Java - Arquivos

BCC 221 - Programação Orientada a Objectos(POO)

Guillermo Cámara-Chávez

Departamento de Computação - UFOP

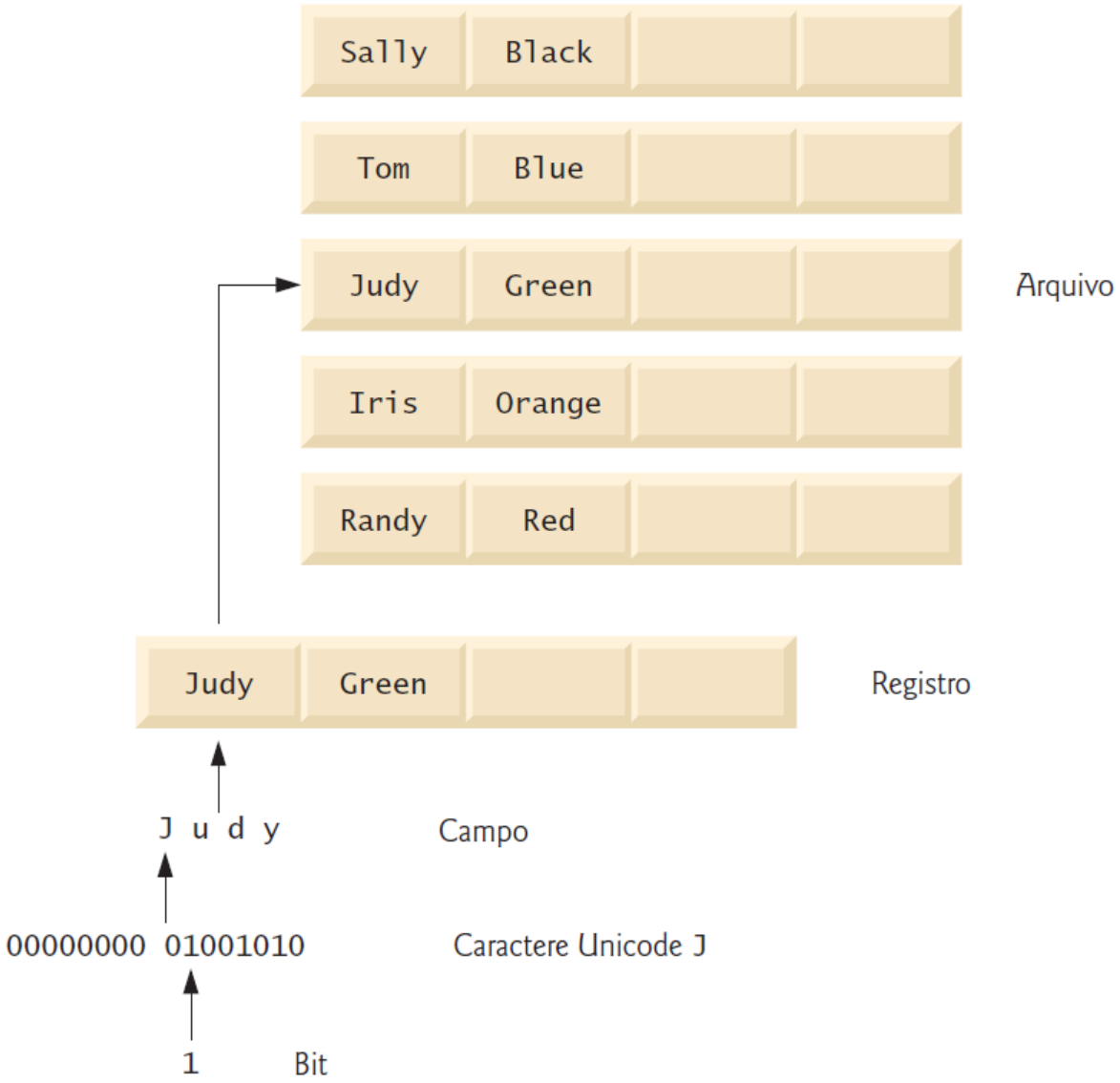
Arquivos

- Nos referimos aos dados armazenados em arquivos como dados **persistentes**
  - Existem além da execução do programa;
  - Armazenados em dispositivos de armazenamento secundário
    - Discos rígidos, discos ópticos, fitas magnéticas, etc.

Arquivos (cont.)

- Itens de dados processados por computadores formam uma hierarquia de dados
  - Tipicamente, vários campos formam um **registro** (uma classe em Java, por exemplo);
  - Um **arquivo** é um grupo de registros relacionados;
  - Para recuperarmos registros específicos de um arquivo, pelo menos um campo é escolhido como **chave do registro**

Arquivos (cont.)



Arquivos (cont.)

- Existem diversas maneiras de organizar registros em um arquivo
  - O mais comum é o **arquivo sequencial**
    - Armazena os registros em ordem de acordo com a chave do registro
- Frequentemente, um grupo de arquivos relacionados é chamado de **base de dados**
  - Controlados por um **sistema gerenciador de bases de dados (SGBD)**

Arquivos (cont.)

- A linguagem Java trata arquivos como fluxos de *bytes* sequenciais
  - O **sistema operacional avisa** quando o **final do fluxo** chegou
  - Em outros casos a indicação de **final de arquivo** é representada por uma **exceção**
  - Ainda, o final de arquivo pode ser indicado por um valor de retorno de um método que processe o fluxo de *bytes*

Arquivos (cont.)

- **Arquivos texto** são criados utilizando–se **fluxos baseados em caracteres**
  - Dados são armazenados como uma sequência de caracteres Unicode
- **Arquivos binários** são criados utilizando-se **fluxos baseados em bytes**
  - Dados são armazenados em formato binário

Arquivos (cont.)

- Um programa Java abre um arquivo através da **criação de um objeto** e da **associação de um fluxo de bytes ou caractere** a ele
  - **FileInputStream**: para leitura de arquivos binários;
  - **FileOutputStream**: para escrita de arquivos binários;
  - **FileReader**: para leitura em arquivos texto;
  - **FileWriter**: para escrita em arquivos texto.
- Além destas classes do pacote **java.io**, também podemos utilizar as classes **Scanner** e **Formatter**

Classe *File*

- A classe File é particularmente útil para **recuperar informações sobre arquivos e diretórios** do sistema de arquivos
  - **Não abrem arquivos** ou **fornecem processamento** de arquivos;
  - São utilizados em conjunto com objetos de outras classes **java.io** para **auxiliar a manipulação de arquivos**.

Classe *File* (cont.)

- A classe possui 4 construtores
  - Localizam arquivos ou diretórios utilizando caminho relativo ou absoluto, ou ainda, um endereço *web*
- A tabela a seguir sumariza os métodos da classe *File*

Classe *File* (cont.)

Método	Descrição
boolean canRead	Retorna *true* se o arquivo puder ser lido pelo programa. Retorna *false* caso contrário.
boolean canWrite	Retorna *true* se o arquivo puder ser escrito pelo programa. Retorna *false* caso contrário.
boolean exists	Retorna *true* se o nome especificado é um arquivo ou diretório. Retorna *false* caso contrário.
boolean isFile	Retorna *true* se o nome especificado é um arquivo Retorna *false* caso contrário.
boolean isDirectory	Retorna *true* se o nome especificado é um diretório. Retorna *false* caso contrário.
boolean isAbsolute	Retorna *true* se o argumento especificado é o caminho absoluto a um arquivo ou diretório. Retorna *false* caso contrário.
String getAbsolutePath	Retorna uma *String* com o caminho absoluto para o arquivo ou diretório.

Classe *File* (cont.)

Diretório	Descrição
String getName	Retorna uma *String* com o nome do arquivo ou diretório.
String getPath	Retorna uma *String* com o caminho do arquivo ou diretório.
String getParent	Retorna uma *String* com o diretório pai do arquivo ou diretório.
long lenght	Retorna uma o comprimento do arquivo, em *Bytes*. Se o objeto representar um diretório, retorna zero.
long lastModified	Retorna uma representação dependente da plataforma do tempo em que o arquivo ou diretório foi modificado pela última vez. **Só é útil para comparação com outros valores também retornados por este método**.
String[] list	Retorna um vetor de *Strings* representando o conteúdo dos diretórios. Retorna *null* se o objeto não representar um diretório.

FileDemonstration.java

```
import java.io.File;
import java.util.Scanner;
public class FileDemonstration {
    public static void  analyzePath(String path){
        File name = new File (path);
        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(), " exists",
              (name.isFile() ? "is a file"
                 : "is not a file" ),
              (name.isDirectory() ? "is a directory"
                 : "is not a directory" ),
              (name.isAbsolute() ? "is absolute path"
                 : "is not absolute path" ),
              "Last modified: ", name.lastModified(),
              "Length: ", name.length(),
              "Path: ", name.getPath(),
              "Absolute path: ",name.getAbsolutePath(),
              "Parent: ", name.getParent());
```

FileDemonstration.java (cont.)

```
            // exibe a listagem do diretorio
            if ( name.isDirectory() ) {
                String directory[] = name.list();
                System.out.println( "\n\nDirectory
                    contents:\n" );
                for (String directoryName : directory)
                    System.out.printf( "%s\n",
                        directoryName );
            }
        }
        else // se nao existir arquivo ou diretorio, exibe a mensagem de
            erro
        {
            System.out.printf( "%s %s", path, "does not
                exist." );
        }
    }
```

FileDemonstration.java (cont.)

```
    public static void main(String[] args) {
        Scanner input = new Scanner( System.in );
        System.out.print( "Enter file or directory name
            : " );
        analyzePath( input.nextLine() );
    }

}
```

FileDemonstration.java (cont.)

```
Enter file or directory name here: C:\Program Files\
    Java
\jdk1.6.0\demo\jfc
jfc exists
is not a file
is a directory
is absolute path
Last modified: 1162570370359
Length: 0
Path: C:\Program Files\Java\jdk1.6.0\demo\jfc
Absolute path: C:\Program Files\Java\jdk1.6.0\demo\jfc
Parent: C:\Program Files\Java\jdk1.6.0\demo
Directory contents:
CodePointIM
FileChooserDemo
Font2DTest
```

FileDemonstration.java (cont.)

```
Enter file or directory name: C:\Program Files\Java\
    jdk1.6.0_11\demo\jfc \Java2D\README.txt
README.txt exists
is a file
is not a directory
is absolute path
Last modified: 1228404384270
Length: 7518
Path: E:\Program Files\Java\jdk1.6.0_11\demo\jfc\Java2D
    \README.txt
Absolute path: E:\Program Files\Java\jdk1.6.0_11\demo\
    jfc\Java2D\README.txt
Parent: E:\Program Files\Java\jdk1.6.0_11\demo\jfc\
    Java2D
```

Arquivos de Texto de Acesso Sequencial

- A classe descrita a seguir encapsula as informações sobre clientes de um banco
  - Número da conta, nome e sobrenome e saldo;
  - Com respectivos *getters* e *setters*;
  - Estas informações formam o registro de cada cliente

AccountRecord.java

```
public class AccountRecord {
    private int account;
    private String firstName;
    private String lastName;
    private double balance;

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

    public AccountRecord(int account, String firstName,
        String lastName, double balance) {
        setAccount(account);
        setFirstName(firstName);
        setLastName(lastName);
        setBalance(balance);
    }
```

AccountRecord.java (cont.)

```
public int getAccount() {
    return account;
}

public void setAccount(int account) {
    this.account = account;
}

public String getFirstName() {
    return firstName;
}

public void setFirstName(String firstName) {
    this.firstName = firstName;
}

public String getLastName() {
    return lastName;
}
```

AccountRecord.java (cont.)

```
    public void setLastName(String lastName) {
        this.lastName = lastName;
    }

    public double getBalance() {
        return balance;
    }

    public void setBalance(double balance) {
        this.balance = balance;
    }
}
```

Arquivos Texto de Acesso Sequencial

- O programa a seguir cria um arquivo texto de acesso sequencial simples
  - Os dados sobre o cliente do banco são obtidos e inseridos no arquivo.

CreateTestFile.java

```
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 gerar saída
        de texto no arquivo
    public void openFile(){
        try{ // append
            //output = new Formatter(FileWriter(çlients.txt",true));
            output = new Formatter("clients.txt");
        }
        catch(SecurityException e){
            System.err.println("You do not have write
                access to this file");
            System.exit(1); // termina o programa
        }
```

CreateTestFile.java (cont.)

```
        catch(FileNotFoundException e){
            System.err.println("Error creating file");
            System.exit(1); // termina o programa
        }
    }
    // adiciona registros ao arquivo
    public void addRecords(){
        AccountRecord record = new AccountRecord();
        Scanner input = new Scanner(System.in);
        System.out.printf("%s\n%s\n%s\n%s\n\n",
          "To terminate input, type the end-of-file
              indicator",
          "when you are prompted to enter input",
          "On UNIX/Linux/Mac OS X type <ctrl> d then
              press Enter",
          "On Windows type <ctrl> z then press Enter");

        System.out.printf("%s\n%s",
           "Enter account number (> 0), first name,
               last name and balance.", "? ");
```

CreateTestFile.java (cont.)

```
// faz um loop até o indicador de fim de arquivo
while (input.hasNext()){
    try{ // gera saída dos valores para o arquivo
        record.setAccount(input.nextInt());
        record.setFirstName(input.next());
        record.setLastName(input.next());
        record.setBalance(input.nextDouble());

        if (record.getAccount() > 0){
            //grave um novo registro
            output.format("%d %s %s %.2f\n",
                record.getAccount(),
                record.getFirstName(),
                record.getLastName(),
                record.getBalance());
        }
        else{
            System.out.println("Account number
                must be greater than 0");
        }
    }
```

CreateTestFile.java (cont.)

```
catch(FormatterClosedException e){
    System.err.println("Error writing to
        file");
    return;
}
catch(NoSuchElementException e){
    System.err.println("Invalid input.
        Please try again.");
    input.nextLine(); // descarta a entrada para que
        o usuário possa tentar novamente
}
```

CreateTestFile.java (cont.)

```
            System.out.printf("%s\n%s",
             "Enter account number (> 0), first name,
                 last name and balance.", "? ");
        }
    }

    public void closeFile(){
        if (output != null){
            output.close();
        }
    }
}
```

Arquivos Texto de Acesso Sequencial

- Um objeto Formatter é utilizado para gravar os dados no arquivo
  - Caso o caminho até o arquivo não seja fornecido, é assumido que o arquivo esteja na mesma pasta em que o programa está;
  - Se o arquivo não existir, será criado
    - Se não puder ser criado, a exceção **FileNotFoundException** é lançada.
  - Se um arquivo existente for aberto, ele será truncado!
    - Todo o conteúdo é descartado.

Arquivos Texto de Acesso Sequencial (cont.)

- Caso não seja permitido escrever o no arquivo, a exceção **SecurityException** é lançada;
- O método **format** escreve no arquivo
  - Com uma formatação semelhante ao *System.out.printf*
- O método **close** fecha o arquivo
  - Se o arquivo não for fechado explicitamente, o sistema operacional o fechará.

CreateTextFileTest.java

```
public class CreateTextFileTest {

    public static void main(String[] args) {
        CreateTextFile app = new CreateTextFile();

        app.openFile();
        app.addRecords();
        app.closeFile();
    }
}
```

CreateTextFileTest.java (cont.)

```
To terminate input, type the end-of-file indicator
when you are prompted to enter input.
On UNIX/Linux/Mac OS X type <ctrl> d then press Enter
On Windows type <ctrl> z then press Enter
Enter account number (> 0), first name, last name and
    balance.
? 100 Bob Jones 24.98
Enter account number (> 0), first name, last name and
    balance.
? 200 Steve Doe -345.67
Enter account number (> 0), first name, last name and
    balance.
? 300 Pam White 0.00
Enter account number (> 0), first name, last name and
    balance.
? 400 Sam Stone -42.16
Enter account number (> 0), first name, last name and
    balance.
? 500 Sue Rich 224.62
Enter account number (> 0), first name, last name and
    balance.
? ^Z
```

Leitura de Arquivos de Texto de Acesso Sequencial

- No exemplo a seguir, o arquivo texto de acesso sequencial criado no exemplo anterior será lido
  - A classe Scanner pode ser utilizada para ler do arquivo, assim como lê do teclado

ReadTextFile.java

```
import java.io.File;
import java.io.FileNotFoundException;
import java.util.NoSuchElementException;
import java.util.Scanner;

public class ReadTextFile {
    private Scanner input;

    // permite ao usuário abrir o arquivo
    public void openFile(){
        try{
            input=new Scanner(new File("clients.txt"));
        }
        catch(FileNotFoundException e){
            System.err.println("Error opening file");
            System.exit(1);
        }
    }
```

ReadTextFile.java (cont.)

```
// lê o registro no arquivo
public void readRecords(){
    AccountRecord record = new AccountRecord();
    System.out.printf( "%-10s%-12s%-12s%10s\n",
    "Account","First Name","Last Name","Balance");
    try{
        while(input.hasNext()){
            record.setAccount(input.nextInt());
            record.setFirstName(input.next());
            record.setLastName(input.next());
            record.setBalance(input.nextDouble());
            // exibe o conteúdo de registro
            System.out.printf( "%-10d%-12s%-12s
                %10.2f\n", record.getAccount(),
                record.getFirstName(),
                record.getLastName(),
                record.getBalance() );
        }
    }
```

ReadTextFile.java (cont.)

```
            //Se as informações no arquivo não estiverem adequadamente
                formatadas
            catch(NoSuchElementException e){
                System.err.println("File improperly formed"
                    );
                input.close();
                System.exit(1);
            }
            //Se Scanner foi fechado antes de os dados serem inseridos
            catch(IllegalStateException e){
                System.err.println("Error reading from file
                    ");
                System.exit(1);
            }
        }
        public void closeFile(){
            if (input != null)
                input.close();
        }
    }
```

Arquivos Texto de Acesso Sequencial

- No construtor do objeto *Scanner*, enviamos um objeto *File* como argumento
  - Especifica que o objeto *Scanner* lerá a partir do arquivo *clients.txt*, no mesmo diretório do programa.
- Para lermos, utilizamos os métodos *nextInt*, *next* e *nextDouble*
  - Se o dado lido é incorreto, a exceção **NoSuchElementException** é lançada
  - O método *hasNext* é utilizado para determinar o final do arquivo

Arquivos Texto de Acesso Sequencial (cont.)

- Se o objeto Scanner é fechado antes da leitura, a exceção **IllegalStateException** é lançada
  - Para fecharmos o *Scanner* utilizamos o método *close*

ReadTextFileTest.java

```
public class ReadTextFileTest {

    public static void main(String[] args) {
        ReadTextFile application = new ReadTextFile();
        application.openFile();
        application.readRecords();
        application.closeFile();
    }

}
```

Saída

Account	First Name	Last Name	Balance
100	Bob	Jones	24.98
200	Steve	Doe	−345.67
300	Pam	White	0.00
400	Sam	Stone	−42.16
500	Sue	Rich	224.62

Arquivos Texto de Acesso Sequencial

- Registros em arquivos de acesso sequencial normalmente não são atualizados em um local específico
  - O tamanho dos campos dos registros pode ser variável, o que corromperia o arquivo original
- Por segurança, é necessário reescrever todo o conteúdo do arquivo

Serialização de Objetos

- Frequentemente, não sabemos exatamente como os dados estão armazenados em um arquivo
  - Nestes casos, é útil ler/escrever um objeto inteiro em um arquivo
  - Java fornece tal mecanismo, chamado de **serialização de objetos**

Serialização de Objetos (cont.)

- Um **objeto serializável** é um objeto representado como uma sequência de *bytes* que incluem os dados do objeto
  - Bem como informação sobre o tipo do objeto e dos dados contidos nele
  - Uma vez escrito em um arquivo, o objeto pode ser lido e **deserializado**
    - Ou seja, recriado na memória

Serialização de Objetos (cont.)

- As classes **ObjectInputStream** e **ObjectOutputStream** permitem que objetos inteiros sejam lidos a partir ou escritos em um fluxo
  - Para utilizar a serialização com arquivos inicializamos objetos destas classes com objetos de fluxos relacionados a arquivos;
  - **FileInputStream** para leitura;
  - **FileOutPutStream** para escrita

Serialização de Objetos (cont.)

- O método **writeObject** recebe como argumento um *Object* que implemente a interface **Serializable** e o escreve em um **OutputStream**
- Correspondentemente, o método **readObject** lê e retorna uma referência a um *Object* a partir de um **InputStream**
  - Depois de lido, pode ser feito um *cast* para o tipo adequado do objeto.
- A serialização de objetos é realizada com fluxos baseados em *bytes*
  - Logo, os arquivos são binários

Serialização de Objetos (cont.)

- O exemplo a seguir demonstra a criação e escrita de um arquivo de acesso sequencial
- Para escrevermos os objetos da classe *AccountRecord*, precisamos alterar nossa classe
  - Os objetos precisam ser serializáveis.

Serialização de Objetos (cont.)

- A interface **Serializable** precisa ser implementada
  - É uma **tagging interface**, não possui métodos;
  - Simplesmente "marca" uma classe como serializável;
  - Qualquer atributo que não seja serializável deve ser declarado como **transient**
  - Tipos primitivos e vetores são serializáveis por padrão

Serialização de Objetos (cont.)

- Desta forma, criamos a classe **AccountRecordSerializable** a partir da classe *AccountRecord*, alterando apenas o cabeçalho da classe:

```
public class AccountRecordSerializable implements
    Serializable
```

- Todos os atributos e métodos são mantidos como na classe original

- Objetos desta classe serão escritos no exemplo a seguir

CreateSequentialFile.java

```
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.ObjectOutputStream;
import java.util.NoSuchElementException;
import java.util.Scanner;
public class CreateSequentialFile
{
   //escreve os dados no arquivo
   private ObjectOutputStream output;
   // permite que o nome do arquivo seja especificado
   public void openFile() {
      try {
         output = new ObjectOutputStream(new
             FileOutputStream("clients.ser"));
      }
      catch ( IOException ioException ){
         System.err.println( "Error opening file.");
      }
   }
```

CreateSequentialFile.java (cont.)

```
// adiciona os registros ao arquivo
public void addRecords() {
  // objeto a ser escrito no arquivo
  AccountRecordSerializable record;
  int accountNumber = 0; // campo numero da conta
  String firstName; // campo nome para o objeto
  String lastName; // campo sobrenome para o objeto
  double balance; // campo saldo para o objeto
  Scanner input = new Scanner( System.in );
  System.out.printf( "%s\n%s", "Enter account number
      (> 0), first name, last
          name and balance.", "? " );
  // repete ate o indicador de final de arquivo
  while ( input.hasNext() ) {
    try {
      accountNumber = input.nextInt();
      firstName = input.next(); // le o nome
      lastName = input.next(); // le o sobrenome
      balance = input.nextDouble(); // le o sal
```

CreateSequentialFile.java (cont.)

```
      if ( accountNumber > 0 ) {
         // cria um novo registro
         record = new AccountRecordSerializable(
            accountNumber, firstName, lastName,
            balance );
         // escreve o objeto no arquivo
         output.writeObject( record );
      }
      else {
         System.out.println("Account number must be
            greater than 0." );
      }
   }
   catch ( IOException ioException ) {
      System.err.println( "Error writing to file." );
      return;
   }
```

CreateSequentialFile.java (cont.)

```
      catch (NoSuchElementException elementException) {
        System.err.println( "Invalid input. Please try
            again." );
        input.nextLine();
      }
      System.out.printf( "%s %s\n%s", "Enter account
          number (>0),", "first name, last name and
          balance.", "? " );
    }
  }
```

CreateSequentialFile.java (cont.)

```
    public void closeFile() {
      try {
          if ( output != null )
          output.close();
      }
      catch ( IOException ioException ) {
        System.err.println( "Error closing file." );
        System.exit( 1 );
      }
    }
}
```

Serialização de Objetos

- O arquivo é aberto na criação de um objeto **FileOutputStream**
  - A string passada para o construtor indica o caminho e o nome do arquivo
  - Uma versão sobrecarregada recebe um argumento booleano adicional indicado se o conteúdo deve ser adicionado ao final do arquivo.
- A mesma política de criação/truncamento de arquivos texto é utilizada para arquivos binários;

Serialização de Objetos (cont.)

- Esta classe não fornece métodos para escrevermos objetos para arquivos
  - Por isto, "embrulhamos" **FileOutputStream** em **ObjectOutputStream**
  - Esta última possui o método **writeObject**,que escreve um objeto serializável inteiro em um **OutputStream**
- O método *close* fecha o arquivo

Leitura: Serialização de Objetos

- O exemplo a seguir lê o arquivo criado no exemplo anterior
  - Um objeto **FileInputStream** "embrulha" um objeto **ObjectInputStream**, abrindo o arquivo binário para leitura
  - O método **readObject** da classe **ObjectInputStream** é utilizado para ler do arquivo e armazenar em um objeto *AccountRecordSerializable*
    - O valor é retornado como um Object;
    - É necessário realizar um cast para a classe adequada na atribuição.

Leitura: Serialização de Objetos (cont.)

- Caso o programa tente ler além do final do arquivo, a exceção **EOFException** é lançada;
- Se a classe do objeto lido não for encontrada, a exceção **ClassNotFoundException** é lançada

ReadSequentialFile.java

```
import java.io.EOFException;
import java.io.FileInputStream;
import java.io.IOException;
import java.io.ObjectInputStream;

public class ReadSequentialFile {
  private ObjectInputStream input;
  //abre o arquivo
  public void openFile(){
    try{
      input = new ObjectInputStream(new FileInputStream
          ( "clients.ser" ) );
    }
    catch ( IOException ioException ){
      System.err.println("Error opening file.");
    }
  }
```

ReadSequentialFile.java (cont.)

```
// Lê o registro de um arquivo
public void readRecords(){
AccountRecordSerializable record;
  System.out.printf("%-10s%-12s%-12s%10s\n", "Account
      ", "First Name", "Last Name", "Balance");
  try {
    while ( true ){
      record = ( AccountRecordSerializable ) input.
          readObject();
      // exibe o conteudo do arquivo
      System.out.printf("%-10d%-12s%-12s%10.2f\n",
          record.getAccount(),
          record.getFirstName(),
          record.getLastName(),
          record.getBalance());
    }
  }
```

ReadSequentialFile.java (cont.)

```
    //se ocorrer uma tentativa de leitura depois do fim do arquivo
    catch ( EOFException endOfFileException ){
      return;
    }
    //se o arquivo acessado não tiver a classe
    catch ( ClassNotFoundException e ){
      System.err.println( "Unable to create object." );
    }
    catch ( IOException e ) {
      System.err.println( "Error during reading from
          file." );
    }
}
```

ReadSequentialFile.java (cont.)

```
    // fecha o arquivo e termina a aplicacao
    public void closeFile() {
      try {
        if ( input != null )
          input.close();
        }
        catch ( IOException ioException ) {
          System.err.println( "Error closing file." );
          System.exit( 1 );
        }
    }
}
```

Arquivos de Acesso Aleatório

- Arquivos de acesso aleatório podem ser lidos ou escritos em posições arbitrárias
  - Determinadas pelo número do registro, ou simplesmente ao final do arquivo;
  - Atualizáveis.
- A classe **RandomAccessFile** fornece facilidades para criação de tais arquivos
  - Trata os arquivos como um vetor de bytes armazenados no sistema de arquivos;

Arquivos de Acesso Aleatório (cont.)

- Há um tipo de "cursor" (**file pointer**), que pode ser posicionado onde necessário;
- Métodos **getFilePointer** (retorna a posição) e **seek** (define uma nova posição);
- Além de métodos para escrita/leitura

Arquivos de Acesso Aleatório (cont.)

- O construtor recebe como argumentos o objeto File, e o modo de abertura
  - "**r**": leitura;
  - "**rw**": leitura e escrita;
  - "**rws**": leitura e escrita e metadados sincronizados com o dispositivo de armazenamento;
  - "**rwd**": leitura e escrita sincronizados com o dispositivo de armazenamento

```
RandomAccessFile raf = RandomAccessFile(file, "
    rw")
```

Arquivos de Acesso Aleatório (cont.)

- Para a leitura do arquivo, existem diversos métodos
  - *read*, *readByte*, *readFloat*, *readInt*, *readLine*, *readShort*, *readLong* ...
- Analogamente, para escrita no arquivo, há métodos correspondentes
  - *write*, *writeByte*, *writeFloat*, *writeInt*, *writeLine*, *writeShort*, *writeLong* ...

Arquivos de Acesso Aleatório (cont.)

- O método **length** retorna o tamanho do arquivo, em bytes;
- O método **getFilePointer** retorna a posição do cursor em relação ao início do arquivo, em *bytes*
- O método **seek** define uma nova posição para o cursor, medida a partir do início do arquivo, em que ocorrerá a próxima operação de escrita ou leitura;
- O método **close** fecha o arquivo

Record.java

```
public class Record {
    private int id;
    private char itemName[] = new char[80];
    private char description[] = new char[255];
    public static final int SIZE = 674;

    public Record(){
        this(0, "", "");
    }

    public Record(int id, String itemName, String
        description) {
        setId(id);
        setItemName(itemName);
        setDescription(description);
    }
```

Record.java (cont.)

```
public void writeFixedLengthString(char field[],
    String value, int size){
    int length = value.length() < size ? value.
        length() : size-1;
    int i;
    for (i = 0; i < length; i++){
        field[i] = value.charAt(i);
    }
    field[i] = '\0';
}

public int getId() {
    return id;
}

public void setId(int id) {
    this.id = id;
}
```

Record.java (cont.)

```
public String getItemName() {
    return new String(itemName);
}

public void setItemName(String itemName) {
    writeFixedLengthString(this.itemName, itemName,
        80);
}

public String getDescription() {
    return new String(description);
}
```

Record.java (cont.)

```
    public void setDescription(String description) {
        writeFixedLengthString(this.description,
            description, 255);
    }

    public String toString(){
        return String.format("\n%d %s %s", getId(),
            getItemName(), getDescription());
    }
}
```

CollectInfo.java

```
import java.util.Scanner;

public class CollectInfo {
    Scanner in = new Scanner(System.in);
    int id;
    String itemName, description;

    public int getId() {
        return id;
    }

    public void setId(int id) {
        this.id = id;
    }

    public String getItemName() {
        return itemName;
    }
```

CollectInfo.java (cont.)

```
public void setItemName(String itemName) {
    this.itemName = itemName;
}

public String getDescription() {
    return description;
}

public void setDescription(String description) {
    this.description = description;
}
```

CollectInfo.java (cont.)

```
    public void accept(){
        System.out.println("Enter your Id:");
        id = in.nextInt();
        in.nextLine();
        System.out.println("Enter your ItemName");
        itemName = in.nextLine();
        System.out.println("Enter item Description");
        description = in.nextLine();
    }
}
```

RandomAccessFiles.java

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

public class RandomAccessFiles {

    private static void writeRecord(Record data){
        RandomAccessFile file = null;
        try{
            file = new RandomAccessFile(new File("
                record.dat"), "rw");
            file.seek(data.getId() * Record.SIZE);
            file.writeInt(data.getId());
            file.writeUTF(data.getItemName());
            file.writeUTF(data.getDescription());
        }
```

RandomAccessFiles.java (cont.)

```
        catch(FileNotFoundException e){
            System.err.println(e.getMessage());
        }
        catch(IOException e){
            System.err.println(e.getMessage());
        }
        finally{
            if (file != null){
                try{
                    file.close();
                }
                catch(IOException e){
                    System.err.println(e.getMessage());
                }
            }
        }
    }
```

RandomAccessFiles.java (cont.)

```
private static Record readRecord(int sid){
    RandomAccessFile file = null;
    Record data = null;
    try{
        file = new RandomAccessFile(new File("
            record.dat"), "rw");
        file.seek(sid * Record.SIZE);
        int id = file.readInt();
        String itemName = file.readUTF();
        String description = file.readUTF();
        data = new Record(id,itemName,description);
    }
    catch(FileNotFoundException e){
        System.err.println(e.getMessage());
    }
    catch(IOException e){
        System.err.println(e.getMessage());
    }
```

RandomAccessFiles.java (cont.)

```
        finally{
            if (file != null){
                try{
                    file.close();
                }
                catch(IOException e){
                    System.err.println(e.getMessage());
                }
            }
        }
        return data;
    }
```

RandomAccessFiles.java (cont.)

```
    public static void main(String[] args) {
        CollectInfo col = new CollectInfo();
        col.accept();

        Record sue = new Record(col.getId(),
            col.getItemName(),
            col.getDescription());

        writeRecord(sue);

        Record out = readRecord(12);
        System.out.println(out);
    }

}
```

RandomAccessFiles.java (cont.)

```
Enter your Id:
12
Enter your ItemName
Television
Enter item Description
lcd

12 Television lcd
```

Outras formas de Leitura e Escrita de Arquivos

- Podem ser realizadas as operações de leitura/escrita em arquivos de forma semelhante às operações de I/O do *console*
- **File()**: representação de um arquivo
- **PrinterWriter()**: usado para escrever em um arquivo da mesma forma como *System.out*
- **Scanner()**: usado para ler da mesma forma como *System.in*

Outras formas de Leitura e Escrita de Arquivos (cont.)

```
import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.PrintWriter;
import java.util.Scanner;

public class JavaPrintWriter {
    public static void main(String[] args) {
        File file = new File("test.txt");
        // escrita
        try{
            // se o arquivo existe apaga os dados
            //PrintWriter out = new PrintWriter(file);
            // append
            PrintWriter out = new PrintWriter(new
                FileOutputStream(file,true));
            out.println("Mike Bolton");
            out.println(42);
            out.close();
        }
```

Outras formas de Leitura e Escrita de Arquivos (cont.)

```
            catch(IOException e){
                System.err.println(e);
            }
            // leitura
            try{
                Scanner in = new Scanner(file);
                String name = in.nextLine();
                int age = in.nextInt();
                System.out.println(name + " " + age);
                in.close();
            }
            catch(FileNotFoundException e){
                System.err.println(e);
            }
        }

    }
```

Exemplo 2 de Serialização : Student.java

```
import java.io.Serializable;

public class Student implements Serializable{
    private String name;
    private double grade;

    public Student(){
        this("",0.0);
    }
    public Student(String name, double grade) {
        this.name = name;
        this.grade = grade;
    }

    public String getName() {
        return name;
    }
```

Exemplo 2 de Serialização : Student.java (cont.)

```
    public void setName(String name) {
        this.name = name;
    }

    public double getGrade() {
        return grade;
    }

    public void setGrade(double grade) {
        this.grade = grade;
    }

    public String toString(){
        return String.format("\n%s %f", getName(),
            getGrade());
    }
}
```

JavaSerialization.java

```
import java.io.EOFException;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.util.ArrayList;

public class JavaSerialization {

    public static void main(String[] args) throws
        FileNotFoundException, IOException,
        ClassNotFoundException {
         File file = new File("students.dat");
         ArrayList<Student> students =new ArrayList<>();
         students.add(new Student("Tom", 7.8));
         students.add(new Student("Dave", 6.5));
         students.add(new Student("Bill", 5.3));
```

JavaSerialization.java (cont.)

```
// append
//FileOutputStream fo = new FileOutputStream(file,true);
//ObjectOutputStream out = new
    AppendableObjectOutputStream(fo);
//serialize
FileOutputStream fo =
    new FileOutputStream(file);
ObjectOutputStream out =
    new ObjectOutputStream(fo);
for (Student s : students){
    out.writeObject(s);
}
out.close();
fo.close();
// deserialize
FileInputStream fi = new FileInputStream(file);
ObjectInputStream in =
    new ObjectInputStream(fi);
ArrayList<Student> students2 =
    new ArrayList<>();
```

JavaSerialization.java (cont.)

```
            try{
                while(true){
                    Student tmp = (Student)in.readObject();
                    students2.add(tmp);
                }
            }
            catch(EOFException e){

            }
            for (Student s : students2){
                System.out.println(s);
            }
        }
    }
```

JavaSerialization.java (cont.)

```
Tom 7.800000

Dave 6.500000

Bill 5.300000
```

JavaSerialization.java (cont.)

Para permitir adicionar mais registros em um arquivo

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

public class AppendableObjectOutputStream extends
    ObjectOutputStream {

    public AppendableObjectOutputStream(OutputStream
        out) throws IOException {
        super(out);
    }

    @Override
    protected void writeStreamHeader() throws
        IOException{

    }
}
```

Exemplo 3

```
import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.InputStreamReader;
import java.io.OutputStreamWriter;

public class JavaBuffered {

  public static void main(String[] args) {
    try{
      //FileOutputStream arq = new FileOutputStream(.ªrquivo.txt", true);
      FileOutputStream arq =
        new FileOutputStream("arquivo.txt");
      OutputStreamWriter os =
        new OutputStreamWriter(arq);
      BufferedWriter out=new BufferedWriter(os);
```

Exemplo 3 (cont.)

```
        out.write("36 ; 56 ; 45\n");
        out.write("46 ; 66 ; 55\n");
        out.write("56 ; 76 ; 65\n");
        out.close();
        os.close();
    }
    catch(Exception e){
        System.err.println("Erro ao escrever o arquivo");
    }

    try{
        FileInputStream arq =
          new FileInputStream("arquivo.txt");
        InputStreamReader is =
          new InputStreamReader(arq);
        BufferedReader in=new BufferedReader(is);
        String linha;
```

Exemplo 3 (cont.)

```
            do{
               linha = in.readLine();
               if (linha != null){
                  String[] palavras = linha.split(";");
                  System.out.println("\nNova linha ------");
                  for (String s : palavras){
                     System.out.print(s + " ");
                  }
               }
            } while (linha != null);
         }
         catch(Exception e){
            System.err.println("Erro ao escrever o arquivo");
         }
      }
   }
```

Saída

```
Nova linha ---------
36    56    45
Nova linha ---------
46    66    55
Nova linha ---------
56    76    65
```

FIM