fusberti@ic.unicamp.br



# Manipulação de Arquivos

prof. Fábio Luiz Usberti

MC322 - Programação Orientada a Objetos

Instituto de Computação - UNICAMP - 2014


# Sumário

# Introdução

## Arquivos binários e arquivos de texto

- Arquivos são tipos de fluxos de dados (*streams*) que podem ser utilizados para a entrada e saída de dados binários ou texto.
- Assim como os arquivos binários, os arquivos texto também armazenam dados em codificação binária, no entanto, cada byte representa um caractere.
- Arquivos texto são decodificados automaticamente por editores de texto, enquanto arquivos binários são lidos por programas que sabem traduzir o conteúdo binário em alguma informação.
- A linguagem Java utiliza o padrão Unicode para armazenamento e recuperação de dados em arquivos texto.

# Classe `File`

## Informações sobre arquivos e diretórios

- A classe `File` (`java.io`) é útil para recuperar informações referentes a arquivos e diretórios armazenados em disco.
- Os objetos `File` não foram definidos para abrir ou manipular arquivos, no entanto eles são frequentemente adotados para a verificação dos arquivos e diretórios que serão manipulados por outros objetos do pacote `java.io`.

# Classe `File`

## Criado objetos `File`

- Um construtor da classe `File` é definido como segue:

```
public File(String pathname)
```

- O argumento `pathname` deve conter o caminho de um arquivo arquivo ou diretório armazenado em disco.
- O caminho pode ser absoluto, iniciando-se da raiz do disco, ou relativo, iniciando-se do diretório onde o aplicativo está sendo executado.

# Classe `File`

## Métodos da classe `File`

| Método | Descrição |
|---|---|
| `boolean canRead()` | Retorna `true` se um arquivo tem permissão de leitura para o aplicativo corrente; retorna `false` caso contrário. |
| `boolean canWrite()` | Retorna `true` se um arquivo tem permissão de escrita para o aplicativo corrente; retorna `false` caso contrário. |
| `boolean isFile()` | Retorna `true` se o nome especificado no argumento é um arquivo; retorna `false` caso contrário. |
| `boolean isDirectory()` | Retorna `true` se o nome especificado no argumento é um diretório; retorna `false` caso contrário. |

# Classe `File`

## Métodos da classe `File`

| Método | Descrição |
|---|---|
| `String getAbsolutePath()` | Retorna uma `String` com o caminho absoluto do arquivo ou diretório associado ao objeto `File`. |
| `String getName()` | Retorna uma `String` com o nome do arquivo ou diretório associado ao objeto `File`. |
| `String getPath()` | Retorna uma `String` com o caminho relativo do arquivo ou diretório associado ao objeto `File`. |
| `String getParent()` | Retorna uma `String` com o diretório pai do arquivo ou diretório associado ao objeto `File`. |

# Classe `File`

## Métodos da classe `File`

| Método | Descrição |
|---|---|
| `long length()` | Retorna o comprimento, em bytes, do arquivo associado ao objeto `File`. Se o objeto estiver associado a um diretório, retorna um valor não especificado. |
| `long lastModified()` | Retorna a última data na qual o arquivo ou diretório associado ao objeto `File` foi modificado. |
| `String[] list()` | Retorna um array de `String` representando o conteúdo do diretório associado ao objeto `File`. Retorna `null` se o objeto `File` estiver associado a um arquivo. |

# Classe File

## Métodos da classe File

```
// FileDemonstration.java
// A classe FileDemonstration demonstra o uso da classe File.
import java.io.File;
import java.util.Date;
import java.util.Scanner;

public class FileDemonstration {
    // mostra informações de um arquivo/diretório especificado pelo usuário
    public void analyzePath(String pathname) {

        File name = new File(pathname);

        // Se arquivo/diretório existe, imprime informações sobre ele
        if (name.exists()) {

            System.out
                    .printf("%s%s\n%s\n%s\n%s\n%s%s\n%s%s\n%s%s\n%s%s\n%s%s",
                            name.getName(), " existe",
                            (name.isFile() ? "É um arquivo"
                                    : "Não é um arquivo"),
                            (name.isDirectory() ? "É um diretório"
                                    : "Não é um diretório"),
                            (name.isAbsolute() ? "É um caminho absoluto"
                                    : "É um caminho relativo"),
                            "Última modificação: ", (new Date(name.lastModified())).toString(),
                            "Tamanho (bytes): ", name.length(),
                            "Caminho relativo: ", name.getPath(),
                            "Caminho absoluto: ", name.getAbsolutePath(),
                            "Diretório pai: ", name.getParent());
    /* continua na próxima página */
```

# Classe File

## Métodos da classe File

```
    /* continua da página anterior */
            // Se for um diretório imprima seu conteúdo
            if (name.isDirectory()) {
                String directory [] = name.list();
                System.out.println("\n\nConteúdo do diretório:\n");

                for (String directoryName : directory)
                    System.out.printf("%s\n", directoryName);
            } // fim if
        } else {
            System.out.printf("%s %s", pathname, "não existe.");
        } // fim if .. else
    } // fim método analyzePath

    public static void main(String args[]) {
        Scanner input = new Scanner(System.in);
        FileDemonstration application = new FileDemonstration();

        System.out.print("Entre com o caminho de um arquivo ou diretório: ");
        application.analyzePath(input.nextLine());
    } // end main

} // fim classe FileDemonstration
```

# E/S em Arquivos

## Entradas e Saídas Padrões

- Um programa Java abre um arquivo instanciando um objeto associado a um fluxo de dados binários ou de texto.
- Quando um programa Java inicia execução, três objetos são automaticamente instanciados e associados às entradas e saídas padrões (`System.in`, `System.out`, `System.err`).
- É possível redirecionar as entradas e saídas padrões através dos métodos `setIn`, `setOut`, `setErr`.

# E/S em Arquivos

## Pacote `java.io`

- Um programa Java realiza o processamento de arquivos utilizando classes do pacote `java.io`.
- O pacote `java.io` inclui as classes `FileInputStream` e `FileOutputStream` para a manipulação de arquivos binários, e as classes `FileReader` e `FileWriter` para a manipulação de arquivos texto.
- Essas classes herdam das classes `InputStream`, `OutputStream`, `Reader` e `Writer`, respectivamente. Essas superclasses definem fluxos de dados utilizados por diferentes dispositivos, como arquivos, banco de dados, uma conexão de rede, uma interface gráfica, etc.

# E/S em Arquivos

## Pacote `java.io`

Fluxo de dados binários: entrada.

- `java.io.InputStream`
  - `java.io.ByteArrayInputStream`
  - `java.io.FileInputStream`
  - `java.io.FilterInputStream`
    - `java.io.BufferedInputStream`
    - `java.io.DataInputStream`
    - `java.io.LineNumberInputStream`
    - `java.io.PushbackInputStream`
  - `java.io.ObjectInputStream`
  - `java.io.PipedInputStream`
  - `java.io.SequenceInputStream`
  - `java.io.StringBufferInputStream`

# E/S em Arquivos

## Pacote `java.io`

Fluxo de dados binários: saída.

- `java.io.OutputStream`
  - `java.io.ByteArrayOutputStream`
  - `java.io.FileOutputStream`
  - `java.io.FilterOutputStream`
    - `java.io.BufferedOutputStream`
    - `java.io.DataOutputStream`
    - `java.io.PrintStream`
  - `java.io.ObjectOutputStream`
  - `java.io.PipedOutputStream`

# E/S em Arquivos

## Pacote `java.io`

Fluxo de dados de texto: entrada.

- `java.io.Reader`
  - `java.io.BufferedReader`
    - `java.io.LineNumberReader`
  - `java.io.CharArrayReader`
  - `java.io.FilterReader`
    - `java.io.PushbackReader`
  - `java.io.InputStreamReader`
    - `java.io.FileReader`
  - `java.io.PipedReader`
  - `java.io.StringReader`

# E/S em Arquivos

## Pacote `java.io`

Fluxo de dados de texto: saída.

- `java.io.Writer`
  - `java.io.BufferedWriter`
  - `java.io.CharArrayWriter`
  - `java.io.FilterWriter`
  - `java.io.OutputStreamWriter`
    - `java.io.FileWriter`
  - `java.io.PipedWriter`
  - `java.io.PrintWriter`
  - `java.io.StringWriter`

# E/S em Arquivos

## Classes Scanner e Formatter

- Além das classes no pacote `java.io`, arquivos texto podem ser lidos e escritos através das classes `Scanner` e `Formatter`.
- A classe `Scanner` é muito utilizada para a entrada de dados (com buffer) do terminal ou de um arquivo.
- A classe `Formatter` permite a saída de dados (com buffer) para qualquer fluxo de dados de caracteres (texto), em especial, arquivos.
- A vantagem da classe `Formatter` consiste em fornecer uma saída formatada de texto, do mesmo modo como o método `System.out.print`.

# E/S em Arquivos

## Exemplo de armazenamento de registros

- A linguagem Java não impõe nenhuma estrutura a um arquivo, logo o conceito de um registro deve ser definido pelo programador ao estruturar os arquivos que serão manipulados pelo aplicativo.
- No exemplo a seguir, considera-se que o programa obtém do usuário um conjunto de registros bancários, com as seguintes informações:
  - Número da conta, já passado em ordem crescente.
  - Nome do cliente.
  - Sobrenome do cliente.
  - Saldo bancário.

# E/S em Arquivos

## Exemplo: registros bancários

```
import java.util.NoSuchElementException;

// AccountRecord.java
// A classe AccountRecord representa um registro bancário.

public class AccountRecord {
    private int account; // numero da conta
    private String firstName; // primeiro nome
    private String lastName; // sobrenome
    private double balance; // saldo

    // CONSTRUTORES
    public AccountRecord() {
        this(0, "", "", 0.0);
    }

    public AccountRecord(int acct, String first, String last, double bal)
            throws NoSuchElementException {
        setAccount(acct);
        setFirstName(first);
        setLastName(last);
        setBalance(bal);
    }
    /* continua na próxima página */
```

# E/S em Arquivos

## Exemplo: registros bancários

```
/* continua da página anterior */
// METODOS ACESSORES

public void setAccount(int acct) throws NoSuchElementException {
    if (acct < 0) {
        System.err.print("Conta bancária inválida.");
        throw new NoSuchElementException();
    }
    account = acct;
}

public int getAccount() {
    return account;
}

public void setFirstName(String first) throws NoSuchElementException {
    if (! first.matches("[a-zA-Z -]*")) {
        System.err.print("Primeiro nome inválido.");
        throw new NoSuchElementException();
    }
    firstName = first;
}

public String getFirstName() {
    return firstName;
}
/* continua na próxima página */
```

# E/S em Arquivos

## Exemplo: registros bancários

```
    /* continua da página anterior */
    public void setLastName(String last) throws NoSuchElementException {
        if (! last.matches("[a-zA-Z -]*")) {
            System.err.print("Sobrenome inválido.");
            throw new NoSuchElementException();
        }
        lastName = last;
    }

    public String getLastName() {
        return lastName;
    }

    public void setBalance(double bal) {
        balance = bal;
    }

    public double getBalance() {
        return balance;
    }

} // fim classe AccountRecord
```

# E/S em Arquivos

## Exemplo: registros bancários

```
// CreateTextFile.java
// A classe CreateTextFile demonstra a escrita de dados em um arquivo texto com a classe Formatter.
import java.io.FileNotFoundException;
import java.util.Formatter;
import java.util.FormatterClosedException;
import java.util.NoSuchElementException;
import java.util.Scanner;

public class CreateTextFile {
    private Formatter output; // objeto utilizado para a saída de texto no
                              // arquivo

    // método responsável por abrir o arquivo texto
    public void openFile(String filename) throws FileNotFoundException {
        output = new Formatter(filename);
    } // fim método openFile

    // fecha arquivo file
    public void closeFile() {
        if (output != null)
            output.close();
    } // fim método closeFile
    /* continua na próxima página */
```

# E/S em Arquivos

## Exemplo: registros bancários

```
/* continua da página anterior */
// adiciona registros bancários em um arquivo texto
public void addRecords() throws FormatterClosedException {
    // objeto contendo um registro bancário
    AccountRecord record = new AccountRecord();

    Scanner input = new Scanner(System.in);
    System.out.printf("%s\n", "Para encerrar, digite <ctrl>-z e Enter");
    System.out.printf("%s\n%s","Entre com o número da conta, primeiro nome, sobrenome e saldo.","> ");

    while (input.hasNext()) // repete até encontrar o caractere EOF
    {
        try {
            // coleta os dados do terminal
            record.setAccount(input.nextInt());  // recebe conta bancária
            record.setFirstName(input.next());  // recebe primeiro nome
            record.setLastName(input.next());  // recebe sobrenome
            record.setBalance(input.nextDouble());  // recebe saldo

            // imprime o registro no arquivo texto
            output.format("%d %s %s %.2f\n", record.getAccount(),
                    record.getFirstName(), record.getLastName(),
                    record.getBalance());
        } catch (NoSuchElementException elementException) {
            System.err.println(" Tente novamente.");
            input.nextLine();  // descarta a entrada para que usuário tente novamente
        } // fim try .. catch
        System.out.printf("%s\n%s","Entre com o número da conta, primeiro nome, sobrenome e saldo.","> ");
    } // fim while
} // fim método addRecords
/* continua na próxima página */
```

# E/S em Arquivos

## Exemplo: registros bancários

```
    /* continua da página anterior */
    public static void main(String args[]) {

        CreateTextFile application = new CreateTextFile();
        String filename = "clients.txt";
        // abre um arquivo, imprime registros nesse arquivo e depois o fecha
        try {
            application.openFile(filename);
            application.addRecords();
        } catch (FileNotFoundException filesNotFoundException) {
            System.err.println("Erro ao criar o arquivo: " + filename);
        } catch (FormatterClosedException formatterClosedException) {
            System.err
                    .println("Tentativa de escrita em um arquivo já fechado: "
                            + filename);
        } finally {
            application.closeFile();
        } // fim try..catch..finally

    } // fim main
} // fim classe CreateTextFile
```

# E/S em Arquivos

## Exemplo: registros bancários

```
// ReadTextFile.java
// A classe ReadTextFile lê registros bancários de um arquivo texto e imprime na tela
import java.io.File;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.util.NoSuchElementException;
import java.util.Scanner;

public class ReadTextFile {
    private Scanner input;

    // configura arquivo de entrada
    public void openFile(String filename) throws FileNotFoundException {

        input = new Scanner(new File(filename));

    } // end method openFile

    // fecha um arquivo
    public void closeFile() {
        if (input != null)
            input.close();
    } // fim método closeFile
    /* continua na próxima página */
```

# E/S em Arquivos

## Exemplo: registros bancários

```
/* continua da página anterior */
// lê de um arquivo os registros bancários
public void readRecords() throws IOException {
    // instanciando objeto para armazenar a informação coletada
    AccountRecord record = new AccountRecord();

    System.out.printf("%-20s%-20s%-15s%10s\n", "Número da conta",
        "Primeiro Nome", "Sobrenome", "Saldo");

    // coleta todos os registros bancários do arquivo de entrada
    while (input.hasNext()) {
        try {
            // obtendo dados do arquivo e armazenando no objeto
            // AccountRecord
            record.setAccount(input.nextInt());  // recebe número da conta
            record.setFirstName(input.next());  // recebe primeiro nome
            record.setLastName(input.next());  // recebe sobrenome
            record.setBalance(input.nextDouble());  // recebe saldo

            // imprime o registro no arquivo texto
            System.out.format("%-20s%-20s%-15s%10.2f\n",
                record.getAccount(), record.getFirstName(),
                record.getLastName(), record.getBalance());

        } catch (NoSuchElementException elementException) {
            System.err.println(" Registro bancário inválido.");
            throw new IOException(elementException);
        } // fim try..catch
    } // fim while
} // fim método readRecords
/* continua na próxima página */
```

# E/S em Arquivos

## Exemplo: registros bancários

```
    /* continua da página anterior */
    public static void main(String args[]) {
        ReadTextFile application = new ReadTextFile();
        String filename = "clients.txt";

        // abre um arquivo, lê os registros, imprime os registros na tela e
        // depois fecha o arquivo de entrada.
        try {
            application.openFile(filename);
            application.readRecords();
        } catch (FileNotFoundException filesNotFoundException) {
            System.err.println("Erro ao abrir o arquivo.");
        } catch (IOException ioexception) {
            System.err.println("Erro ao ler o arquivo.");
        } finally {
            application.closeFile();
        } // fim try..catch

    } // fim main
} // fim classe ReadTextFile
```

# Serialização de Objetos

## Armazenando objetos em disco

- A linguagem Java fornece um mecanismo denominado serialização (*serialization*) que permite armazenar o estado de um objeto em memória permanente (arquivo).
- Um objeto serializado consiste em uma sequencia de bytes que representam o tipo do objeto, os tipos de dados armazenados e seus valores.
- Ao serializar um objeto, a JVM percorre todo o grafo de objetos no qual o objeto está inserido. Ou seja, todos os objetos referenciados pelo objeto original passam por uma serialização recursiva.
- Uma vez que um objeto serializado foi escrito em um arquivo, ele pode ser lido e desserializado, ou seja, o objeto pode ser recriado na memória principal e usado por um aplicativo.
- Aplicações que se comunicam pela internet, por exemplo, podem trocar objetos pela rede através da serialização.

# Serialização de Objetos

## Classes ObjectInputStream and ObjectOutputStream

- As classes ObjectInputStream e ObjectOutputStream permitem que objetos sejam lidos e escritos a partir de um fluxo de dados como um arquivo.
- Para realizar a serialização e desserialização de objetos com arquivos binários, é possível inicializar objetos ObjectInputStream e ObjectOutputStream com objetos FileInputStream e FileOutputStream.
- Exemplo: Para inicializar um ObjectInputStream com um FileInputStream, basta passar o objeto FileInputStream como argumento para o construtor de ObjectInputStream. Esse processo é denominado empacotamento de um objeto.

# Serialização de Objetos

## Classes serializáveis

- Classes serializáveis em Java requerem a definição de um código numérico denominado `serialVersionUID`. Esse código é utilizado no processo de desserialização para verificar a compatibilidade da classe que está sendo recebida. Ou seja, se o receptor carregou uma classe com código distinto do objeto recebido, então uma exceção `InvalidClassException` é lançada.
- É recomendado que o programador atribua valores próprios de `serialVersionUID` para suas classes, caso contrário o compilador cria um código default baseado em hash e esse código é dependente do compilador Java.
- Exemplo de declaração do `serialVersionUID`:

```
private static final long serialVersionUID = -6395589202193223169L;
```

# Serialização de Objetos

## Classes serializáveis

- O exemplo a seguir mostra a serialização e desserialização de objetos do tipo `AccountRecord`.
- Como fluxo de dados, são utilizados arquivos binários para a escrita e leitura dos objetos.
- Para serializar e desserializar um objeto, é necessário que a classe correspondente implemente a interface `Serializable`.
- A interface `Serializable` não possui métodos, pois corresponde somente a um rótulo (*tag interfaces*) que indica para o compilador que o objeto pode ser serializado.
- Para ser serializada, a classe `AccountRecord` pode ser atualizada da seguinte forma:

```
import java.io.Serializable;

public class AccountRecord implements Serializable {

    private static final long serialVersionUID = 302L;

    // restante do corpo da classe sem alterações...

}
```

# Serialização de Objetos

## Exemplo: registros bancários

```
// CreateSequentialFile.java
// A classe CreateSequentialFile serializa objetos em um arquivo utilizando a classe ObjectOutputStream.
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.ObjectOutputStream;
import java.util.NoSuchElementException;
import java.util.Scanner;

public class CreateSequentialFile {
    private ObjectOutputStream output; // imprime o estado de um objeto para um arquivo
    // configurando o arquivo de saida
    public void openFile(String filename) throws IOException {
        try {
            output = new ObjectOutputStream(new FileOutputStream(filename));
        } catch (IOException ioException) {
            System.err.print("Erro ao abrir o arquivo: ");
            throw ioException;
        }
    } // fim método openFile

    // fecha arquivo e termina a aplicação
    public void closeFile() throws IOException {
        try {
            if (output != null)
                output.close();
        } catch (IOException ioException) {
            System.err.print("Erro ao fechar o arquivo: ");
            throw ioException;
        } // fim try .. catch
    } // fim método closeFile
    /* continua na próxima página */
```

# Serialização de Objetos

## Exemplo: registros bancários

```
/* continua da página anterior */
// imprime um registro em um arquivo
public void addRecords() throws IOException {
    AccountRecordSerializable record; // objeto que será serializado
    Scanner input = new Scanner(System.in);
    System.out.printf("%s\n", "Para encerrar, digite <ctrl>-z e Enter");
    System.out.printf("%s\n%s", "Entre com o número da conta, primeiro nome, sobrenome e saldo.", "> ");
    // laço se repete até alcançar um caractere EOF
    while (input.hasNext()) {
        try {
            // obtendo dados do terminal e instanciando um objeto Account
            int accountNumber = input.nextInt(); // recebe número da conta
            String firstName = input.next(); // recebe primeiro nome
            String lastName = input.next(); // recebe sobrenome
            double balance = input.nextDouble(); // recebe saldo

            record = new AccountRecordSerializable(accountNumber,
                    firstName, lastName, balance);
            output.writeObject(record); // imprime objeto no arquivo
        } catch (IOException ioException) {
            System.err.print("Erro ao escrever no arquivo: ");
            throw ioException;
        } catch (NoSuchElementException elementException) {
            System.err.println(" Tente novamente.");
            input.nextLine(); // descarta a entrada para que usuário tente novamente
        } // fim try .. catch
        System.out.printf("%s\n%s","Entre com o número da conta, primeiro nome, sobrenome e saldo.", "> ");
    } // fim while
} // fim método addRecords
/* continua na próxima página */
```

# Serialização de Objetos

## Exemplo: registros bancários

```
    /* continua da página anterior */
    public static void main(String args[]) {
        CreateSequentialFile application = new CreateSequentialFile();
        String filename = "clients.bin";

        try {
            application.openFile(filename);
            application.addRecords();
        } catch (IOException e) {
            System.err.println(filename);
            e.printStackTrace();
        } finally {
            // o bloco try..catch abaixo é exclusivo para o fechamento do arquivo
            try {
                application.closeFile();
            } catch (IOException e) {
                System.err.println(filename);
                e.printStackTrace();
            } // fim try..catch
        } // fim try..catch.. finally

    } // fim main
} // fim classe CreateSequentialFile
```

# Serialização de Objetos

## Exemplo: registros bancários

```
// ReadSequentialFile.java
// A classe ReadSequentialFile desserializa objetos de registros bancários de um arquivo.
import java.io.EOFException;
import java.io.FileInputStream;
import java.io.IOException;
import java.io.InvalidClassException;
import java.io.ObjectInputStream;

public class ReadSequentialFile {
    private ObjectInputStream input;
    // configura arquivo de entrada
    public void openFile(String filename) throws IOException {
        try {
            input = new ObjectInputStream(new FileInputStream(filename));
        } catch (IOException ioException) {
            System.err.println("Erro ao abrir o arquivo: ");
            throw ioException;
        }
    } // fim método openFile

    // fecha arquivo e termina aplicação
    public void closeFile() throws IOException {
        try {
            if (input != null)
                input.close();
        } catch (IOException ioException) {
            System.err.println("Erro ao fechar arquivo: ");
            throw ioException;
        } // fim try .. catch
    } // fim método closeFile
    /* continua na próxima página */
```

# Serialização de Objetos

## Exemplo: registros bancários

```
/* continua da página anterior */
// lê de um arquivo os registros bancários, instanciando objetos para armazenar a informação coletada
public void readRecords() throws IOException {
    AccountRecordSerializable record;
    System.out.printf("%-20s%-20s%-15s%10s\n",
        "Número da conta", "Primeiro Nome", "Sobrenome", "Saldo");

    // coleta todos os registros bancários do arquivo de entrada
    try {
        while (true) {
            record = (AccountRecordSerializable) input.readObject();
            // imprime na tela os registros coletados
            System.out.printf("%-20s%-20s%-15s%10.2f\n",
                record.getAccount(), record.getFirstName(),
                record.getLastName(), record.getBalance());
        } // fim while
    } catch (EOFException endOfFileException) {
        // essa exceção é utilizada apenas para terminar a leitura do arquivo de entrada
        return;
    } catch (ClassNotFoundException classNotFoundException) {
        System.err.print("Classe incompatível para desserialização.");
        System.exit(1);
    } catch (InvalidClassException invalidClassException) {
        System.err.print("Classe incompatível para desserialização.");
        System.exit(1);
    } catch (IOException ioException) {
        System.err.print("Erro de leitura do arquivo: ");
        throw ioException;
    } // fim try .. catch
} // fim método readRecords
/* continua na próxima página */
```

# Serialização de Objetos

## Exemplo: registros bancários

```
    /* continua da página anterior */
    public static void main(String args[]) {
        ReadSequentialFile application = new ReadSequentialFile();
        String filename = "clients.bin";

        try {
            application.openFile(filename);
            application.readRecords();
        } catch (IOException e) {
            System.err.println(filename);
            e.printStackTrace();
        } finally {
            // o bloco try..catch abaixo é exclusivo para o fechamento do arquivo
            try {
                application.closeFile();
            } catch (IOException e) {
                System.err.println(filename);
                e.printStackTrace();
            } // fim try..catch
        } // fim try..catch.. finally
    } // fim main
} // fim classe ReadSequentialFile
```

# Serialização de Objetos

## Palavra-chave `transient`

- Para uma classe ser serializável, todos seus atributos de instância devem ser serializáveis. Por default, todos os tipos primitivos são serializáveis.
- Tipos referenciados são serializáveis se implementam a interface `Serializable`. Exemplos: Classes empacotadoras, `String` e arrays, são do tipo `Serializable`.
- Ao percorrer o grafo de objetos durante uma serialização, se for encontrado um objeto de uma classe não serializável , uma exceção `NotSerializableException` será lançada.
- Para evitar esse problema, atributos de instância não serializáveis devem ser declarados com a palavra-chave `transient`.
- Atributos `transient` são ignorados no processo de serialização do objeto.

# Serialização de Objetos

## Serialização em heranças

- Subclasses de uma classe serializável também são serializáveis.
- É possível serializar um objeto de uma subclasse, cuja superclasse não é serializável. Nesse caso, vale observar que:
  - A superclasse deve conter um construtor sem argumento que será utilizado para inicializar os atributos de instância da superclasse. Caso contrário, será lançado um erro em tempo de execução.
  - Os atributos de instância da superclasse não são serializados, ou seja, seus valores são perdidos na serialização.
  - Uma superclasse não serializável pode ter seus atributos serializados de forma explícita pela subclasse, ou seja, armazenando os valores desses atributos em um fluxo de dados e em seguida recuperando-os no processo de desserialização.

# Referências

1. Java: Como Programar, Paul Deitel & Heivey Deitel; Pearson; 7a. Ed. (no. chamada IMECC – 05.133 D368j)
2. Data Structures and Algorithms with Object Oriented Design Patterns in Java, Bruno Preiss; (http://www.brpreiss.com/books/opus6/)
3. The Java Tutorials (Oracle) (http://docs.oracle.com/javase/tutorial/)
4. Guia do Usuário UML, Grady Booch et. al.; Campus(1999)
5. Java Pocket Guide - Robert Liguori & Patricia Liguori; O'Reilley, 2008.