Streams e Arquivos em Java™

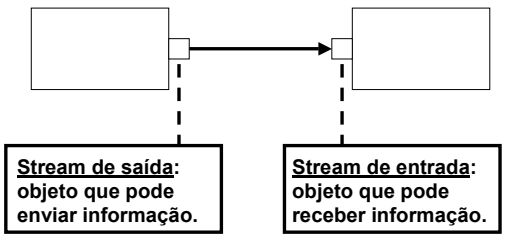
riccioni@sj.univali.br

Sumário

- Streams
  - streams de dados
  - streams de objetos (serialização)
- Arquivos e Persistência
  - implementação manual
  - implementação automática (serialização)

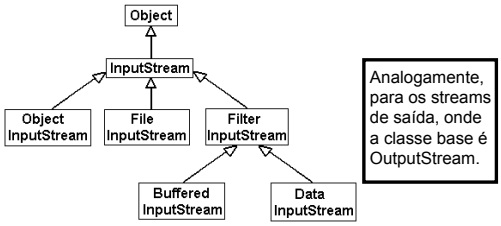
Streams em Java (pacote java.io)

- Extremidades de um canal de comunicação unidirecional (*one-way*).



Streams Orientados a Byte

- Os streams de entrada/saída orientados a byte têm como classe base as classes abstratas InputStream/OutputStream.

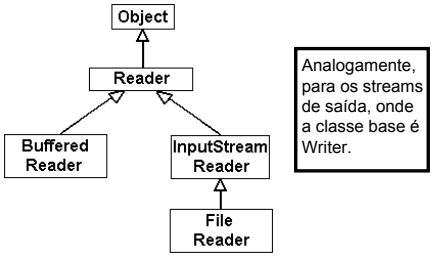


Analogamente, para os streams de saída, onde a classe base é OutputStream.

(Resumo da hierarquia)

Streams Orientados a Caracter

- Implementam entrada/saída de caracteres em formato Unicode (2 bytes por caracter).



Analogamente, para os streams de saída, onde a classe base é Writer.

(Resumo da hierarquia)

Classe Abstrata InputStream

- abstract int read()
  - lê um byte e retorna o byte lido ou -1 quando atinge o final do stream.
- int read(byte[] b)
  - lê um vetor de bytes e retorna o número de bytes lidos ou -1 quando atinge o final do stream.
    (lê, no máximo, b.length bytes)
- int read(byte[] b, int inicio, int quantidade)
  - lê uma quantidade de bytes e armazena no vetor a partir da posição inicial dada.

Classe Abstrata OutputStream

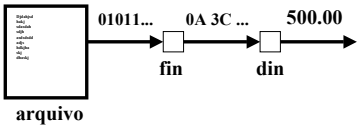
- abstract void write(int b)
  - escreve o byte especificado.
- void write(byte [ ] b)
  - escreve o vetor de bytes especificado.
- void write(byte [ ] b, int inicio, int quantidade)
  - escreve uma quantidade de bytes do vetor a partir da posição inicial especificada.

Combinando Streams

- FileInputStream implementa leitura de bytes a partir de arquivos.
- Outros streams, como DataInputStream, podem ler tipos mais complexos como strings e números, mas não podem ler de arquivos.
- Responsabilidades diferentes são separadas em classes diferentes.
- Podemos combinar streams (um stream é usado no construtor de outro stream) de modo a obter o comportamento desejado.

Combinando Streams (Exemplo)

```
fin = new FileInputStream("arquivo");
din = new DataInputStream(fin);
double salario = din.readDouble();
...
```



Streams de Arquivos

- FileInputStream

  FileInputStream(String nomeArquivo)

  FileInputStream(File arquivo)

- FileOutputStream

  FileOutputStream(String nomeArquivo)
  - caso já exista, o arquivo será apagado automaticamente.

  FileOutputStream(String nomeArquivo, boolean append)
  - caso append == true, o arquivo existente não será apagado e os dados serão adicionados ao final.

Streams com Buffer

- Eficiência em operações de entrada/saída.
- BufferedInputStream

  BufferedInputStream(InputStream outroStream)
  - cria um stream com buffer para o outro stream de entrada.

  BufferedInputStream(InputStream outroStream, int n)
  - permite especificar o tamanho do buffer de n bytes.

- BufferedOutputStream

  BufferedOutputStream(OutputStream outroStream)
  - cria um stream com buffer para o outro stream de saída.

  BufferedOutputStream(OutputStream outro, int n)
  - permite especificar o tamanho do buffer de n bytes.

DataInput e DataOutput

- Interfaces que definem vários métodos de uso geral, para ler números, strings, etc.

- Estes métodos são implementados pelas classes de streams mais utilizadas:

  DataInputStream
  DataOutputStream
  — para entrada/saída de dados.

  ObjectInputStream
  ObjectOutputStream
  — para serialização de objetos.

Interfaces DataInput e DataOutput

- Métodos para ler/escrever números, strings, ...

DataInput

```
boolean readBoolean()
double readDouble()
int readInt()
String readLine()
```

- fim de linha: '\n', '\r', '\r\n' ou EOF

DataOutput

```
void writeBoolean(boolean b)
void writeDouble(double d)
void writeInt(int i)
void writeBytes(String s)
```


Estudo de Caso: Persistência no SisMat

- Em cada classe, vamos implementar como os objetos daquela classe são armazenados e recuperados de um arquivo.
- No SisMat, as classes que precisam implementar persistência são:
  - Pessoa
  - Aluno
  - Disciplina
  - Associacao e
  - Aplicacao

A Interface Persistente

- Precisamos forçar a implementação de persistência nestas classes.
- Então, vamos definir uma interface com os métodos adequados:

```
interface Persistente {
  void salveSe(DataOutputStream saida);
  void carregueSe(BufferedReader entrada);
}
```

- Assim, todo objeto persistente terá métodos para salvar ou recuperar seu estado.

Exemplo para a Classe Pessoa

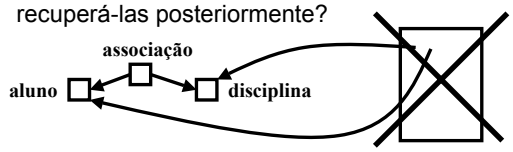
```
class Pessoa implements Persistente {
  private String nome;

  //...

  public void salveSe(DataOutputStream saida)  {
    saida.writeBytes(nome + "\n");
  }

  public void carregueSe(BufferedReader entrada)  {
    nome = entrada.readLine();
  }
}
```


Exemplo para a Subclasse Aluno

```
class Aluno extends Pessoa {
  private String matricula;

  //...

  public void salveSe(DataOutputStream saida) {
    super.salveSe(saida);
    saida.writeBytes(matricula + "\n");
  }

  public void carregueSe(BufferedReader entrada) {
    super.carregueSe(entrada);
    matricula = entrada.readLine();
  }
}
```


Como salvar Referências?

- Podemos implementar persistência facilmente nas classes Aluno e Disciplina, pois não há referência a outros objetos.
- Mas a classe Associacao representa que um objetoA (aluno) está associado a outro objetoB (disciplina).
- Como podemos armazenar as referências e recuperá-las posteriormente?



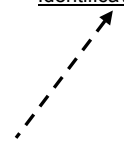
Como Referenciar um Objeto?

- Uma forma de implementar é armazenar a chave de cada objeto referenciado:
  - matrícula do aluno e
  - código da disciplina.
- Quando um objeto (associação) faz referência a outro objeto (aluno ou disciplina), armazenamos a chave do objeto referenciado.
- Portanto, vamos definir uma interface:

```
interface IdentificavelPorChave {
  String retorneChave();
}
```


Implementação da Classe Pessoa

```
class Pessoa implements Persistente,
                        IdentificavelPorChave {
  private String nome;

  //...salveSe...
  //...carregueSe...

  public String retorneChave() {
    return nome;
  }
}
```



Implementação da Classe Aluno

- A chave de um aluno é o seu número de matrícula (redefine o método herdado):

```
class Aluno extends Pessoa {
  private String matricula;

  //...salveSe...
  //...carregueSe...

  public String retorneChave()     {
    return matricula;
  }
}
```

Implementação da Classe Disciplina

```
class Disciplina implements Persistente,
                            IdentificavelPorChave {
  private Integer codigo;
  private String nome;
  //...salveSe...
  //...carregueSe...

  public String retorneChave()     {
    return codigo.toString();
  }
}
```

Chave string simplifica a implementação.

A Classe Associação com Persistência

```
public class Associacao implements Persistente {
   private Object objetoA;
   private Object objetoB;

   public void salveSe(DataOutputStream saida) {
     // 1. salva o nome da classe do objetoA
     // 2. salva a chave do objetoA

     // 3. salva o nome da classe do objetoB
     // 4. salva a chave do objetoB
   }
}
```

- Importante: como a chave é String, não precisamos salvar o nome da classe da chave de cada objeto!

A Classe Associação com Persistência

```
public class Associacao implements Persistente {
   private Object objetoA;
   private Object objetoB;

   public void salveSe(DataOutputStream saida) {
     String a, b;
     saida.writeBytes(objetoA.getClass().getName() +"\n");
     a = ((IdentificavelPorChave)objetoA).retorneChave();
     saida.writeBytes(a + "\n");
     saida.writeBytes(objetoB.getClass().getName() +"\n");
     b = ((IdentificavelPorChave)objetoB).retorneChave();
     saida.writeBytes(b + "\n");
   }
}
```

A Classe Associação com Persistência

```
public class Associacao implements Persistente {
  //...
  private static Aplicacao aplicacao;

  public void carregueSe(BufferedReader entrada) {
     // 1. determina o nome da classe do objetoA.
     // 2. determina a chave do objetoA.
     // 3. Pergunta à aplicação qual objeto da classe
          determinada tem essa chave.

     // idem para objetoB...
  }
}
```


A Classe Associação com Persistência

```
public class Associacao implements Persistente {
  //...
  private static Aplicacao aplicacao;

  public void carregueSe(BufferedReader entrada) {
     Class classeA, classeB;
     String chaveA, chaveB;

     classeA = Class.forName(entrada.readLine());
     chaveA = entrada.readLine();
     objetoA = aplicacao.retorneObjeto(classeA, chaveA);
     classeB = Class.forName(entrada.readLine());
     chaveB = entrada.readLine();
     objetoB = aplicacao.retorneObjeto(classeB, chaveB);
  }
}
```


Implementação da Classe Aplicação

```
public class Aplicacao implements Persistente {
   public Object retorneObjeto(Class classe,
                               String chave) {
     if(classe.getName().equals("Aluno")) {
       return cadAlunos.get(chave);
     }
     if(classe.getName().equals("Disciplina")) {
       return cadDisciplinas.get(chave);
     }
     return null;  // caso contrário.
   }
}
```

Implementação da Classe Aplicação

```
public class Aplicacao implements Persistente {

  public void salveSe(String nomeArquivo) {
    // cria o arquivo de saída
    File arqsaida = new File(nomeArquivo);
    DataOutputStream saida;
    saida = new DataOutputStream(
                        new FileOutputStream(arqsaida));

    this.salveSe(saida);

    // libera os recursos alocados
    saida.close();
  }
}
```


Implementação da Classe Aplicação

```
public class Aplicacao implements Persistente {

  //...

  public void salveSe(DataOutputStream saida) {

    this.salveHashtable(cadAlunos, saida);

    this.salveHashtable(cadDisciplinas, saida);

    this.salveArrayList(cadMatriculas, saida);
  }

}
```


Salvando os Alunos ou Disciplinas

```
public class Aplicacao implements Persistente {
   void salveHashtable(Hashtable tabela,
                       DataOutputStream saida) {
      saida.writeBytes(tabela.size() + "\n");
      Enumeration elementos = tabela.elements();
      while(elementos.hasMoreElements()) {
        Persistente objeto;
        objeto = (Persistente)elementos.nextElement();
        objeto.salveSe(saida);
      }
   }
}
```

polimorfismo

Recuperando os Alunos ou Disciplinas

```
public class Aplicacao implements Persistente {
  void carregueHashtable(Class classe,
                         Hashtable tabela,
                         BufferedReader entrada) {
    int n = Integer.parseInt(entrada.readLine());
    for(int i=0; i < n; i++) {
      Object novoObjeto = classe.newInstance();
      ((Persistente)novoObjeto).carregueSe(entrada);
      tabela.put(
            ((IdentificavelPorChave)novoObjeto).retorneChave(),
            novoObjeto);
    }
  }
}
```


Streams de Objetos (serialização)

- ObjectInputStream e ObjectOutputStream implementam streams para ler ou escrever objetos.
- Até mesmo as referências que um objeto faz a outros objetos são traduzidas adequadamente, podendo ser recuperadas exatamente.
- Permitem implementar persistência de uma maneira transparente. (Desempenho?!)
- Classes de objetos que podem ser serializados devem implementar a interface Serializable.

A Interface Serializable

- A interface Serializable não possui métodos, mas existe apenas por questões de segurança.
- Portanto, a única diferença é na declaração da classe. Exemplo:

```
public class Pessoa implements Serializable {
   // nenhuma alteração!
}
```

Serialização das Estruturas de Dados

```
public class Aplicacao implements Persistente {

  public void salveSe(String nomeArquivo) {
    // cria o arquivo de saída
    File arqsaida = new File(nomeArquivo);
    ObjectOutputStream saida;
    saida = new ObjectOutputStream(
                        new FileOutputStream(arqsaida));

    this.salveSe(saida);

    // libera os recursos alocados
    saida.close();
  }
}
```


Serialização das Estruturas de Dados

```
public class Aplicacao implements Persistente {

 //...

 public void salveSe(ObjectOutputStream saida) {

   saida.writeObject(cadAlunos);

   saida.writeObject(cadDisciplinas);

   saida.writeObject(cadMatriculas);
  }

}
```


Recuperando as Estruturas de Dados

```
public class Aplicacao implements Persistente {
  //...

  public void carregueSe(ObjectInputStream entrada) {

    cadAlunos = (Hashtable) entrada.readObject();

    cadDisciplinas = (Hashtable) entrada.readObject();

    cadMatriculas = (VetorAssociacao) entrada.readObject();

  }
}
```