

Streams, Arquivos e Serialização

Prof. Dr. Lucas C. Ribas

Disciplina: Programação Orientada a Objetos

Departamento de Ciências de Computação e Estatística



UNIVERSIDADE ESTADUAL PAULISTA
“JÚLIO DE MESQUITA FILHO”



IBILCE / UNESP - CÂMPUS DE SÃO JOSÉ DO RIO PRETO



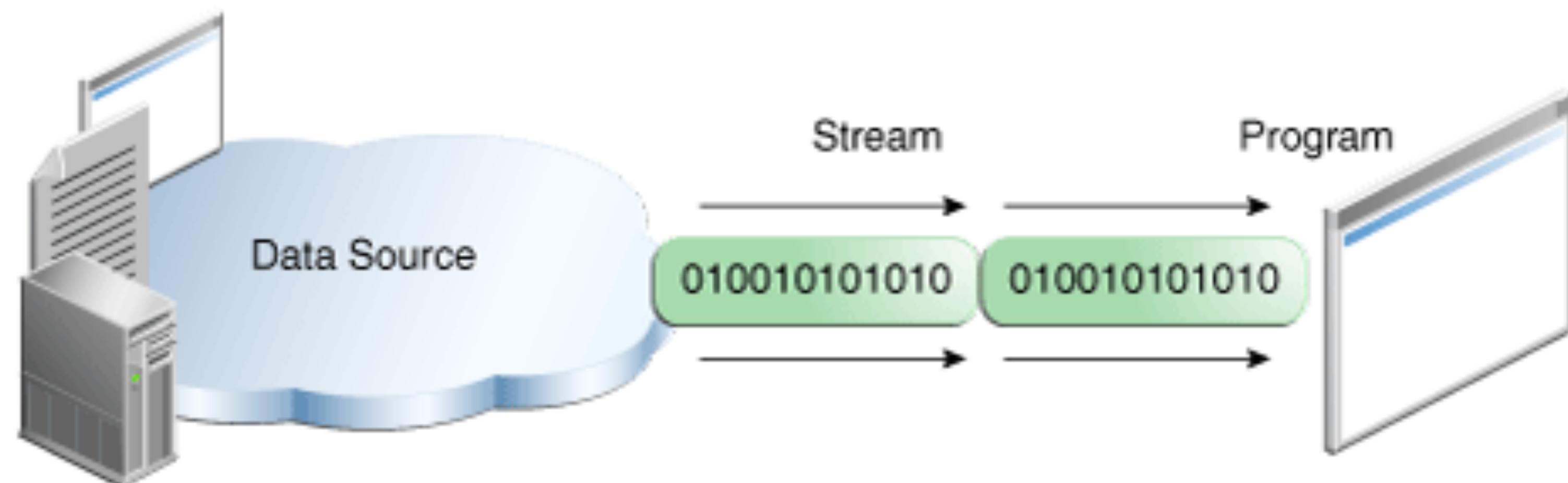
- Introdução às streams
- Streams de bytes e de caracteres
- Buffered streams
- E/S em linha de comando
- Streams de objetos
- Serialização
- Acesso aleatório em arquivos



- Streams representam um fluxo contínuo de dados
 - Entrada ou saída
- Podem se referir a diversas fontes
 - Arquivos, outros programas, conexão de rede, entrada padrão (teclado), memória, etc.
- Streams se baseiam no fluxo **unidirecional** de dados, que podem ser de diferentes tipos
 - Tipos primitivos: byte, char, etc.
 - Objetos
- Algumas streams simplesmente deixam os dados fluirem; outras podem transformar os dados

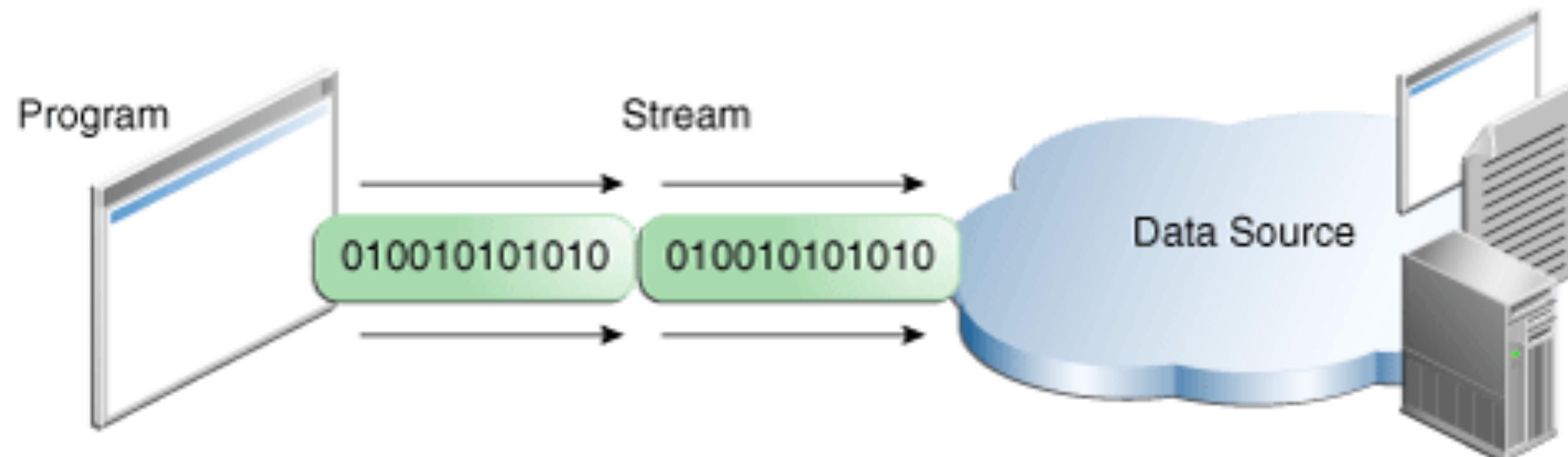


- Lendo informações para um programa





- Escrevendo informações de um programa



Byte Streams



- Usada para ler e escrever bytes (8 bits)
- Existem várias classes para manipular streams de bytes
 - Descendem de [InputStream](#) [OutputStream](#)
- Veremos exemplos utilizando
 - [FileInputStream](#)
 - [FileOutputStream](#)

Byte Streams



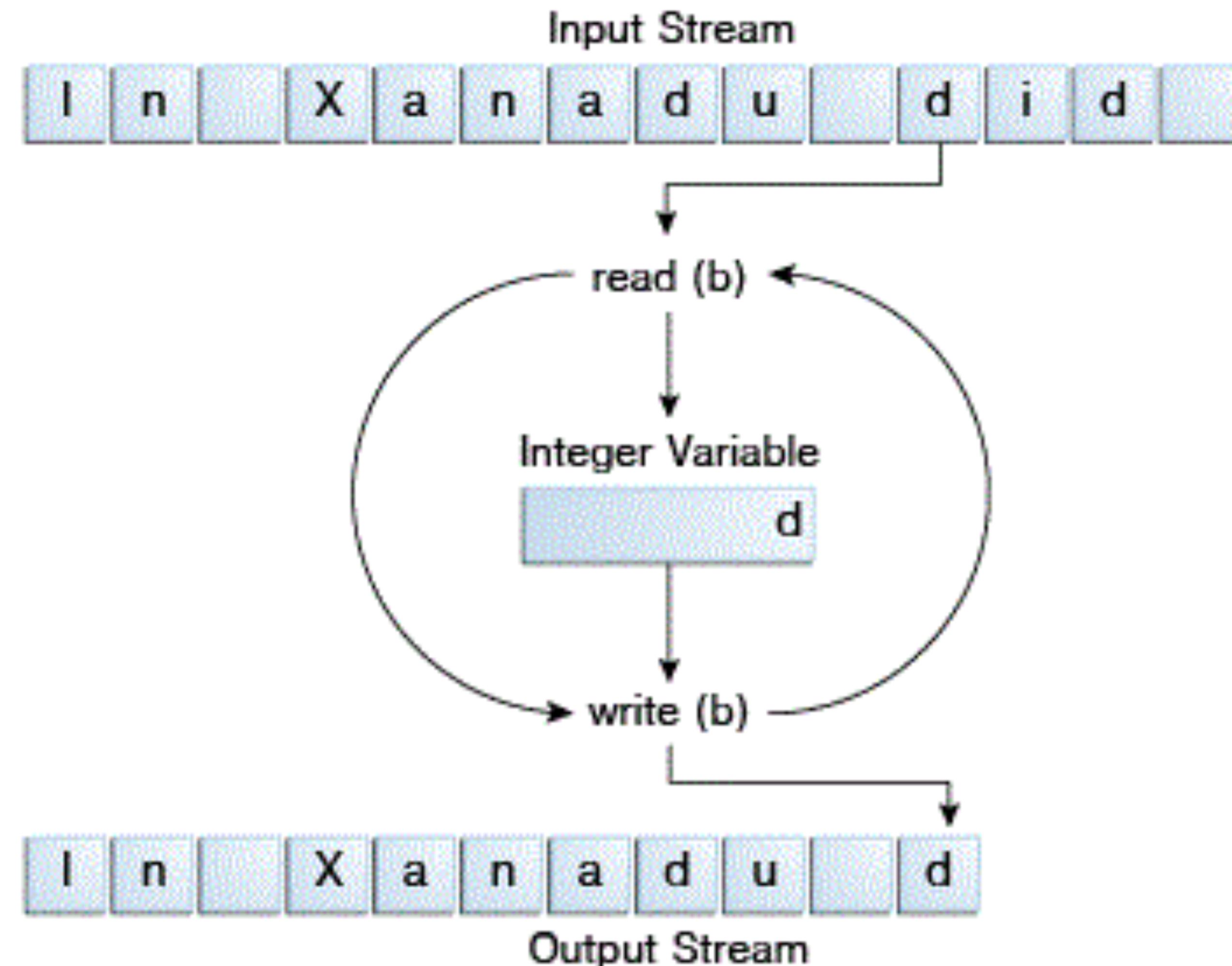
```
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;

public class CopyBytes {
    public static void main(String[] args) throws IOException {
        FileInputStream in = null;
        FileOutputStream out = null;

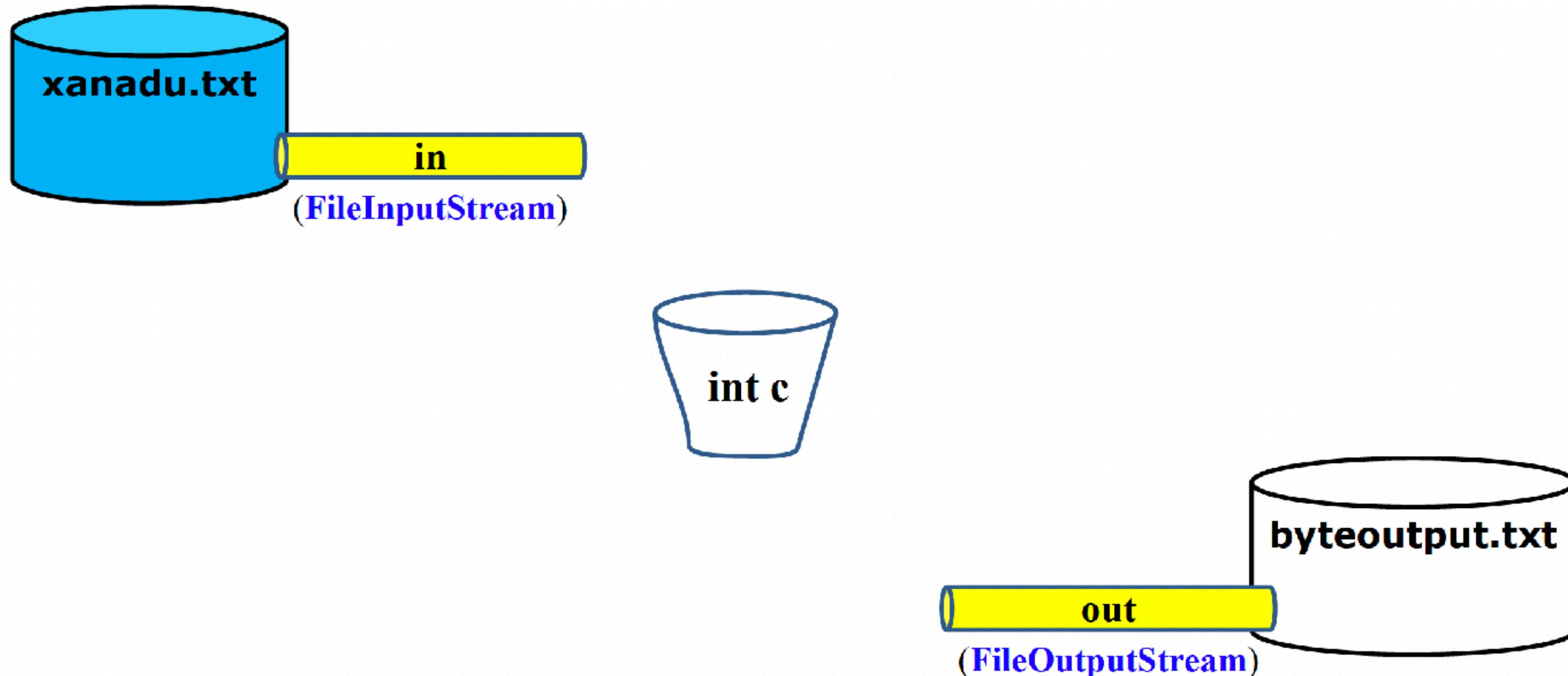
        try {
            in = new FileInputStream("xanadu.txt");
            out = new FileOutputStream("byteoutput.txt");
            int c;
            while ((c = in.read()) != -1) {
                out.write(c);
            }
        } finally {
            if (in != null)
                in.close();
            if (out != null)
                out.close();
        }
    }
}
```

In Xanadu did Kubla Khan
A stately pleasure-dome decree:
Where Alph, the sacred river, ran
Through caverns measureless to man
Down to a sunless sea.

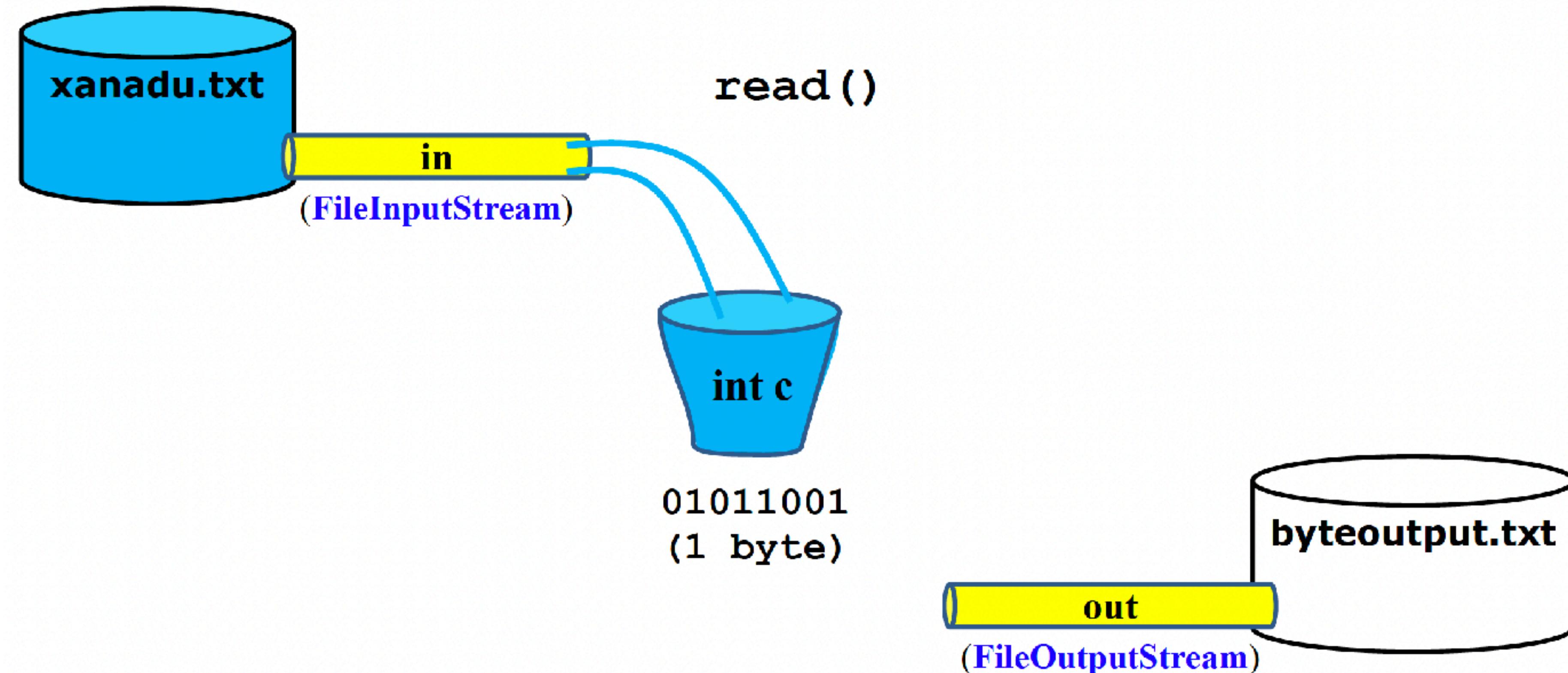
Byte Streams



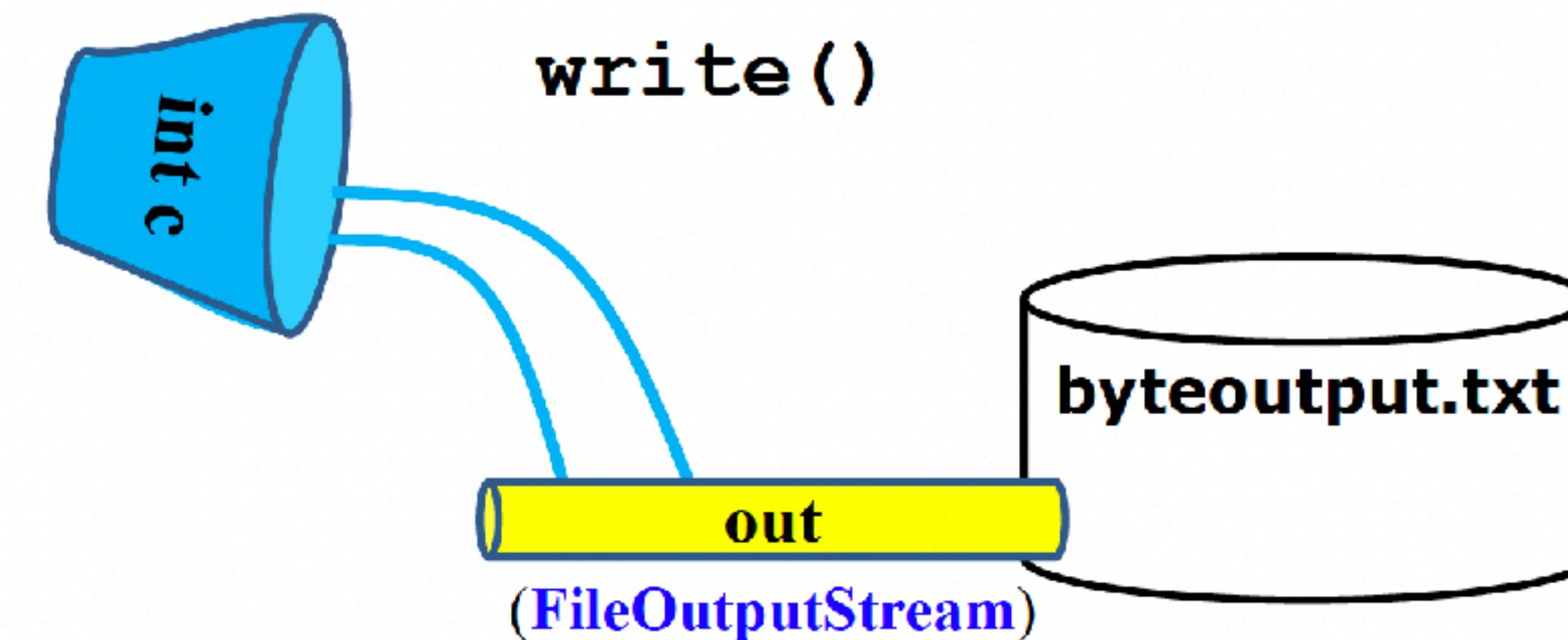
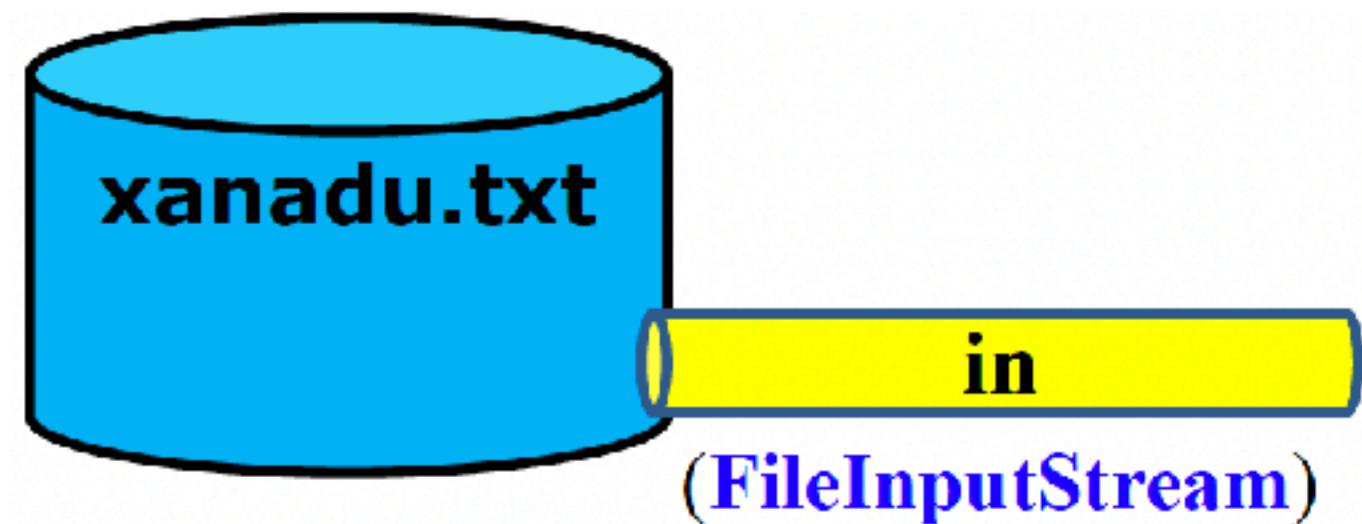
Byte Streams



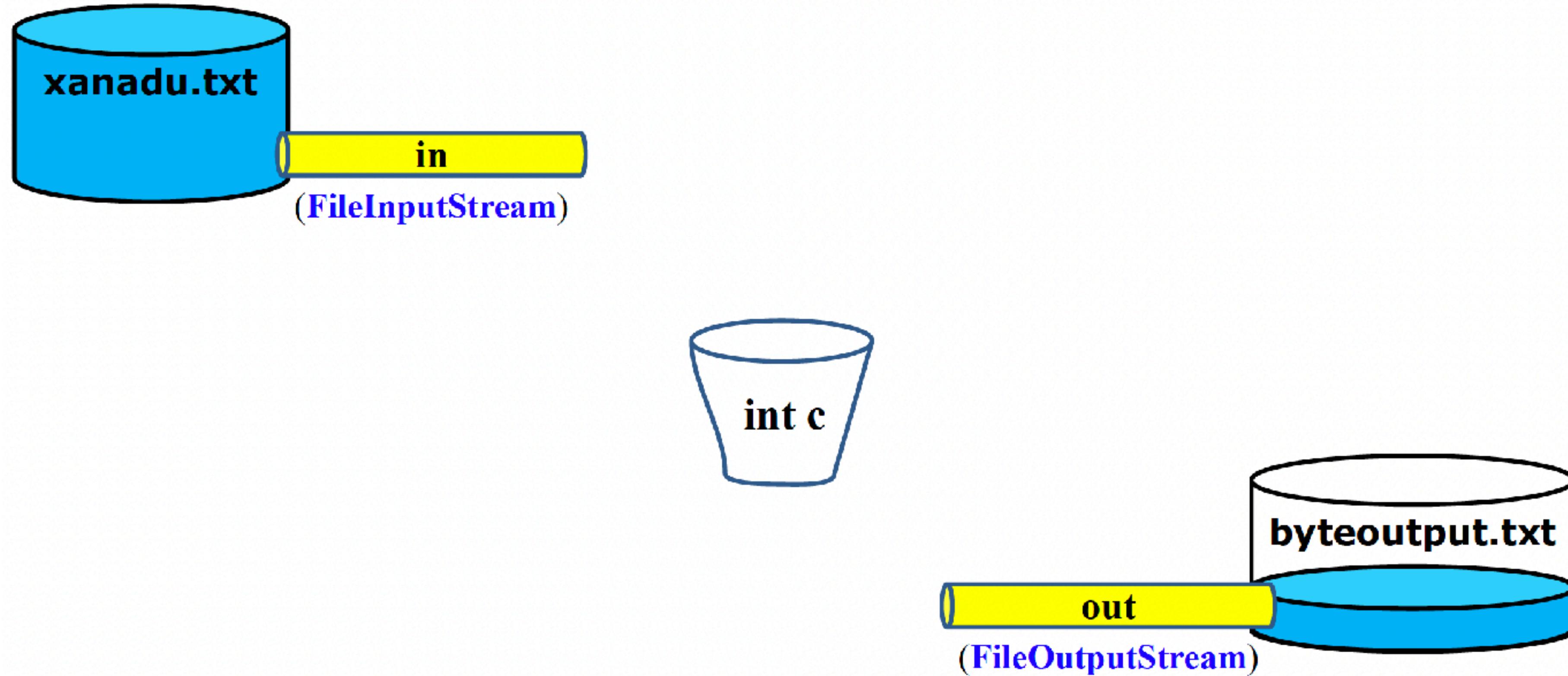
Byte Streams



Byte Streams



Byte Streams





- Stream de bytes é de baixo nível
- Como no exemplo anterior, em geral estamos interessados em dados no formato texto
- Para isso, é mais interessante utilizar streams de caracteres
- O processo é, em geral, muito parecido



- Assim como com bytes, existem várias classes de streams para trabalhar com caracteres
 - Derivam das classes [Reader](#) e [Writer](#)
- Veremos um exemplo com as classes
 - [FileReader](#)
 - [FileWriter](#)

Char Streams



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

public class CopyCharacters {
    public static void main(String[] args) throws IOException {
        FileReader inputStream = null;
        FileWriter outputStream = null;

        try {
            inputStream = new FileReader("xanadu.txt");
            outputStream = new FileWriter("charoutput.txt");
            int c;
            while ((c = inputStream.read()) != -1) {
                outputStream.write(c);
            }
        } finally {
            if (inputStream != null)
                inputStream.close();
            if (outputStream != null)
                outputStream.close();
        }
    }
}
```



- A classe CopyCharacters é muito parecida com a classe CopyBytes
 - Ambas lêem os dados e armazenam em uma variável do tipo **inteiro**
 - No primeiro caso (CopyByte), a variável do tipo inteiro armazena um byte nos seus últimos 8 bits
 - No segundo caso (CopyCharacter), a variável armazena um caracter nos últimos 16 bits
 - Lembre-se que, em Java
 - Um **char** possui 16 bits
 - Um **int** possui 32 bits



- Haverá diferença no número de iterações nos dois códigos?
 - Depende do texto
 - Suponha que estejamos usando codificação UTF-8
 - Todos os caracteres da tabela ASCII são representados por 1 byte
 - Se só houver caracteres ASCII, não há diferença
 - Se houver outros caracteres, 2 ou mais bytes serão utilizados para representação (haverá diferença)
 - Ex: acentos, cedilha, alfabeto Grego

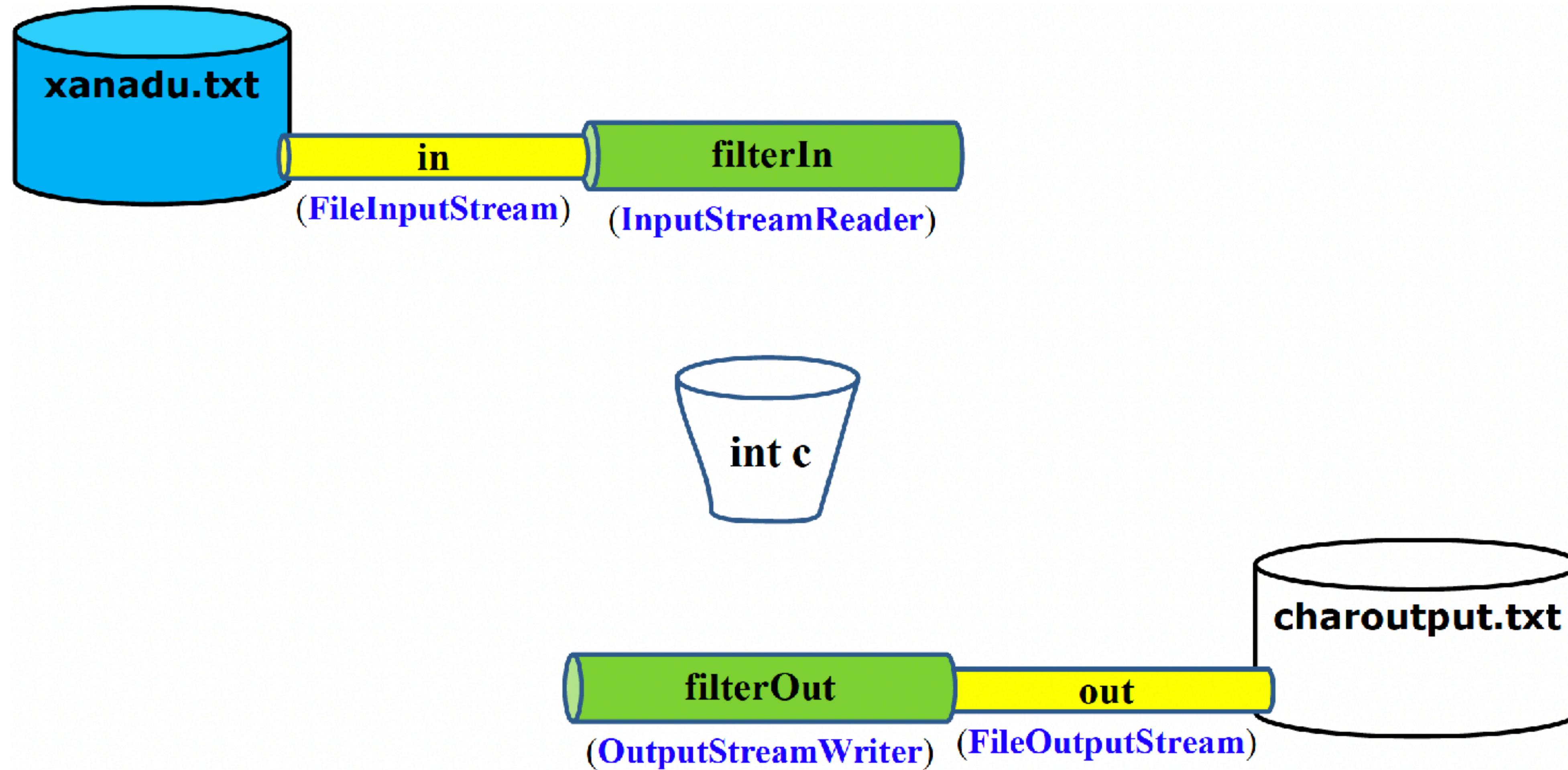


- Streams de caracteres também podem ser usados como filtros para streams de bytes
 - Wrappers, Bridge
- Stream de bytes faz a leitura
- Stream de caracteres converte o dado lido para caracteres



Lucas C. Ribas

Char Streams



Char Streams



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

public class CopyCharacters {
    public static void main(String[] args) throws IOException {
        FileInputStream in = null;
        InputStreamReader filterIn = null;
        FileOutputStream out = null;
        OutputStreamWriter filterOut = null;

        try {
            in = new FileInputStream("xanadu.txt");
            filterIn = new InputStreamReader(in);
            out = new FileOutputStream("charoutput.txt");
            filterOut = new OutputStreamWriter(out);

            int c;
            while ((c = filterIn.read()) != -1) {
                filterOut.write(c);
            }
            ...
        }
    }
}
```

Buffered Streams



- Todos os streams vistos até aqui são do tipo que não usam *buffers* de memória
 - Isso é, todo chamada de leitura/escrita é repassada diretamente para o SO
- Isso torna o processo menos eficiente, pois essas operações são, em geral, lentas
 - Acesso a disco
 - Troca de dados pela rede

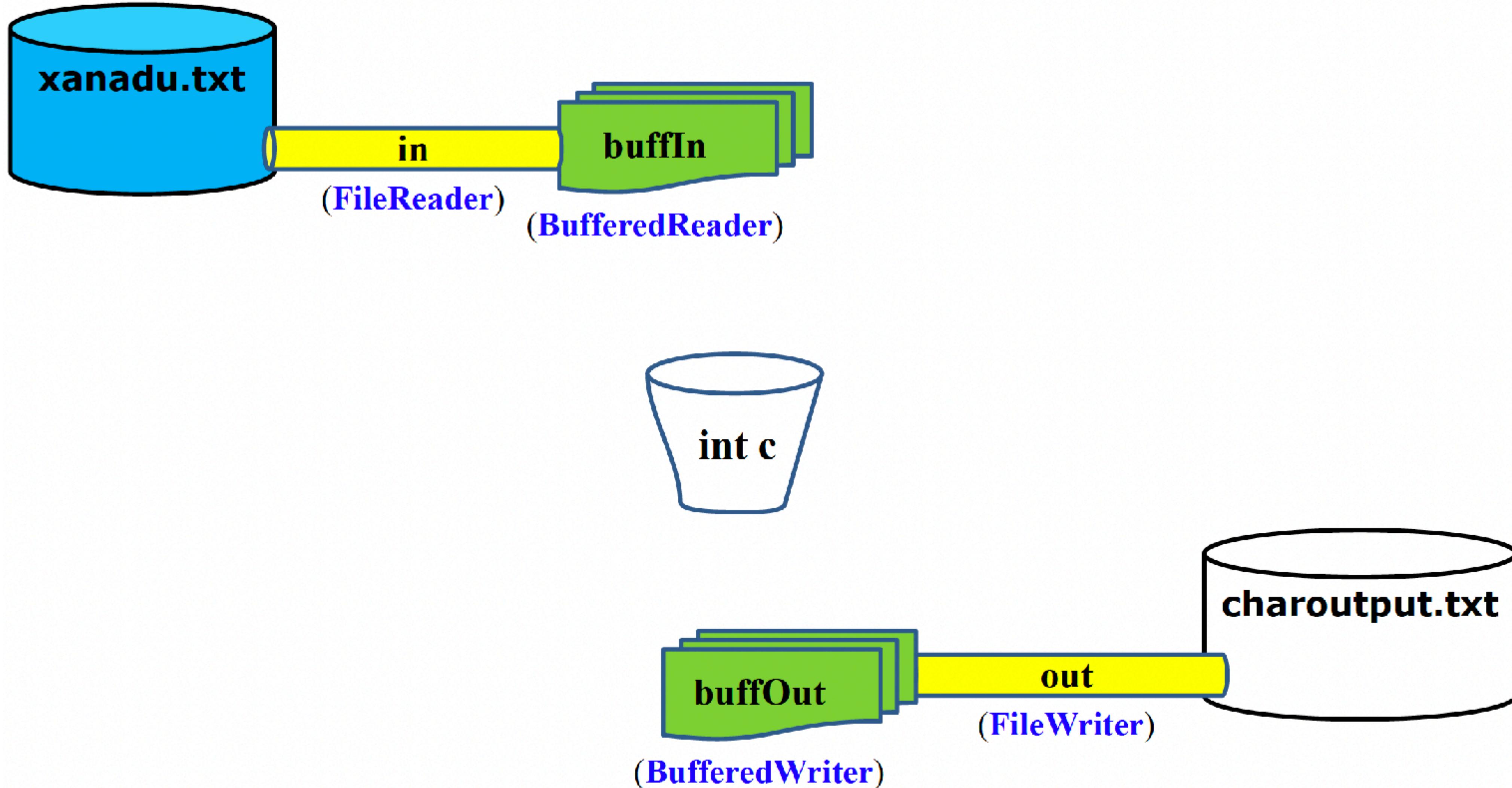


- Para tornar esse processo mais eficiente, o Java fornece streams que usam *buffers* de memória
- Leitura
 - Sempre que uma leitura é requisitada, dados são lidos do buffer (memória)
 - Quando o buffer estiver vazio, novos dados são carregados do arquivo para o buffer
- Escrita
 - Sempre que uma escrita é requisitada, dados são escritos no buffer (memória)
 - Quando o buffer estiver cheio, os dados são descarregados para o arquivo



- Podemos usar buffered streams como filtros das streams sem buffer
 - No construtor da buffered stream passamos a stream sem buffer
- Existem quatro tipos de buffered streams no Java
 - BufferedInputStream (byte)
 - BufferedOutputStream (byte)
 - BufferedReader (caracteres)
 - BufferedWriter (caracteres)

Buffered Streams



Char Streams



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

public class CopyCharacters {
    public static void main(String[] args) throws IOException {
        FileReader in = null;
        BufferedReader buffIn = null;
        FileWriter out = null;
        BufferedWriter buffOut = null;

        try {
            in = new FileReader("xanadu.txt");
            buffIn = new BufferedReader(in);
            out = new FileWriter("charoutput.txt");
            buffOut = new BufferedWriter(out);

            String c;
            while ((c = buffIn.readLine()) != null) {
                buffOut.write(c);
                buffOut.newLine();
            }
            ...
        }
    }
}
```



- Outra vantagem da classe BufferedReader é a possibilidade de leitura do arquivo linha por linha
- Um dos problemas que surge com os diferentes SO é a terminação de linha diferente
 - \n
 - \r
 - \r\n
- O método **readLine()** da classe BufferedReader identifica automaticamente o terminador de linha que compõe o arquivo
- Método **newLine()** para escrever nova linha (BufferedWriter)



- Durante processos de escrita, podemos descarregar o buffer, sem que ele tenha atingido seu limite
 - Flushing
- Quando o método **flush()** de um stream é invocado, o buffer de escrita é descarregado
 - Só terá efeito prático em um stream do tipo buffered

Classe Scanner

A classe Scanner



- A classe Scanner não é um stream propriamente dito
- Porém, ela facilita a leitura de dados por *tokens*
 - Por padrão, os *tokens* são identificados por separadores em branco (*blank*)
 - Espaços em branco, tabulações, final de linha...
- Com ela, podemos especificar o tipo primitivo a ser lido em cada chamada

A classe Scanner



```
import java.io.*;
import java.util.Scanner;

public class ScanXan {
    public static void main(String[] args) throws IOException {
        Scanner s = null;

        try {
            s = new Scanner(new BufferedReader(
                new FileReader("xanadu.txt"))
            );
        }

        while (s.hasNext())
            System.out.println(s.next());
    } finally {
    if (s != null)
        s.close();
}
}
```



In
Xanadu
did
Kubla
Khan
A
stately
pleasure-dome

...



● Outro exemplo: somando números

```
import java.io.FileReader;
import java.io.BufferedReader;
import java.io.IOException;
import java.util.Scanner;
import java.util.Locale;

public class ScanSum {
    public static void main(String[] args) throws IOException {
        Scanner s = null;
        double sum = 0;

        try {
            s = new Scanner(new BufferedReader(
                new FileReader("numbers.txt")))
        };
        s.useLocale(Locale.US); // Decimal separator
```



- Outro exemplo: somando números

```
while (s.hasNext()) {  
    if (s.hasNextDouble()) {  
        sum += s.nextDouble();  
    } else {  
        s.next();  
    }  
}  
} finally {  
    s.close();  
}  
  
System.out.println(sum);  
}  
}
```

E/S de linha de comando



- Programas que interagem com o usuário pela linha de comando
- Há duas possibilidades
 - Uso de streams padrões
 - System.in
 - System.out
 - System.err
 - Uso da classe **Console**



○ Streams padrões

- Não há a necessidade de abrir essas streams (são do sistema)
- Por razões históricas, são streams de bytes
 - Saídas padrão são do tipo [PrintStream](#)
 - Entrada padrão é do tipo [InputStream](#)
- As streams de saída padrão já possuem métodos que manipulam caracteres [ex: `println()`]
- A stream de entrada padrão necessita de outra classe para converter os dados para caracteres
 - Podemos utilizar a classe [Scanner](#)



● Classe Console

- Uma alternativa mais avançada para as streams padrões
- Uma instância de Console deve ser obtida através da classe **System**
 - Se não estiver disponível, retornará **null**

```
Console c = System.console();
if (c == null) {
    System.err.println("No console.");
    System.exit(1);
}
...
```



○ Classe Console

- Fornece os métodos **read()** e **write()**
- Fornece o método **readPassword()**
 - Não imprime os caracteres digitados na tela

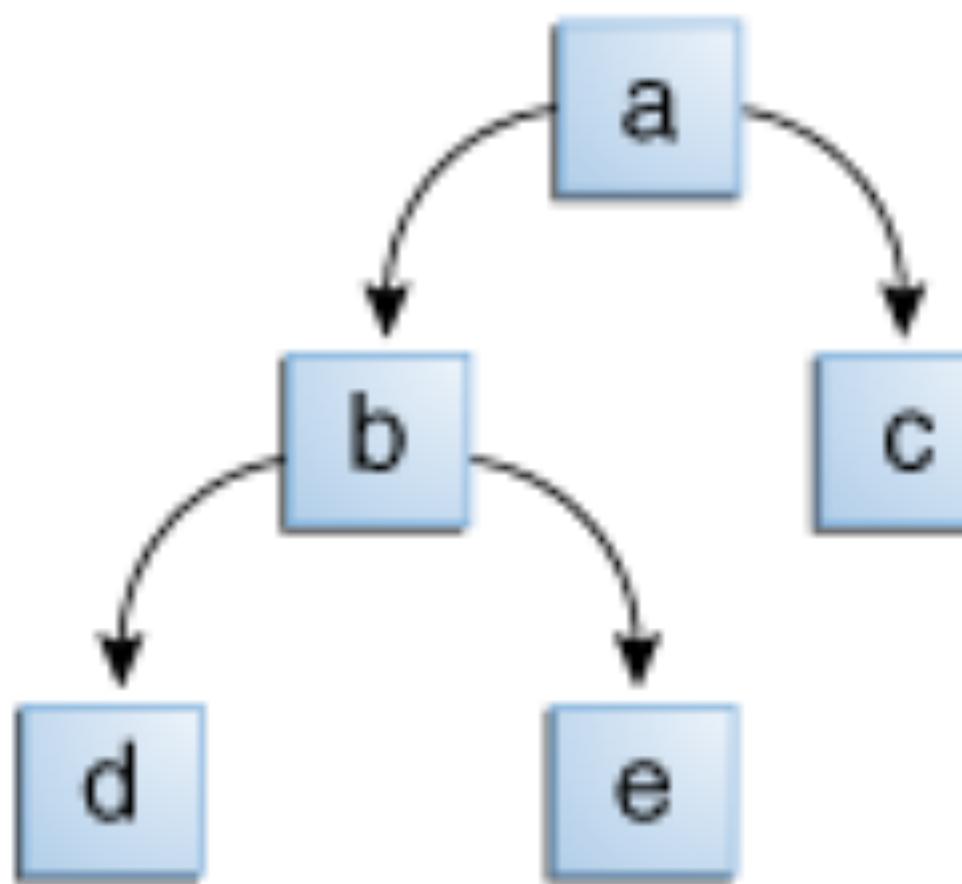
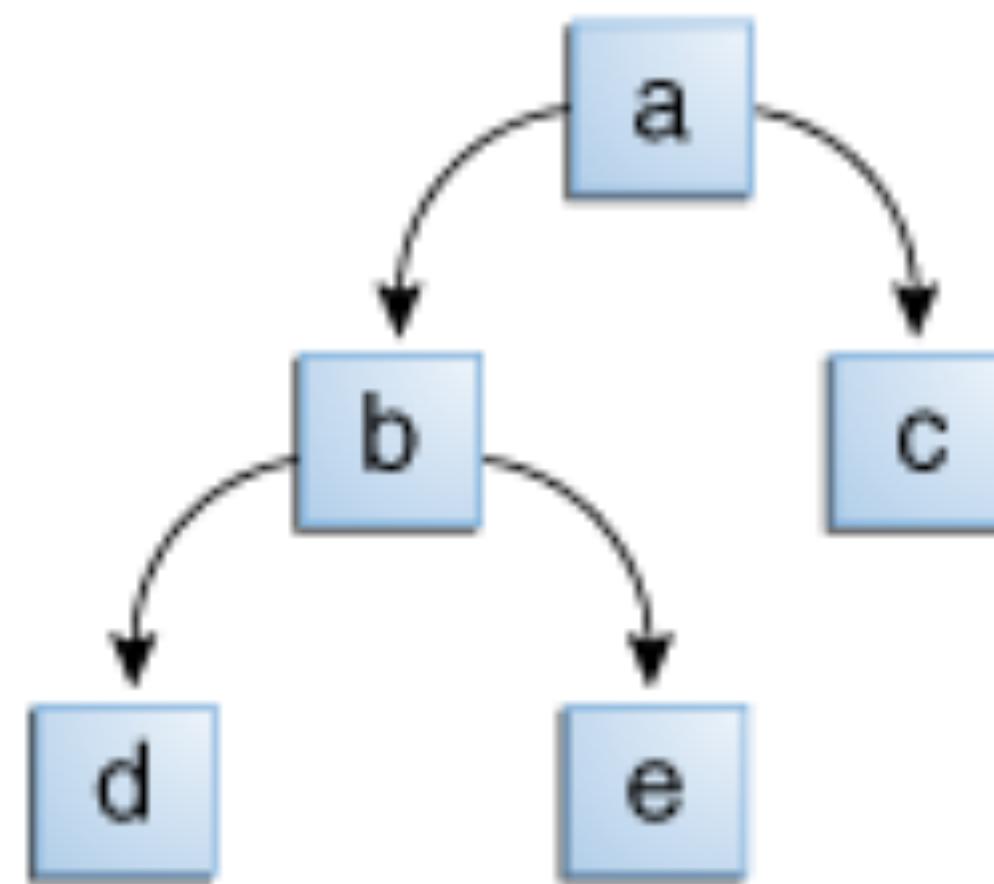
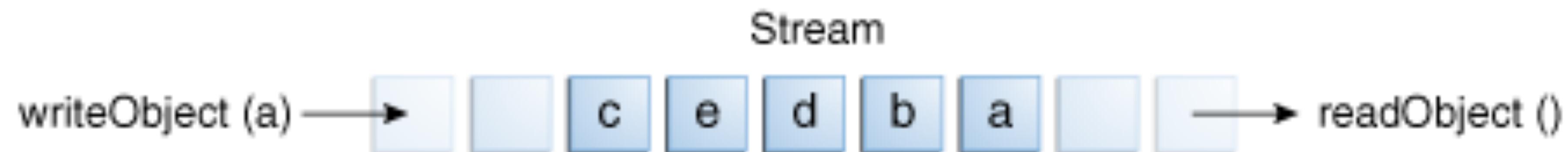
Streams de Objetos

Streams de Objetos



- Além das streams para tipos primitivos, também existe uma stream para objetos
 - ObjectInputStream
 - ObjectOutputStream
- Ou seja, objetos inteiros podem ser escritos e lidos
- Note que um objeto pode conter vários campos, de tipos primitivos ou de outros tipos objetos
 - E estes, por sua vez, podem conter outros objetos
 - Toda essa teia de elementos será salva junto com o objeto requisitado

Streams de Objetos



Streams de Objetos



```
ObjectOutputStream out = null;
try {
    out = new ObjectOutputStream(new FileOutputStream("out.dat"));
    BigDecimal num1 = new BigDecimal(1000);
    BigDecimal num2 = new BigDecimal(20000);
    out.writeObject(num1);
    out.writeObject(num2);
} finally {
    out.close();
}

ObjectInputStream in = null;
try {
    in = new ObjectInputStream(new FileInputStream("out.data"));
    BigDecimal num3 = (BigDecimal) in.readObject();
    BigDecimal num4 = (BigDecimal) in.readObject();

    System.out.format("Num1 = %d. Num2 = %d",
                      num3.intValue(), num4.intValue());
} finally {
    in.close();
}
```

Serialização



- Para ler e gravar objetos, o Java utiliza um recurso chamado de **serialização** de objetos
- Um objeto serializado é representado como uma sequência de bytes
- Após serializado, o objeto pode ser gravado utilizando o stream
- Um objeto serializado também pode ser lido e **deserializado**, voltando a ser um objeto na memória



- Para que um objeto possa ser serializado e desserializado, sua classe precisa implementar a interface **Serializable**
- Esta interface não tem métodos
 - Interface de marcação
- Em classes do tipo **Serializable**, todas as variáveis de instância também precisam ser **Serializable**
 - Ou declaradas como **transient**
 - Variáveis do tipo **transient** são ignoradas durante o processo de serialização
- Todos os tipos primitivos são serializáveis



● Vantagem

- Facilidade para gravar e ler informações
- Não é preciso salvar cada informação dos objetos separadamente

● Desvantagem

- Mais restrito
- Se a estrutura da classe for alterada, objetos salvos não poderão ser lidos

Informações de arquivos

Informações de arquivos e diretórios



● java.io.File

- Checar a existência de arquivos e diretórios
- Criar novos arquivos e diretórios
- Deletar arquivos e diretórios
- Listar arquivos e diretórios dentro de um diretório
- ...
- File.separator
 - Windows (\), Linux (/) e Mac (/) são diferentes

● Pacote java.nio (classes [Path](#) e [Files](#))

- Funcionalidades semelhantes, mais recursos
- Ex: monitorar (vigar) mudanças de arquivos

Acesso aleatório em arquivos



- O uso de streams permite apenas o acesso sequencial (fluxo) de dados
- Do contrário, precisamos usar classes que permitam a leitura e escrita de forma aleatória
 - RandomAccessFile
- Neste caso, trabalhamos com um ponteiro que indica a posição atual de acesso no arquivo
- Método **seek(pos)** da classe **RandomAccessFile** posiciona o ponteiro do arquivo para a posição “pos” no arquivo.

Acesso aleatório em arquivos



```
File fRand = new File("random.txt");
RandomAccessFile raf = new RandomAccessFile(sRand, "rw");
String books[] = new String[5];
books[0] = "Professional JPP\n";
books[1] = "The Java API\n";
books[2] = "Java Security\n";
books[3] = "Java Security Handbook\n";
books[4] = "Hacking Exposed J2EE\n";

// Escreve a partir do inicio
for (int i = 0; i < books.length; i++) {
    raf.writeUTF(books[i]);
}

raf.seek(0);                                //volta ao inicio
raf.writeUTF("Professional JSP\n");          //sobreescreve
raf.seek(raf.length());                      //vai para o final
raf.writeUTF("Servlet Programming\n");        //escreve (append)
raf.seek(0);                                //inicio de novo
```

Acesso aleatório em arquivos



```
File fRand = new File("random.txt");
RandomAccessFile raf = new RandomAccessFile(fRand, "rw");
String books[] = new String[5];
books[0] = "Professional JPP\n";
books[1] = "The Java API\n";
books[2] = "Java Security\n";
books[3] = "Java Security Handbook\n";
books[4] = "Hacking Exposed J2EE\n";

// Escreve a partir do inicio
for (int i = 0; i < books.length; i++)
    raf.writeUTF(books[i]);
}

raf.seek(0);                                //volta ao inicio
raf.writeUTF("Professional JSP\n");          //sobreescreve
raf.seek(raf.length());                      //vai para o final
raf.writeUTF("Servlet Programming\n");        //escreve (append)
raf.seek(0);                                //início de novo
```

Acesso aleatório em arquivos



```
File fRand = new File("random.txt");
RandomAccessFile raf = new RandomAccessFile(fRand, "rw");
String books[] = new String[5];
books[0] = "Professional JPP\n";
books[1] = "The Java API\n";
books[2] = "Java Security\n";
books[3] = "Java Security Handbook\n";
books[4] = "Hacking Exposed J2EE\n";

// Escreve a partir do inicio
for (int i = 0; i < books.length; i++)
    raf.writeUTF(books[i]);
}

raf.seek(0);                                //volta ao inicio
raf.writeUTF("Professional JSP\n");          //sobreescreve
raf.seek(raf.length());                      //vai para o final
raf.writeUTF("Servlet Programming\n");        //escreve (append)
raf.seek(0);                                //inicio de novo
```

Professional JPP
The Java API
Java Security
Java Security Handbook
Hacking Exposed J2EE

Acesso aleatório em arquivos



```
File fRand = new File("random.txt");
RandomAccessFile raf = new RandomAccessFile(fRand, "rw");
String books[] = new String[5];
books[0] = "Professional JPP\n";
books[1] = "The Java API\n";
books[2] = "Java Security\n";
books[3] = "Java Security Handbook\n";
books[4] = "Hacking Exposed J2EE\n";

// Escreve a partir do inicio
for (int i = 0; i < books.length; i++)
    raf.writeUTF(books[i]);
}

→ raf.seek(0);                                //volta ao inicio
raf.writeUTF("Professional JSP\n");           //sobreescreve
raf.seek(raf.length());                        //vai para o final
raf.writeUTF("Servlet Programming\n");          //escreve (append)
raf.seek(0);                                  //inicio de novo
```

Professional JPP
The Java API
Java Security
Java Security Handbook
Hacking Exposed J2EE

Acesso aleatório em arquivos



```
File fRand = new File("random.txt");
RandomAccessFile raf = new RandomAccessFile(fRand, "rw");
String books[] = new String[5];
books[0] = "Professional JPP\n";
books[1] = "The Java API\n";
books[2] = "Java Security\n";
books[3] = "Java Security Handbook\n";
books[4] = "Hacking Exposed J2EE\n";

// Escreve a partir do inicio
for (int i = 0; i < books.length; i++)
    raf.writeUTF(books[i]);
}

raf.seek(0);                                //volta ao inicio
→ raf.writeUTF("Professional JSP\n");        //sobreescreve
raf.seek(raf.length());                      //vai para o final
raf.writeUTF("Servlet Programming\n");        //escreve (append)
raf.seek(0);                                //inicio de novo
```

Professional JSP

The Java API

Java Security

Java Security Handbook

Hacking Exposed J2EE



```
File fRand = new File("random.txt");
RandomAccessFile raf = new RandomAccessFile(fRand, "rw");
String books[] = new String[5];
books[0] = "Professional JPP\n";
books[1] = "The Java API\n";
books[2] = "Java Security\n";
books[3] = "Java Security Handbook\n";
books[4] = "Hacking Exposed J2EE\n";

// Escreve a partir do inicio
for (int i = 0; i < books.length; i++)
    raf.writeUTF(books[i]);
}

raf.seek(0);                                //volta ao inicio
raf.writeUTF("Professional JSP\n");          //sobreescreve
→ raf.seek(raf.length());                   //vai para o final
raf.writeUTF("Servlet Programming\n");        //escreve (append)
raf.seek(0);                                //inicio de novo
```

Professional JSP

The Java API

Java Security

Java Security Handbook

Hacking Exposed J2EE



```
File fRand = new File("random.txt");
RandomAccessFile raf = new RandomAccessFile(fRand, "rw");
String books[] = new String[5];
books[0] = "Professional JSP\n";
books[1] = "The Java API\n";
books[2] = "Java Security\n";
books[3] = "Java Security Handbook\n";
books[4] = "Hacking Exposed J2EE\n";

// Escreve a partir do inicio
for (int i = 0; i < books.length; i++)
    raf.writeUTF(books[i]);
}

raf.seek(0);                                //volta ao inicio
raf.writeUTF("Professional JSP\n");          //sobreescreve
raf.seek(raf.length());                      //vai para o final
→ raf.writeUTF("Servlet Programming\n");      //escreve (append)
raf.seek(0);                                //inicio de novo
```

Professional JSP

The Java API

Java Security

Java Security Handbook

Hacking Exposed J2EE

Servlet Programming



```
File fRand = new File("random.txt");
RandomAccessFile raf = new RandomAccessFile(fRand, "rw");
String books[] = new String[5];
books[0] = "Professional JPP\n";
books[1] = "The Java API\n";
books[2] = "Java Security\n";
books[3] = "Java Security Handbook\n";
books[4] = "Hacking Exposed J2EE\n";

// Escreve a partir do inicio
for (int i = 0; i < books.length; i++)
    raf.writeUTF(books[i]);
}

raf.seek(0);                                //volta ao inicio
raf.writeUTF("Professional JSP\n");          //sobreescreve
raf.seek(raf.length());                      //vai para o final
raf.writeUTF("Servlet Programming\n");        //escreve (append)
→ raf.seek(0);                             //inicio de novo
```

Professional JSP

The Java API

Java Security

Java Security Handbook

Hacking Exposed J2EE

Servlet Programming



```
File fRand = new File("random.txt");
RandomAccessFile raf = new RandomAccessFile(fRand, "rw");
String books[] = new String[5];
books[0] = "Professional JPP\n";
books[1] = "The Java API\n";
books[2] = "Java Security\n";
books[3] = "Java Security Handbook\n";
books[4] = "Hacking Exposed J2EE\n";

// Escreve a partir do inicio
for (int i = 0; i < books.length; i++)
    raf.writeUTF(books[i]);
}

raf.seek(0);
raf.writeUTF("P");
raf.seek(raf.length());
raf.writeUTF("S");
raf.seek(0);
```

Professional JSP
The Java API
Java Security
Java Security Handbook
Hacking Exposed J2EE
Servlet Programming

Abaixo pode conter um código para leitura do arquivo, uma vez que o ponteiro foi colocado no início novamente.

o inicio
creve
a o final
(append)
de novo



- Introdução à streams
- Streams de bytes e de caracteres
- Buffered streams
- E/S em linha de comando
- Streams de objetos
- Serialização
- Acesso aleatório em arquivos





Professor: acabou a aula

Eu:





- Crie um aplicativo que rode em terminal, contendo duas funções
 - Cadastro de usuário
 - Fazer login
- O cadastro de usuários deve requisitar três informações de cada novo usuário
 - Nome completo
 - Nome de usuario (login)
 - Senha

- As informações dos usuários devem ser salvas em arquivos, de duas maneiras distintas
 - Arquivo texto
 - Arquivo de objetos que representam usuários
- Nos arquivos texto, coloque cada informação de um usuário em uma linha; separe os usuários por uma linha em branco
- Crie uma codificação para a senha
 - Função que codifica a senha antes de salvá-la
 - Função que decodifica a senha ao carregá-la
- Para os arquivos com objetos, crie uma classe serializável chamada **User**



- O usuário pode escolher qual fonte de informações usar para o login
(arquivo texto ou arquivo com objetos)
 - Informações devem coincidir
- Para fazer login, o aplicativo deve varrer o arquivo e verificar se o usuário existe e se a senha está correta.
- Se o login falhar, escreva uma mensagem de erro
 - Falha no login
- Se o login tiver sucesso, escreva outra mensagem
 - Login com sucesso



- DEITEL, H. M. & DEITEL, P.J. "Java : como programar", Bookman, 2017.
- Material baseado nos slides:
 - Luiz E. Virgilio da Silva. Notas de Aula de Programação Orientada a Objetos (ICMC/USP).
 - José Fernando Junior. Notas de Aula de Programação Orientada a Objetos (ICMC/USP).