ter o controle da leitura/escrita de cada atributo.

Fechamento de Recursos no Java

```
import java.io.*;

public void salvarDadosContas(String nomeArquivo){
    try{
        FileWriter arq = new FileWriter(nomeArquivo);

        for(ContaBancaria conta: contas){
            arq.write(conta.getNumero() + "," + conta.getNome() + "," + conta.getSaldo() + "\n");
        }
        arq.close();
    }
    catch(IOException e){
        System.out.println("Falha ao salvar no arquivo " + nomeArquivo);
    }
}
```

Fechamento de Recursos no Java

```
import java.io.*;
public void salvarDadosContas(String nomeArquivo){
    try{
        FileWriter arq = new FileWriter(nomeArquivo);

        for(ContaBancaria conta: contas){
            ➔ arq.write(conta.getNumero() + "," + conta.getNome() + "," + conta.getSaldo() + "\n");
        }
        arq.close();
    }
    catch(IOException e){
        System.out.println("Falha ao salvar no arquivo " + nomeArquivo);
    }
}
```

O que acontece se ocorrer um **erro ao tentarmos gravar dados** em um arquivo?

Fechamento de Recursos no Java

```
import java.io.*;
public void salvarDadosContas(String nomeArquivo){
    try{
        FileWriter arq = new FileWriter(nomeArquivo);

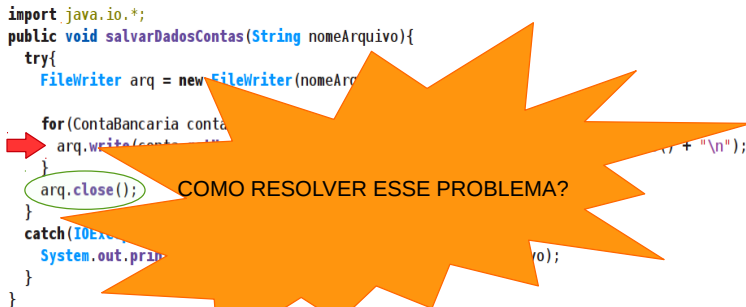
        for(ContaBancaria conta: contas){
            → arq.write(conta.getNumero() + "," + conta.getNome() + "," + conta.getSaldo() + "\n");
        }
        arq.close();
    }
    catch(IOException e){
        System.out.println("Falha ao salvar no arquivo " + nomeArquivo);
    }
}
```

O que acontece se ocorrer um **erro ao tentarmos gravar dados** em um arquivo?

Resp.: Uma exceção é lançada e, portanto, o **fechamento do arquivo não é realizado**.

Fechamento de Recursos no Java

```
import java.io.*;
public void salvarDadosContas(String nomeArquivo){
    try{
        FileWriter arq = new FileWriter(nomeArquivo);
        for(ContaBancaria conta : contas){
            arq.write(conta.toString() + "\n");
        }
        arq.close();
    } catch (IOException e){
        System.out.println("Erro ao salvar dados.");
    }
}
```



COMO RESOLVER ESSE PROBLEMA?

O que acontece se ocorrer um **erro ao tentarmos gravar dados** em um arquivo?

Resp.: Uma exceção é lançada e, portanto, o **fechamento do arquivo não é realizado**.

Fechamento de Recursos no Java

```
import java.io.*;
public void salvarDadosContas(String nomeArquivo){
    FileWriter arq = null;
    try{
        FileWriter arq = new FileWriter(nomeArquivo);
        arq = new FileWriter(nomeArquivo);
        for(ContaBancaria conta: contas){
            arq.write(conta.getNumero() + "," + conta.getNome() + "," + conta.getSaldo() + "\n");
        }
        arq.close();
    }
    catch(IOException e){
        System.out.println("Falha ao salvar no arquivo " + nomeArquivo);
    }
    finally{
        if(arq != null){
            try{
                arq.close();
            }
            catch(IOException e){
                System.err.println(e.getMessage());
            }
        }
    }
}
```

Fechamento de Recursos no Java

```
import java.io.*;

public void salvarDadosContas(String nomeArquivo){
    FileWriter arq = null;
    try{
        FileWriter arq = new FileWriter(nomeArquivo);
        arq = new FileWriter(nomeArquivo);
        for(ContaBancaria conta: contas){
            arq.write(conta.getNumero() + "," + conta.getNome() + "," + conta.getSaldo() + "\n");
        }
        arq.close();
    }
    catch(IOException e){
        System.out.println("Falha ao salvar no arquivo " + nomeArquivo);
    }
    finally{
        if(arq != null){
            try{
                arq.close();
            }
            catch(IOException e){
                System.err.println(e.getMessage());
            }
        }
    }
}
```

Para que serve?

Fechamento de Recursos no Java

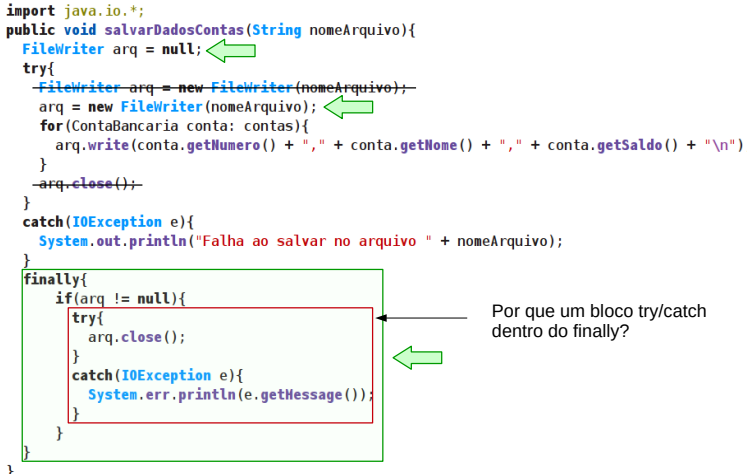
```
import java.io.*;
public void salvarDadosContas(String nomeArquivo){
    FileWriter arq = null;
    try{
        FileWriter arq = new FileWriter(nomeArquivo);
        arq = new FileWriter(nomeArquivo);
        for(ContaBancaria conta: contas){
            arq.write(conta.getNumero() + "," + conta.getNome() + "," + conta.getSaldo() + "\n");
        }
        arq.close();
    }
    catch(IOException e){
        System.out.println("Falha ao salvar no arquivo " + nomeArquivo);
    }
    finally{
        if(arq != null){
            try{
                arq.close();
            }
            catch(IOException e){
                System.err.println(e.getMessage());
            }
        }
    }
}
```

Para que serve?

Resp.: Para verificar se o arquivo realmente foi aberto, pois um erro pode ter ocorrido exatamente na tentativa de abertura do arquivo.

Fechamento de Recursos no Java

```
import java.io.*;
public void salvarDadosContas(String nomeArquivo){
    FileWriter arq = null;
    try{
        FileWriter arq = new FileWriter(nomeArquivo);
        arq = new FileWriter(nomeArquivo);
        for(ContaBancaria conta: contas){
            arq.write(conta.getNumero() + "," + conta.getNome() + "," + conta.getSaldo() + "\n");
        }
        arq.close();
    }
    catch(IOException e){
        System.out.println("Falha ao salvar no arquivo " + nomeArquivo);
    }
    finally{
        if(arq != null){
            try{
                arq.close();
            }
            catch(IOException e){
                System.err.println(e.getMessage());
            }
        }
    }
}
```



Por que um bloco try/catch dentro do finally?

Fechamento de Recursos no Java

```
import java.io.*;
public void salvarDadosContas(String nomeArquivo){
    FileWriter arq = null;
    try{
        FileWriter arq = new FileWriter(nomeArquivo);
        arq = new FileWriter(nomeArquivo);
        for(ContaBancaria conta: contas){
            arq.write(conta.getNumero() + "," + conta.getNome() + "," + conta.getSaldo() + "\n");
        }
        arq.close();
    }
    catch(IOException e){
        System.out.println("Falha ao salvar no arquivo " + nomeArquivo);
    }
    finally{
        if(arq != null){
            try{
                arq.close();
            }
            catch(IOException e){
                System.err.println(e.getMessage());
            }
        }
    }
}
```

Por que um bloco try/catch dentro do finally?

Resp.: Porque o método de fechamento do arquivo (close) também pode lançar exceção.

Fechamento de Recursos no Java

```
import java.io.*;

public void salvarDadosContas(String nomeArquivo){
    FileWriter arq = null;
    try{
FileWriter arq = new F
        arq = new F
        for(ContaBancaria c : contas){
            arq.write(c.getSaldo() + "\n");
        }
arq.close();
    }
    catch(IOException e){
        System.out.println("Falha ao salvar no arquivo " + nomeArquivo);
    }
    finally{
        if(arq != null){
            try{
                arq.close();
            }
            catch(IOException e){
                System.err.println(e.getMessage());
            }
        }
    }
}
```

Quais os problemas com esta abordagem?

Fechamento de Recursos no Java

```
import java.io.*;
public void salvarDadosContas(String nomeArquivo){
    FileWriter arq = null;
    try{
        FileWriter arq = new F
        arq = new F
        for(ContaBancaria c : contas){
            arq.write(c.getSaldo() + "\n");
        }
        arq.close();
    }
    catch(IOException e){
        System.out.println("Falha ao salvar no arquivo " + nomeArquivo);
    }
    finally{
        if(arq != null){
            try{
                arq.close();
            }
            catch(IOException e){
                System.err.println(e.getMessage());
            }
        }
    }
}
```

Quais os problemas com esta abordagem?

- O esquecimento do **if** no bloco **finally**, pode resultar num **NullPointerException**.
- Se o método **close()** do bloco **finally** lançar uma exceção, essa exceção lançada “esconderá” qualquer outra exceção lançada no bloco **try**.

Fechamento de Recursos no Java

```
import java.io.*;
public void salvarDadosContas(String nomeArquivo){
    FileWriter arq = null;
    try{
FileWriter arq = new FileWriter(nomeArquivo);
        arq = new FileWriter(nomeArquivo);
        for(ContaBancaria conta: contas){
            arq.write(conta.getNumero() + "," + conta.getNome() + " + conta.getSaldo() + "\n");
        }
arq.close();
    }
    catch(IOException e){
        System.err.println(e.getMessage());
    }
    finally{
        if(arq != null){
            try{
                arq.close();
            }
            catch(IOException e){
                System.err.println(e.getMessage());
            }
        }
    }
}
```

Como resolver esses problemas?

Fechamento de Recursos no Java

TRY-WITH-RESOURCES

- A finalidade da sintaxe try-with-resources é eliminar a necessidade de se fechar explicitamente todos os recursos abertos.
- Além disso, ela lida automaticamente com os casos especiais mencionados anteriormente.

```
import java.io.*;
public void salvarDadosContas(String nomeArquivo){
    FileWriter arq = null;
    try(FileWriter arq = new FileWriter(nomeArquivo)){
        FileWriter arq = new FileWriter(nomeArquivo);
        for(ContaBancaria conta: contas){
            arq.write(conta.getNumero() + "," + conta.getNome() + "," + conta.getSaldo() + "\n");
        }
        arq.close();
    }
    catch(IOException e){
        System.out.println("Falha ao salvar no arquivo " + nomeArquivo);
    }
}
```

Perguntas?

