Programação para Web I
Arquivos

Profa. Silvia Bertagnolli

Arquivos no Java

Em Java a entrada e saída de dados em arquivos é realizada usando os fluxos ou streams

Todas as classes pertencem ao pacote java.io

A principal classe é a classe File, que compreende um ponteiro para um caminho no sistema de arquivos

FileWriter e FileReader

Arquivos de Caracteres

Classes **FileWriter** e **FileReader** são usadas para gravar e ler um fluxo de caracteres de um arquivo

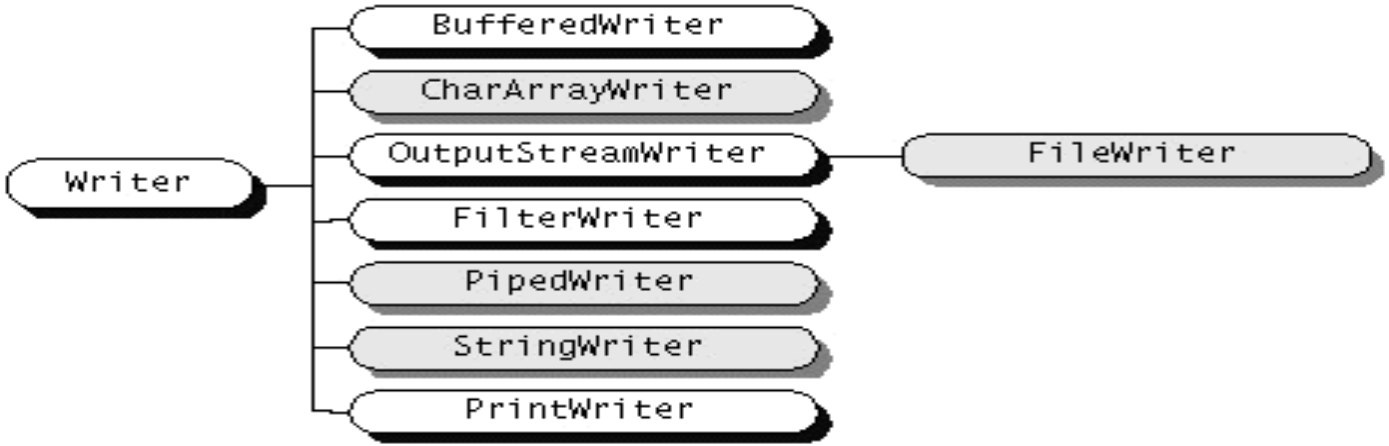
Métodos principais:

- read() - definido na classe **Reader** para leitura de dados de um arquivo

- write() - definido na classe **Writer** para escrita em arquivo

A classe FileReader nos fornece o método read() que lê um único caractere do arquivo e retorna o número inteiro de seu código na tabela unicode, caso seja o fim do arquivo retornará -1

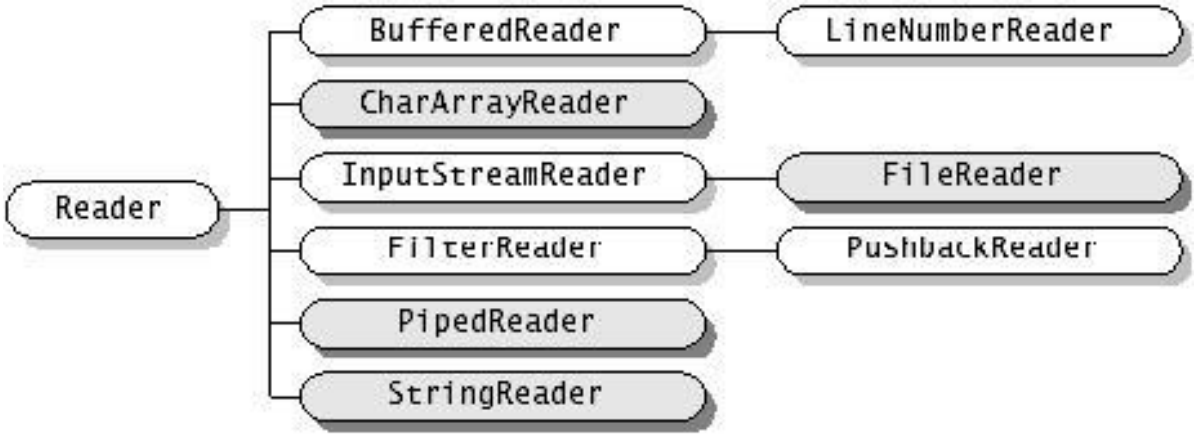
Hierarquia de Classes: writer



Escrevendo caracteres e strings

```
File arqE= new File(“Arquivo.txt");
try{
    FileWriter fw = new FileWriter(arqE) ;
    fw.write('2');
    fw.write("2");
    fw.flush();
    fw.close();
}catch(IOException e){
    System.out.println("Exceção na escrita!");
}
```

Hierarquia de Classes: reader



Lendo caracteres e strings

```
File arqLeit = new File(“Arquivo.txt");
try{
   FileReader fr = new FileReader(arqLeit);
   int  c =   fr.read();
   while( c != -1){
        System.out.print( (char) c );
        c =  fr.read();
   }
}catch(FileNotFoundException e){
       System.out.println("Arquivo não encontrado!");
}catch(IOException e){
        System.out.println("Exceção na leitura!");
}
```

Exercícios: FileReader e FileWriter

1. Faça a leitura de palavras que devem ser gravadas em arquivo qualquer. Use para a leitura de palavras a classe JOptionPane

2. No projeto Arquivos crie a classe ManipulaArquivos que deve declarar os métodos abaixo:

    1. Método de classe gravarArquivo(String nomeArq), que lê dados com JOptionPane e grava no arquivo

    2. Método de classe lerArquivo(String nomeArq), que lê os dados do arquivo e mostra os dados lidos em janela

    3. Método main() deve chamar os métodos gravarArquivo() e lerArquivo()

BufferedWriter e BufferedReader

BufferedWriter e BufferedReader

As classes BufferedWriter e BufferedReader são usadas, respectivamente, para escrever e ler caracteres usando um buffer

A vantagem de armazenar caracteres em buffer é que ele fornece uma gravação eficiente (melhor desempenho) de caracteres, de matrizes e strings individuais

BufferedWriter

```
File arquivo = new File(“Arquivo.txt");
FileWriter fw = new FileWriter( arquivo ) ;
BufferedWriter escrita = new BufferedWriter(fw);
escrita.write( "teste" );
escrita.newLine();
escrita.write( "teste2");
escrita.flush();
escrita.close();
```

BufferedReader

```
File arquivo = new File(“Arquivo.txt");
FileReader fr = new FileReader(arquivo);
BufferedReader leitura = new BufferedReader(fr);
String content;
while( ( content = leitura.readLine() ) != null){
    System.out.println( content );
}
leitura.close();
```

InputStream e OutputStream

InputStream e OutputStream

Fluxos baseados em bytes - representam dados no formato binário

Arquivos binários - criados a partir de fluxos baseados em bytes, lidos por um programa que converte os dados em formato legível por humanos

A classe FileOutputStream é usada para gravar bytes em um arquivo

A classe FileInputStream é usada para ler bytes de um arquivo

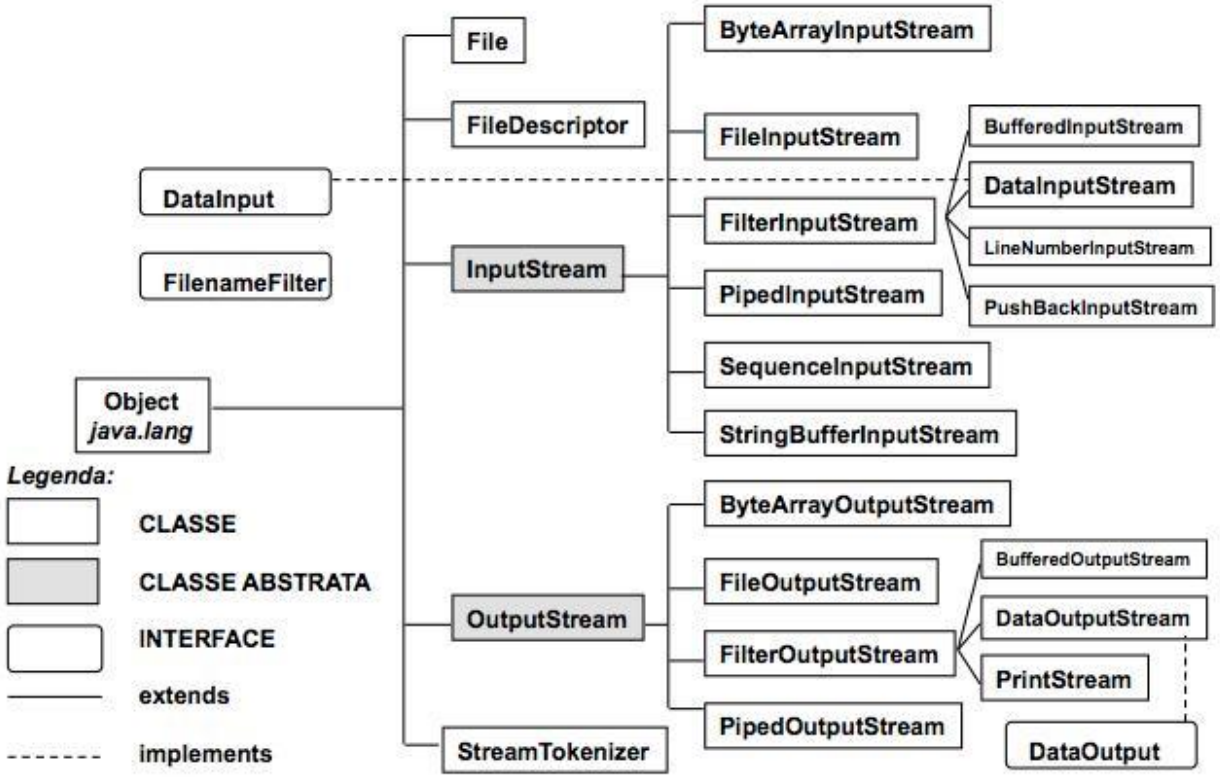
InputStream e OutputStream

Para escrever dados em um arquivo é usado o método **write** que pode receber um byte ou um vetor de bytes

O método **getBytes** converte os caracteres da String em bytes, pois a classe OutputStream precisa desse formato para que os bytes sejam gravados

O método **read** retorna -1 se chegar ao final do arquivo

Classes para Manipular Arquivos



InputStream e OutputStream

Classes para leitura e escrita de um **byte** ou de uma sequência de **bytes**

Métodos principais:

`read()` - definido na classe InputStream para leitura

`write()` - definido na classe OutputStream para escrita

FileOutputStream

```
File arquivo = new File(“Arquivo.bin");
try{
        OutputStream saida=new FileOutputStream(arquivo);
         byte[] b = {50,51,52,53};
         String string = "Teste com várias palavras";
         saida.write( 53 ); saida.write( b );
         saida.write( string.getBytes() );
         saida.flush(); saida.close();
}catch(SecurityException e){System.out.println("Exc. segurança!");
}catch(FileNotFoundException e){
            System.out.println("Arq. não encontrado!");
}catch(IOException e){System.out.println("Exceção na escrita!");
}
```

FileInputStream

```
try{
  InputStream entrada = new FileInputStream(arquivo);
  int content=0;
  while ( (content = entrada.read() ) != -1) {
    System.out.println( content +" - "+ ( (char) content) );
  }
  entrada.close();
}catch(SecurityException e){ System.out.println("Exc seg");
}catch(FileNotFoundException e){
      System.out.println("Arquivo não encontrado!");
}catch(IOException e){System.out.println("Exceção escrita!");
}
```

TRY COM RECURSOS

Sem usar try com recursos

```
File arquivo = new File(“Arquivo.bin");
try{
        OutputStream saida=new FileOutputStream(arquivo);
         byte[] b = {50,51,52,53};
         String string = "Teste com várias palavras";
         saida.write( 53 ); saida.write( b );
         saida.write( string.getBytes() );
         saida.flush(); saida.close();
}catch(SecurityException e){System.out.println("Exc. segurança!");
}catch(FileNotFoundException e){
            System.out.println("Arq. não encontrado!");
}catch(IOException e){System.out.println("Exceção na escrita!");
}
```

Sem usar try com recursos

```
try{
  InputStream entrada = new FileInputStream(arquivo);
  int content=0;
  while ( (content = entrada.read() ) != -1) {
    System.out.println( content +" - "+ ( (char) content) );
  }
  entrada.close();
}catch(SecurityException e){ System.out.println("Exc seg");
}catch(FileNotFoundException e){
      System.out.println("Arquivo não encontrado!");
}catch(IOException e){System.out.println("Exceção escrita!");
}
```

Usando try com recursos

```
File arquivo = new File("Arquivo.bin");
try (OutputStream saida = new FileOutputStream(arquivo);
     InputStream entrada = new FileInputStream(arquivo);) {

    //gravando fluxo baseados em bytes
    byte[] b = {50,51,52,53};
    saida.write(string.getBytes());

   //lendo fluxo baseados em bytes
   int content=0;
   while ( (content = entrada.read() ) != -1)
     System.out.println( content +" - "+ ( (char) content) );
}catch(Exception e) {
   System.out.println("Exceção de leitura ou escrita!");
}
```

Exercícios

Refazer os códigos do projeto dessa semana usando try com recursos e incluir todos os blocos catch necessários para cada um dos exemplos

ObjectOutputStream e ObjectInputStream

ObjectOutputStream e ObjectInputStream

A classe ObjectInputStream permite ler objetos de um arquivo no formato binário

A classe ObjectOutputStream permite gravar objetos em um arquivo no formato binário

Classes são usados para a leitura e a gravação de objetos serializados

Gravando Objetos

```
ObjectOutputStream out = new ObjectOutputStream(new
FileOutputStream( nomeArq) );

Pessoa objPessoa = new Pessoa();

out.writeObject(objPessoa);

out.flush();

out.close();
```

Lendo Objetos

```
ObjectInputStream in = new ObjectInputStream(new
FileInputStream( nomeArq) );

objPessoa = (Pessoa) in.readObject();

System.out.println(objPessoa);

in.close();
```

Serializando Objetos

```
public class Pessoa implements Serializable {

      //....

}
```

Para que um objeto possa ser armazenado em um arquivo no formato correto é necessário que a sua classe implemente a interface Serializable !!!