OO

Persistência

Profa.: Márcia Sampaio Lima

EST - UEA

Persistência

- Característica de um estado sobreviver ao processo que o criou.
- Sem essa capacidade, o estado só existiria na RAM, que é volátil.
- Na prática, consiste em armazenar o estado como dados em memória secundária em um arquivo.

Persistência

- Java é capaz de processar arquivos que requisitam grandes quantidades de dados persistentes.

- Esses são os dados que ficam armazenados dentro dos arquivos, sendo que a sua duração vai além da finalização de um programa.

Tratamento de Erros

- **read() :** Ação de leitura de dados, usada para obter dados de um dispositivo de entrada.

- **write(): Ação executada** para enviar um dado para um dispositivo de saída.

Persistência

- Para trabalhar com entrada e saída de dados são utilizada as classes que estão dentro do **java.io**.
- Essas classes oferecem algumas funcionalidades como:
  - Manipulação de entrada e saída de bytes – transferência de dados binários;
  - Manipulação de entrada e saída de caracteres – transferência de textos;

Persistência

- Para trabalhar com entrada e saída de dados são utilizada as classes que estão dentro do **java.io**.
- Essas classes oferecem algumas funcionalidades como:
  - Buffers - melhoram a eficiência da velocidade de leitura e escrita;
  - Conversão de formatos – texto e formato interno de dados binários.

Persistência

- Classe *FileWriter()*
  - Serve para escrever em arquivos de texto.
  - A classe FileWriter escreve diretamente no arquivo.
  - Exemplo:

```
// cria um novo arquivo
 FileWriter fw = new FileWriter("dados.dat");

//Efetua um append no arquivo
FileWriter fw = new FileWriter("dados.dat",true );
```

Persistência

```
import java.io.FileWriter;
import java.io.IOException;

public class Execucao {

    public static void main(String[] args) {

     try {
            FileWriter fw = new FileWriter("dados.dat", true);

     } catch (IOException e) {
        e.printStackTrace();
     }
  }
}
```

Persistência

```
import java.io.FileWriter;
import java.io.IOException;

public class Execucao {

    public static void main(String[] args) {

     try {
            FileWriter fw = new FileWriter("dados.dat", true);

     } catch (IOException e) {
        e.printStackTrace();
     }
   }
}
```

Cria arquivo dados.dat

Persistência

Importações adequadas

```
import java.io.FileWriter;
import java.io.IOException;

public class Execucao {

    public static void main(String[] args) {

     try {
            FileWriter fw = new FileWriter("dados.dat", true);

     } catch (IOException e) {
        e.printStackTrace();
     }
   }
}
```

Persistência

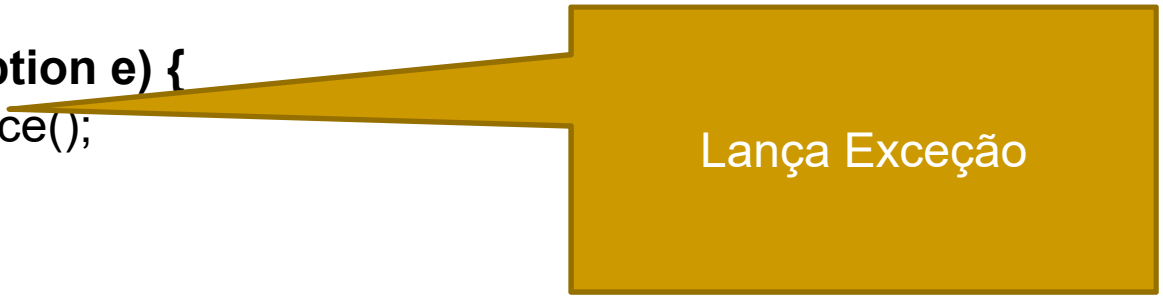
```
import java.io.FileWriter;
import java.io.IOException;

public class Execucao {

    public static void main(String[] args) {

     try {
            FileWriter fw = new FileWriter("dados.dat", true);

     } catch (IOException e) {
        e.printStackTrace();
     }
   }
}
```



Persistência

- A classe BufferedWriter():
  - Utilizada para efetuar a escrita,
  - Bom desempenho,
  - Possui alguns métodos que são independentes de sistema operacional, como quebra de linhas.

Persistência

- A criação do objeto BufferedWriter:

```
//construtor recebe como argumento o objeto do tipo
//FileWriter

BufferedWriter wirter = new BufferedWriter( fw );
```

Persistência

```
import java.io.FileWriter;
import java.io.IOException;

public class Execucao {

    public static void main(String[] args) {

     try {
            FileWriter fw = new FileWriter("dados.dat", true);

            BufferedWriter writer = new BufferedWriter(fw);

     } catch (IOException e) {
        e.printStackTrace();
     }
  }
}
```

Persistência

- Com o bufferedwriter criado, agora é possível escrever conteúdo no arquivo através do método write():

```
//escreve o conteúdo no arquivo
writer.write( "Texto a ser escrito no txt" );

//quebra de linha
writer.newLine();
```

Persistência

```
....
    public static void main(String[] args) {

     try {
            FileWriter fw = new FileWriter("dados.dat", true);

            BufferedWriter writer = new BufferedWriter(fw);
            writer.write("Maria 18 4.5");
            writer.newLine();
            writer.write("Joao 20 7.8");
            writer.newLine();
            writer.write("Jose 23 10.8");
            writer.newLine();

      } catch (IOException e) {
         e.printStackTrace();
      }
   }
}
```

Persistência

- Após escrever tudo que queria, é necessário fechar os buffers e informar ao sistema que o arquivo não está mais sendo utilizado:

```
//fecha os recursos
writer.close();
fw.close();
```

Persistência

```
....
    public static void main(String[] args) {

     try {
            FileWriter fw = new FileWriter("dados.dat", true);

            BufferedWriter writer = new BufferedWriter(fw);
            writer.write("Maria 18 4.5");
            writer.newLine();
            writer.write("Joao 20 7.8");
            writer.newLine();
            writer.write("Jose 23 10.8");
            writer.newLine();

            writer.close();
            fw.close();
.....
```

Persistência

- As classes FileReader() e BufferedReader() servem para ler arquivos em formato texto.

Persistência

- A classe FileReader ():
  - recebe como argumento o objeto File do arquivo a ser lido:

```
//construtor que recebe o objeto do tipo arquivo
  FileReader fr = new FileReader( arquivo );
```

  - A classe BufferedReader, fornece o método readLine() para leitura do arquivo:

```
//construtor que recebe o objeto do tipo
//FileReader

BufferedReader br = new BufferedReader( fr );
```

Persistência

```
.....

String linha;
 try {
        FileReader fr = new FileReader("dados.txt");
        BufferedReader reader = new BufferedReader(fr);

 } catch (FileNotFoundException e) {
     e.printStackTrace();
 }
.....
```

Persistência

- Para ler o arquivo, utilizar:
  - ready(), retorna se o arquivo tem mais linhas a ser lido,
  - readLine(), que retorna a linha atual e passa o buffer para a próxima linha.

```
//enquanto houver mais linhas
while( reader.ready() ){
    //lê a proxima linha
    String linha =  reader.readLine();
    //faz algo com a linha
}
```

```
.....
String linha;
 try {
    FileReader fr = new FileReader("dados.txt");
   BufferedReader reader = new BufferedReader(fr);

    while (reader.ready()){
          linha = reader.readLine();
          System.out.println(linha);
     }

   } catch (FileNotFoundException e) {
          e.printStackTrace();
    } catch (IOException e) {
           e.printStackTrace();
     }

}
```

Persistência

- Após ler tudo que queria, é necessário fechar os buffers e informar ao sistema que o arquivo não está mais sendo utilizado:

```
//fecha os recursos
reader.close();
fr.close();
```

```
.....
String linha;
 try {
    FileReader fr = new FileReader("dados.txt");
   BufferedReader reader = new BufferedReader(fr);

    while (reader.ready()){
          linha = reader.readLine();
          System.out.println(linha);
     }

     reader.close();
     fr.close();

   } catch (FileNotFoundException e) {
          e.printStackTrace();
    } catch (IOException e) {
           e.printStackTrace();
     }

}
```

Persistência

- Classe Aluno()
  - Atributos: nome, idade e nota

```
public class Alunos {
   private String nome;
   private int idade;
   private double media;

public Alunos(String pNome, int pIdade, double pMedia){
    this.nome = pNome;
   this.idade = pIdade;
    this.media = pMedia;
}

@Override
public String toString(){
  return new String ( this.nome + " " + this.idade + " " + this.media);
}
```

- Gravando....

```
public class Execucao {

public static void main(String[] args) {
    Alunos al1 = new Alunos("Maria", 18, 4.5);
    Alunos al2 = new Alunos("Joao", 20, 7.8);
   Alunos al3 = new Alunos("Jose", 23, 10.8);

    try {
        FileWriter fw = new FileWriter("dados.txt", true);
         BufferedWriter writer = new BufferedWriter(fw);

          writer.write(al1.toString());
          writer.newLine();
          writer.write(al2.toString());
          writer.newLine();
          writer.write(al3.toString());

           writer.close();
            fw.close();

} catch (IOException e) {
```

- Lendo....

```
String vet[];
String linha;
try {
FileReader fr = new FileReader("dados.txt");
BufferedReader reader = new BufferedReader(fr);
      while (reader.ready()){
         linha = reader.readLine();
         vet = linha.split(" ");
         al1.setNome(vet[0]);
         al1.setIdade(Integer.parseInt(vet[1]));
         al1.setMedia(Double.parseDouble(vet[2]));

         al1.mostrarDados();
      }
} catch (FileNotFoundException e) {
   e.printStackTrace();
} catch (IOException e) {
   e.printStackTrace();
}
```

Persistência

- **E se tivesse uma lista de Alunos???**

Serialização

- Serialização
  - É um processo no qual a instância de um objeto é transformada em uma seqüência de bytes.
  - É útil quando precisamos enviar objetos pela rede, salvar no disco, ou comunicar de uma JVM com outra.
  - O estado atual do objeto é “congelado” e na outra “ponta” nós podemos “descongelar” este objeto sem perder nenhuma informação.

Serialização

- Para habilitar a **serialização** deve explicitamente implementar a interface Serializable().
- Esta interface é apenas de marcação, pois não tem nenhum método a ser implementado.
  - Serve apenas para que a JVM saiba que aquela determinada Classe está hábil para ser serializada.

Serialização

- Sua única finalidade é anunciar que a classe que está implementando pode ser serializada.
- Ou seja os objetos desse tipo poderão ser salvos através do mecanismo de serialização.
- Se sua superclasse “FOR-UM” tipo serializable, você também será.

Serialização

- Serialização é a técnica que permite transformar o estado de um objeto em uma seqüência bytes.
- Depois que um objeto for serializado ele pode ser gravado (ou persistido) em um arquivo de dados e recuperado do arquivo e desserializado para recriar o objeto na memória.

Serialização

- Exemplo:

```
import java.io.Serializable;

public class Alunos implements Serializable{

private String nome;
private int idade;
private double media;

public Alunos(String pNome, int pIdade, double pMedia){
this.nome = pNome;
this.idade = pIdade;
this.media = pMedia;
}

.....
```

Serializando e Deserializando

- Precisamos de 2 classes:
  - WriteObject () - é responsável por serializar determinada instancia.
  - ReadObject () - é responsável por deserializar determinada instancia

Serializando

- Precisamos de 2 classes:
  - FileOutputStream()
    - é responsável por criar o arquivo fisicamente no disco, assim poderemos realizar a escrita neste.

```
FileOutputStream fout = new FileOutputStream("alunos.ser");
```

  - ObjectOutputStream ()
    - escreve os objetos no FileOutputStream

```
ObjectOutputStream oos = new ObjectOutputStream(fout);
```

Serialização

- São necessários dois fluxos para que algo útil possa ser feito:
  - Para representar a conexão, e
  - Para chamar métodos.

  - FileOutputStream : tem métodos para a gravação de bytes.
  - ObjectOutputStream: tem métodos para gravar objetos.

Serialização

```
public class Execucao {


public static void main(String[] args) {
   Alunos al1 = new Alunos("Maria", 18, 4.5);
   Alunos al2 = new Alunos("Joao", 20, 7.8);
   Alunos al3 = new Alunos("Jose", 23, 10.8);

   try {
      FileOutputStream  fout = new FileOutputStream ("aluno.ser");
      ObjectOutputStream oos = new ObjectOutputStream(fout);

       oos.writeObject(al1);

        oos.close();
   } catch (Exception e) {
         e.printStackTrace();
    }
}
```

Desserializando

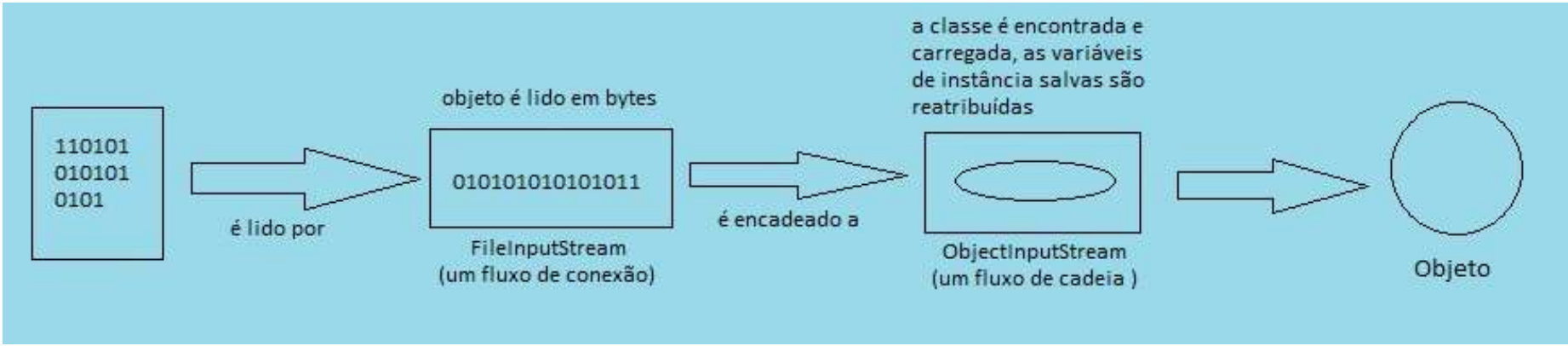
- Quando um objeto é desserializado, a JVM tenta reconstituí-lo criando um novo objeto que tenha o mesmo estado que objeto serializado tinha.

- Exceto pelas variáveis ***transientes***, que são reconstituídas com nulo ou com valores primitivos padrão.

Desserializando

- Se a desserialização for em um local diferente (outro sistema por exemplo) deverá existir a cópia da classe de origem (que foi utilizada para fazer a serialização anteriormente) para que seja feita a desserialização desse objeto que está sendo manipulado.

Desserialização



```
try {
     FileInputStream fin = new FileInputStream("aluno.ser");
     ObjectInputStream ois = new ObjectInputStream(fin);

      al1 = (Alunos) ois.readObject();

      ois.close();

    al1.mostrarDados();


} catch (Exception e) {
    e.printStackTrace();
}
}
```

Serialização e Desserialização

- Objetos do tipo ArrayList??

```
public static void main(String[] args) {

  Alunos al1 = new Alunos("Maria", 18, 4.5);
  Alunos al2 = new Alunos("Joao", 20, 7.8);
  Alunos al3 = new Alunos("Jose", 23, 10.8);

  ArrayList<Alunos> todos = new ArrayList<Alunos>();
  todos.add(al1);
  todos.add(al2);
  todos.add(al3);

  try {
    FileOutputStream  fout = new FileOutputStream ("aluno.ser");
    ObjectOutputStream oos = new ObjectOutputStream(fout);

    oos.writeObject(todos);

    oos.close();
  } catch (Exception e) {
    e.printStackTrace();
  }
```

```
public static void main(String[] args) {

 todos = new ArrayList<Alunos>();

try {
 FileInputStream fin = new FileInputStream("aluno.ser");

 ObjectInputStream ois = new ObjectInputStream(fin);

 todos = (ArrayList<Alunos>) ois.readObject();

 ois.close();

 for(int i=0; i < todos.size(); i++){
      todos.get(i).mostrarDados();
 }

} catch (Exception e) {
     e.printStackTrace();
}

}
```