Programação 2

Arquivos, fluxos e serialização de objetos

Prof. Domingo Santos

domingos.santos@upe.br

UNIVERSIDADE DE PERNAMBUCO

Resumo

- Enum

- Arquivos
  - texto
  - Serializados

Conteúdo baseado no capítulo 15 do livro de Deitel

Enum

Enum

Uma classe enum em Java é uma forma especial de classe usada para definir um **conjunto fixo de constantes**.

Essas constantes representam valores imutáveis, como os dias da semana, meses do ano, estados de um pedido, etc.

Usar enum torna o código mais legível e menos propenso a erros, limitando os valores que uma variável pode assumir.

Exemplo:

```
public enum DiaDaSemana {
    SEGUNDA, TERCA, QUARTA, QUINTA, SEXTA, SABADO, DOMINGO;
}
```

Enum

Exemplo 1 :

```
public enum DiaDaSemana {
    SEGUNDA,
    TERCA,
    QUARTA,
    QUINTA,
    SEXTA,
    SABADO,
    DOMINGO;

    public boolean isDiaUtil() {
        switch (this) {
            case SABADO:
            case DOMINGO:
                return false;
            default:
                return true;
        }
    }
}
```

```
public class ExemploEnum {
    public static void main(String[] args) {
        DiaDaSemana hoje = DiaDaSemana.QUINTA;

        switch (hoje) {
            case SEGUNDA:
                System.out.println("Hoje é segunda-feira!");
                break;
            case TERCA:
                System.out.println("Hoje é terça-feira!");
                break;
            case QUARTA:
                System.out.println("Hoje é quarta-feira!");
                break;
            case QUINTA:
                System.out.println("Hoje é quinta-feira!");
                break;
            case SEXTA:
                System.out.println("Hoje é sexta-feira!");
                break;
            case SABADO:
                System.out.println("Hoje é sábado!");
                break;
            case DOMINGO:
                System.out.println("Hoje é domingo!");
                break;
        }
    }
}
```

Enum

Exemplo 1:

```
public enum DiaDaSemana {
    SEGUNDA,
    TERCA,
    QUARTA,
    QUINTA,
    SEXTA,
    SABADO,
    DOMINGO;

    public boolean isDiaUtil() {
        switch (this) {
            case SABADO:
            case DOMINGO:
                return false;
            default:
                return true;
        }
    }
}
```

```
public class ExemploEnum {
    public static void main(String[] args) {
        DiaDaSemana hoje = DiaDaSemana.QUINTA;

        System.out.println("Hoje é " + hoje);
        System.out.println("É dia útil? " + hoje.isDiaUtil());
    }
}
```

Enum

Exemplo 2:

```
public enum MenuOption {
// declara o conteúdo do tipo enum
    ZERO_BALANCE(1),
    CREDIT_BALANCE(2),
    DEBIT_BALANCE(3),
    END(4);
    // opção atual de menu
    private final int value;
    // construtor
    private MenuOption(int value) {
        this.value = value;
    }
} // fim do enum de MenuOption
```

```
public class MainEnum {
   private final static MenuOption[] choices = MenuOption.values();
    public static void main(String[] args) {

       int id_option = 2;
       MenuOption option = choices[id_option];

       System.out.println("A opção selecionada é: "+ option);

       switch (option) {
           case ZERO_BALANCE:
               System.out.println("Saldo zero");
               break;
           case CREDIT_BALANCE:
               System.out.println("Crédito");
               break;
           case DEBIT_BALANCE:
               System.out.println("Débito");
               break;
           case END:
               System.out.println("Encerrar");
               break;
       }
   }
}
```

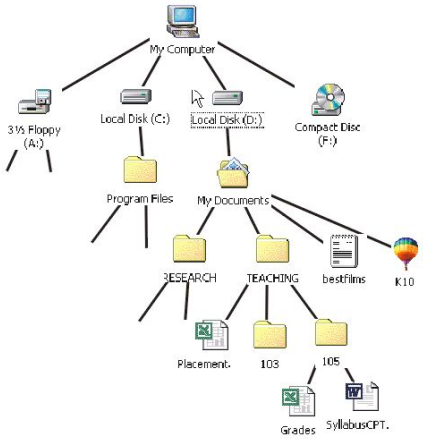
Arquivos

Introdução

Dados armazenados em variáveis e arrays são temporários

Para retenção de longo prazo dos dados, mesmo depois de os programas que os criaram serem fechados, os computadores usam arquivos.

Dados mantidos nos arquivos são **dados persistentes**



Arquivos e fluxos

Fluxo de bytes sequencial

- Fluxos baseados em bytes:
  - geram e insere dados em um formato binário;
  - um char tem dois bytes, um int tem quatro bytes, um double tem oito bytes etc.
  - arquivos binários

- Fluxos baseados em caracteres:
  - geram e inserem dados como uma sequência de caracteres na qual cada caractere tem dois bytes;
  - o número de bytes para determinado valor depende do número de caracteres nesse valor.
  - Por exemplo, o valor 2000000000 requer 20 bytes (10 caracteres a dois bytes por caractere), mas o valor 7 só demanda dois bytes (um caractere a dois bytes por caractere).
  - arquivos de texto

Classes e interfaces NIO

Usando classes e interfaces NIO para obter informações de arquivo e diretório

As interfaces Path e DirectoryStream e as classes Paths e Files (todas do pacote java.nio.file) são úteis para recuperar
informações sobre arquivos e diretórios no disco:

- **Interface Path**: os objetos das classes que implementam essa interface representam o local de um arquivo ou diretório. Objetos Path não abrem arquivos nem fornecem capacidades de processamento deles.

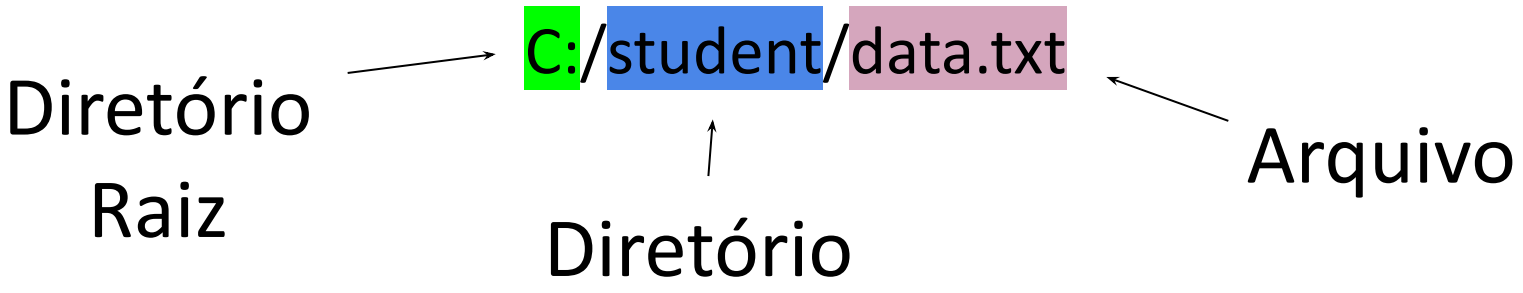
- **Classe Paths**: fornece os métodos static utilizados para obter um objeto Path representando um local de arquivo ou diretório.

- **Classe Files**: oferece os métodos static para manipulações de arquivos e diretórios comuns, como copiar arquivos; criar e excluir arquivos e diretórios; obter informações sobre arquivos e diretórios; ler o conteúdo dos arquivos; obter objetos que permitem manipular o conteúdo de arquivos e diretórios; e mais.

- **Interface DirectoryStream**: os objetos das classes que implementam essa interface possibilita que um programa itere pelo conteúdo de um diretório.

Classes e interfaces NIO

- Um caminho de arquivo ou diretório especifica sua localização em disco
- Um **caminho absoluto** contém todos os diretórios, desde o diretório raiz, que encaminha a um arquivo ou diretório específico
- **Caminho relativo** representa a parte parcial do diretório completo



Obtendo objetos Path de URIs

- Uma versão sobrecarregada do método static Files usa um objeto URI para localizar o arquivo ou diretório.

- Um **Uniform Resource Identifier (URI)** é uma forma mais geral dos Uniform Resource Locators (URLs) que são utilizados para pesquisar sites na web

    - Em Windows:
        - file://C:/data.txt
    - Em linux:
        - file:/home/student/data.txt

Exemplo: obtendo informações de arquivo e diretório

```
import java.io.IOException;
import java.nio.file.DirectoryStream;
import java.nio.file.Files;
import java.nio.file.Path;
import java.nio.file.Paths;
import java.util.Scanner;

public class FileAndDirectoryInfo {

   public static void main(String[] args) throws IOException {
       Scanner input = new Scanner(System.in);
       System.out.println("Enter file or directory name:");

       // cria o objeto Path com base na entrada de usuário
       Path path = Paths.get(input.nextLine());
   } // fim de main
}
```

Path.get(String path): Método static da classe Paths, que representa o local de um arquivo ou diretório em um objeto Path

Exemplo: obtendo informações de arquivo e diretório

```
. . .
public class FileAndDirectoryInfo {

   public static void main(String[] args) throws IOException {
    . . .
       // se o caminho existe, gera uma saída das informações sobre ele
       if (Files.exists(path))
       {
           // exibe informações sobre o arquivo (ou diretório)
           System.out.printf("%n%s exists%n", path.getFileName());
           System.out.printf("%s a directory%n",
                   Files.isDirectory(path) ? "Is" : "Is not");
           System.out.printf("%s an absolute path%n",
                   path.isAbsolute() ? "Is" : "Is not");
           System.out.printf("Last modified: %s%n",
                   Files.getLastModifiedTime(path));
           System.out.printf("Size: %s%n", Files.size(path));
           System.out.printf("Path: %s%n", path);
           System.out.printf("Absolute path: %s%n", path.toAbsolutePath());
       . . .
       } else{
           System.out.printf("%s does not exist%n", path);
       }
   } // fim de main
}
```

Enter file or directory name: ./src/

src exists

Is a directory

Is not an absolute path

Last modified:
2024-05-16T20:56:00.2436366Z
Size: 4096

Path: ./src
Absolute path:
/home/domingossj/eclipse-workspace/HelloWorld/./src

elloWorld/./src/StaticCharMethods.java

Exemplo: obtendo informações de arquivo e diretório

```
. . .
public class FileAndDirectoryInfo {
   public static void main(String[] args) throws IOException {
   . . .
       // se o caminho existe, gera uma saída das informações sobre ele
       if (Files.exists(path))
       {
       . . .
           if (Files.isDirectory(path)) // listagem de diretório de saída
           {
               System.out.printf("%nDirectory contents:%n");
               // objeto para iteração pelo conteúdo de um diretório
               DirectoryStream<Path> directoryStream
                       = Files.newDirectoryStream(path);

               for (Path p : directoryStream) {
                   System.out.println(p);
               }

           }
       } else{
           System.out.printf("%s does not exist%n", path);
       }
   } // fim de main
}
```

```
Enter file or directory name: ./src/

Directory contents:
./src/StringCompare.java
./src/StringMiscellaneous.java
./src/StringConstructors.java
./src/StringMiscellaneous2.java
./src/StringIndexMethods.java
./src/StaticCharMethods.java
./src/StringStartEnd.java
./src/FileAndDirectoryInfo.java
./src/TokenTest.java
./src/StringValueOf.java
./src/com
./src/SubString.java
./src/StringConcatenation.java
```

Caracteres separadores

Utilizado para separar diretórios e arquivos em um caminho

Em um computador Windows, o caractere separador é uma barra invertida (\).

Em um sistema Linux ou Mac OS X, é uma barra (/).

O Java processa esses dois caracteres de maneira idêntica em um nome de caminho.

Por exemplo, se fôssemos utilizar o caminho: C:\Program Files\Java\jdk1.6.0_11\demo/jfc. Ainda funcionaria

Arquivos de texto de acesso sequencial

Salvando informações em Arquivo - parte 1

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

public class CreateTextFile {
   // envia uma saída de texto para um arquivo
   private static Formatter output;

   public static void main(String[] args) {
       openFile();
       addRecords();
       closeFile();
   }
   public static void closeFile() {
       if (output != null) {
           output.close();
       }
   }
  . . .
}
```

Arquivos de texto de acesso sequencial

Salvando informações em Arquivo - parte 2

```
public class CreateTextFile {
   . . .

   public static void openFile() {
       try {
           output = new Formatter("clients.txt"); // abre o arquivo
       } catch (SecurityException securityException) {
           System.err.println("Write permission denied. Terminating.");
           System.exit(1); // termina o programa
       } catch (FileNotFoundException fileNotFoundException) {
           System.err.println("Error opening file. Terminating.");
           System.exit(1); // termina o programa
       }
   }
  . . .
}
```

Arquivos de texto de acesso sequencial

Salvando informações em Arquivo - parte 3

Método addRecords

```
public class CreateTextFile {
   . . .
   public static void addRecords() {
       Scanner input = new Scanner(System.in);
       // faz um loop até o indicador de fim de arquivo
       String loopCondition = "";
       while (!loopCondition.equals("0")) {
           try {


           } catch (NoSuchElementException elementException) {
               System.err.println("Entrada Invalida\n ");
               input.next(); // Clear the invalid input

           }
           System.out.print("Digite 0-para sair, "
                   + "qualquer tecla- para continuar: ");
           loopCondition = input.next();
       } // fim do while
   }
}
```

Conteúdo dentro do Try Catch

```
System.out.printf("\nDigite o No da conta: ");
int nConta = input.nextInt();

System.out.printf("Digite 1o Nome: ");
String nome1 = input.next();

System.out.printf("Digite último Nome: ");
String nome2 = input.next();

System.out.printf("Digite o saldo: ");
double saldo = input.nextDouble();

output.format("%d %s %s %.2f%n", nConta,
       nome1, nome2, saldo);
```

Execução:

Digite o No da conta: 345
Digite 1o Nome: "Domingos"
Digite último Nome: "Santos"
Digite o saldo: 50,6
Digite 0-para sair, qualquer tecla- para continuar: 1

Digite o No da conta: 123
Digite 1o Nome: "Oliveira"
Digite último Nome: "Gomes"
Digite o saldo: 20000,32
Digite 0-para sair, qualquer tecla- para continuar: 1

Digite o No da conta: qwe
Entrada Invalida

Digite 0-para sair, qualquer tecla- para continuar: 0

Arquivos de texto de acesso sequencial

Lendo informações do Arquivo - parte 1

```
import java.io.IOException;
import java.nio.file.Paths;
import java.util.NoSuchElementException;
import java.util.Scanner;

public class ReadTextFile {

   private static Scanner input;

   public static void main(String[] args) {
       openFile();
       readRecords();
       closeFile();
   }

   public static void openFile() {
       try {
           input = new Scanner(Paths.get("clients.txt"));
       } catch (IOException ioException) {
           System.err.println("Error opening file. Terminating.");
           System.exit(1);
       }
   }
}
```

Arquivos de texto de acesso sequencial

Lendo informações do Arquivo - parte 2

```
public class ReadTextFile {
   . . .
   // lê o registro no arquivo
   public static void readRecords() {
       System.out.printf("%-10s%-12s%-12s%10s%n", "Conta",
               "1o Nome", "Útimo Nome", "Saldo");
       try {
           while (input.hasNext()) // enquanto houver mais para ler
           {
               // exibe o conteúdo de registro
               System.out.printf("%-10d%-12s%-12s%10.2f%n", input.nextInt(),
                       input.next(), input.next(), input.nextDouble());
           }
       } catch (NoSuchElementException elementException) {
           System.err.println("File improperly formed. Terminating.");
       }
   } // fim do método readRecords

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

Execução:

Conta	1o Nome	Útimo Nome	Saldo
345	"Domingos"	"Santos"	50,60
123	"Oliveira"	"Gomes"	20000,32

Estudo de caso: um programa de consulta de crédito (pg 517)

Classe MenuOption.java

```
public enum MenuOption {
// declara o conteúdo do tipo enum
   ZERO_BALANCE(1),
   CREDIT_BALANCE(2),
   DEBIT_BALANCE(3),
   END(4);
   private final int value; // opção atual de menu
   // construtor
   private MenuOption(int value) {
       this.value = value;
   }
} // fim do enum de MenuOption
```

Estudo de caso: um programa de consulta de crédito (pg 517)

Classe CreditInquiry.java pt 1

```
import java.io.IOException;
import java.lang.IllegalStateException;
import java.nio.file.Paths;
import java.util.NoSuchElementException;
import java.util.Scanner;
public class CreditInquiry {
   private final static MenuOption[] choices = MenuOption.values();

   public static void main(String[] args) {
      MenuOption accountType = getRequest();// obtém a solicitação do usuário
      while (accountType != MenuOption.END) {
         switch (accountType) {
            case ZERO_BALANCE:
               System.out.printf("%nAccounts with zero balances:%n");
               break;
            case CREDIT_BALANCE:
               System.out.printf("%nAccounts with credit balances:%n");
               break;
            case DEBIT_BALANCE:
               System.out.printf("%nAccounts with debit balances:%n");
               break;
         }
         readRecords(accountType);
         accountType = getRequest(); // obtém a solicitação do usuário
      }
   }
   . . .
}
```

Estudo de caso: um programa de consulta de crédito (pg 517)

Classe CreditInquiry.java pt 2

```
public class CreditInquiry {
   . . .
   private static MenuOption getRequest() {
      int request = 4;
      // exibe opções de solicitação
      System.out.printf("%nEnter request%n%s%n%s%n%s%n%s%n" ,
            " 1 - List accounts with zero balances" ,
            " 2 - List accounts with credit balances" ,
            " 3 - List accounts with debit balances" ,
            " 4 - Terminate program" );
      try {
         Scanner input = new Scanner(System.in);
         do // insere a solicitação de usuário
         {
            System.out.printf("%n? ");
            request = input.nextInt();
         } while ((request < 1) || (request > 4));
      } catch (NoSuchElementException noSuchElementException) {
         System.err.println("Invalid input. Terminating." );
      }
      return choices[request - 1]; // retorna o valor enum da opção
   }
 . . .
}
```

Estudo de caso: um programa de consulta de crédito (pg 517)

Classe CreditInquiry.java pt 3

```
public class CreditInquiry {
   . . .
   private static boolean shouldDisplay(
           MenuOption accountType, double balance) {
       if ((accountType == MenuOption.CREDIT_BALANCE) && (balance < 0)) {
           return true;
       } else if ((accountType == MenuOption.DEBIT_BALANCE) && (balance > 0)) {
           return true;
       } else if ((accountType == MenuOption.ZERO_BALANCE) && (balance == 0)) {
           return true;
       }
       return false;
   }
  . . .
}
```

Estudo de caso: um programa de consulta de crédito (pg 517)

Arquivo utilizado (clients.txt):

345 "Domingos" "Santos" 50,60
123 "Oliveira" "Gomes" 20000,32
444 "Rubens" "Silva" 0

Execução:

```
Enter request
 1 - List accounts with zero balances
 2 - List accounts with credit balances
 3 - List accounts with debit balances
 4 - Terminate program
? 1

Accounts with zero balances:
444     "Rubens"    "Silva"         0,00

Enter request
 1 - List accounts with zero balances
 2 - List accounts with credit balances
 3 - List accounts with debit balances
 4 - Terminate program
? 3

Accounts with debit balances:
345     "Domingos"  "Santos"        50,60
123     "Oliveira"  "Gomes"      20000,32

Enter request
 1 - List accounts with zero balances
 2 - List accounts with credit balances
 3 - List accounts with debit balances
 4 - Terminate program
? 4
```

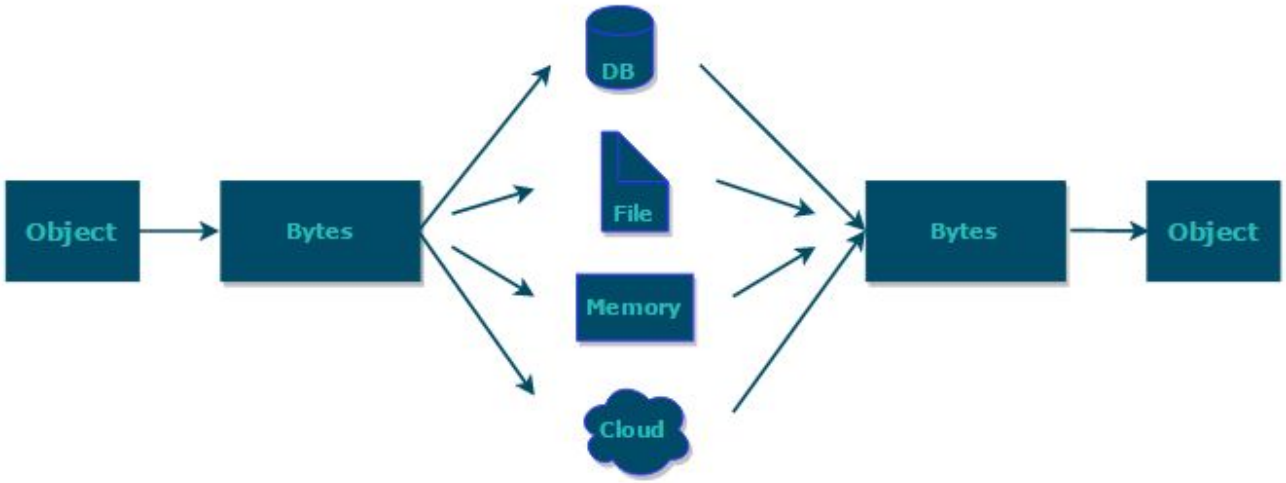
Serialização de objeto

Quando salvando arquivos como texto, informações são perdidas devido que não há como garantir se veio de um int, uma String ou um double

Um objeto serializado é representado como uma sequência de bytes que inclui os dados do objeto, bem como as informações sobre o tipo dele e a natureza dos dados armazenados nele

Depois que um objeto serializado foi gravado em um arquivo, ele pode ser lido a partir do arquivo e desserializado

Classes ObjectInputStream e ObjectOutputStream (pacote java.io) que, respectivamente, implementam as interfaces ObjectInput e ObjectOutput, permitem que objetos inteiros sejam lidos ou gravados em um fluxo



Serialização de objeto

Exemplo: serialização da Classe Account, página 520

```
import java.io.Serializable;

public class Account implements Serializable {

   private int account;
   private String firstName;
   private String lastName;
   private double balance;

   public Account() {
       this(0, "", "", 0.0); // chama outro construtor
   }
   // inicializa uma Account com os valores fornecidos
   public Account(int account, String firstName,
           String lastName, double balance) {
       this.account = account;
       this.firstName = firstName;
       this.lastName = lastName;
       this.balance = balance;
   }
   // Aplicação dos respectivos gets e sets dos atributos
  . . .
}
```

A interface Serializable é uma interface de tags.

Essa interface não contém nenhum método.

Uma classe que implementa Serializable é marcada com tags como um objeto Serializable.

Serialização de objeto

Exemplo: serialização da Classe Account, página 520; Método CreateSequentialFile pt 1

```
import java.io.IOException ;
import java.io.ObjectOutputStream ;
import java.nio.file.Files ;
import java.nio.file.Paths ;
import java.util.NoSuchElementException ;
import java.util.Scanner ;

public class CreateSequentialFile  {
   private static ObjectOutputStream  output; // gera saída dos dados no arquivo
   public static void main(String[] args) {
       openFile();
       addRecords();
       closeFile();
   }
   public static void openFile() {    // abre o arquivo clients.ser
       try {
           output  = new ObjectOutputStream(
                   Files.newOutputStream(Paths.get("clients.ser")));
       } catch (IOException ioException) {
           System.err.println("Error opening file. Terminating." );
           System.exit(1); // termina o programa
       }
   }
}
```

Serialização de objeto

Exemplo: serialização da Classe Account, página 520; Método CreateSequentialFile pt 2

```
public class CreateSequentialFile {
   . . .
   public static void addRecords() {
       Scanner input = new Scanner(System.in);
       String loopCondition = "";
       while (!loopCondition.equals("0")) // faz um loop até o indicador de fim de arquivo
       {
           try {
               System.out.printf("\nDigite o No da conta: "); int nConta = input.nextInt();
               System.out.printf("Digite 1o Nome: "); String nome1 = input.next();
               System.out.printf("Digite último Nome: "); String nome2 = input.next();
               System.out.printf("Digite o saldo: "); double saldo = input.nextDouble();
               Account record = new Account(nConta, nome1, nome2, saldo);
               output.writeObject(record);  // serializa o objeto de registro em um arquivo

           } catch (NoSuchElementException elementException) {
               System.err.println("Invalid input. Please try again.");
               input.nextLine(); // descarta entrada para o usuário tentar de novo
           } catch (IOException ioException) {
               System.err.println("Error writing to file. Terminating.");
               break;
           }
           System.out.print("Digite 0-para sair, qualquer tecla- para continuar: ");
           loopCondition= input.next();
       }
   }
  . . .
}
```

Serialização de objeto

Exemplo: serialização da Classe Account, página 520; Método CreateSequentialFile pt 3

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

no fim da execução cria o objeto serializado chamado: clients.ser

Execução:

Digite o No da conta: 1
Digite 1o Nome: Monkey
Digite último Nome: Luffy
Digite o saldo: 0
Digite 0-para sair, qualquer tecla- para continuar: 1

Digite o No da conta: 2
Digite 1o Nome: Vinsmoke
Digite último Nome: Sanji
Digite o saldo: 200
Digite 0-para sair, qualquer tecla- para continuar: 0

Serialização de objeto

Exemplo: serialização da Classe Account, página 520; Método ReadSequentialFile pt 1

```
import java.io.EOFException;
import java.io.IOException;
import java.io.ObjectInputStream;
import java.nio.file.Files;
import java.nio.file.Paths;

public class ReadSequentialFile {

   private static ObjectInputStream input;

   public static void main(String[] args) {
       openFile();
       readRecords();
       closeFile();
   }
   // permite que o usuário selecione o arquivo a abrir
   public static void openFile() {
       try // abre o arquivo
       {
           input = new ObjectInputStream(
                   Files.newInputStream(Paths.get("clients.ser")));
       } catch (IOException ioException) {
           System.err.println("Error opening file.");
           System.exit(1);
       }
   }
 . . .
}
```

Serialização de objeto

Exemplo: serialização da Classe Account, página 520; Método ReadSequentialFile pt 2

```
public class ReadSequentialFile {
   . . .
   public static void readRecords() {
       System.out.printf("%-10s%-12s%-12s%10s%n", "Account",
               "First Name", "Last Name", "Balance");
       try {
           while (true) // faz um loop até ocorrer uma EOFException
           {
               Account record = (Account) input.readObject();

               // exibe o conteúdo de registro
               System.out.printf("%-10d%-12s%-12s%10.2f%n",
                       record.getAccount(), record.getFirstName(),
                       record.getLastName(), record.getBalance());
           }
       } catch (EOFException endOfFileException) {
           System.out.printf("%no more records%n");
       } catch (ClassNotFoundException classNotFoundException) {
           System.err.println("Invalid object type. Terminating.");
       } catch (IOException ioException) {
           System.err.println("Error reading from file. Terminating.");
       }
   } // fim do método readRecords
  . . .
}
```

Serialização de objeto

Exemplo: serialização da Classe Account, página 520; Método ReadSequentialFile pt 1

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

Leitura do objeto serializado chamado clients.ser:

Account	First Name	Last Name	Balance
1	Monkey	Luffy	0,00
2	Vinsmoke	Sanji	200,00