Introdução a Programação

Arquivos

Topicos da Aula

- Hoje aprenderemos a importância de Persistência
  - Pra que serve?
  - Persistindo dados: Arquivos
- Arquivos
  - Operações Basicas
  - Modos de arquivo
    - Texto
    - Binario
  - Modos de acesso
    - Seqüencial
    - Aleatorio
  - Pacote java.io
  - Otimizando acesso com Buffers

Persistência... pra quê?

- Não perder os dados no fim da execução de um programa
  - Memória temporária(volatil)
    - principal
    - Mais rápida e cara
  - Memória permanente
    - secundária
    - mais lenta e barata

Arquivos

- Um arquivo representa um elemento de informação armazenado em memória secundária (disco)
- Características:
  - Informações são persistidas
  - Atribui-se nomes aos elementos de informação (arquivos e diretórios),em vez de endereços de memória
  - Acesso às informações são mais lentos
- Para melhorar velocidade de acesso, a cada acesso, transfere-se trechos maiores do arquivo para espaços da memória (buffers)

Arquivos

- Informações são armazenadas como *Streams* (fluxo seqüencial) de bytes
- Não existem tipos
- Seqüência de bytes
  - como os dados são representados na memória

Operações sobre Arquivos

- Operações:
  - Abertura
    - Sistema Operacional (SO) encontra arquivo pelo nome
    - Prepara buffer na memória
  - Leitura
    - SO recupera trecho solicitado do arquivo
    - Parte ou todo trecho pode vir do buffer
  - Escrita
    - SO altera conteúdo do arquivo
    - Alteração é feita primeiro no buffer para depois der transferida para o disco
  - Fechamento
    - Informação do buffer é atualizada no disco
    - Área do buffer utilizada na memória é liberada

Modos de Arquivo

- Dois modos de acesso:
  - Texto e Binário
- Modo Texto:
  - É interpretado como uma seqüência de caracteres(char) agrupadas em linhas
  - Linhas são separadas por um caractere de nova linha
  - Unidade básica: char
  - Vantagens:
    - Pode ser lido facilmente por uma pessoa
    - Editado por editores de texto convencionais
  - Desvantagens
    - Codificação dos caracteres pode variar (ASCII, UTF-8, ISSO-8859, etc)
    - Arquivos tendem a ser maiores (todas os dados são convertidos para caracteres)

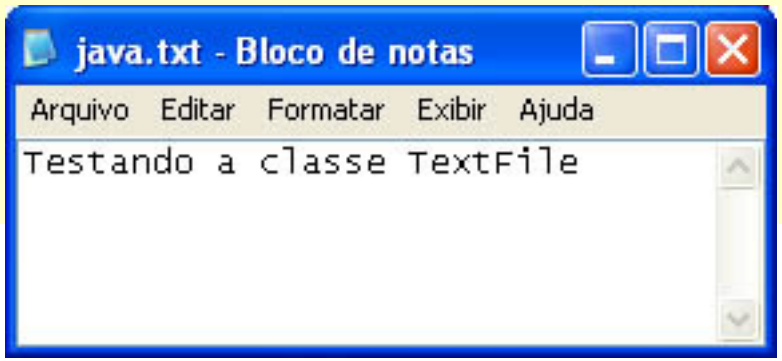
Exemplo miniJava.io.TextFile

```
public class TesteTextFile {
   public static void main(String args[]) {
        String str = "Testando a classe TextFile";
        try {
             TextFile arquivo = new TextFile("java.in", "java.txt");
             arquivo.writeString(str);
             arquivo.close();
        } catch (FileNotFoundException e) {
             MiniJavaSystem console = new MiniJavaSystem();
             console.println("O arquivo não foi achado!");
        } catch (IOException e) {
             MiniJavaSystem console = new MiniJavaSystem();
             console.println("Algum problema ocorreu.");
        }
   }
}
```

Cria o objeto para manipular um arquivo e o abre

Escreve uma String no arquivo

Fecha o arquivo



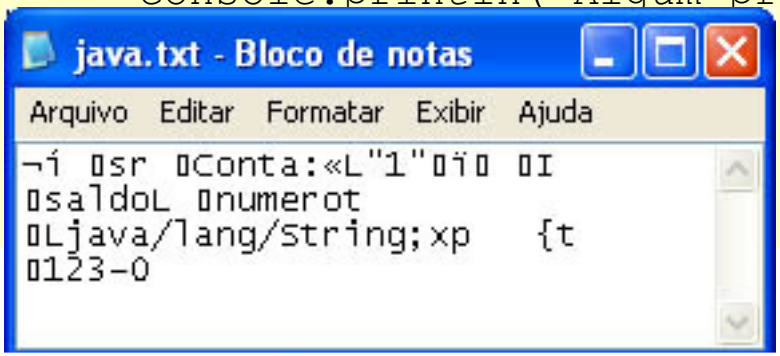
Modos de Arquivo

- Modo Binário:
  - Dados são armazenados da mesma forma que são armazenados na memória principal
  - Pode permitir a escrita de objetos
    - Serializable
  - Unidade básica: byte
  - Vantagens:
    - Facilmente interpretados por programas
    - Maior velocidade de manipulação
    - Arquivos são, geralmente, mais compactos
  - Desvantagem
    - Difícil de serem entendidos por pessoas (ilegíveis)

Exemplo miniJava.io.JavaFile

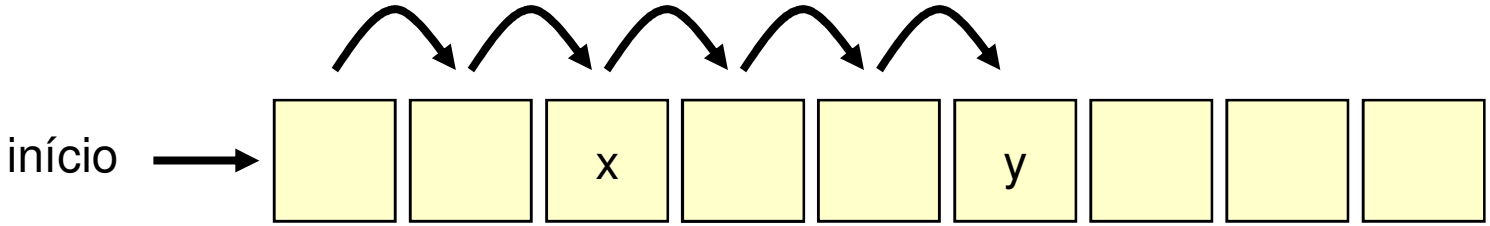
```
public class TesteJavaFile {
   public static void main(String args[]) {
        Conta c1 = new Conta(123, "123-0");
        try {
                JavaFile arquivo = new JavaFile("java.in", "java.txt");
                arquivo.writeObject(c1);
                arquivo.close();
        } catch (FileNotFoundException e) {
                MiniJavaSystem console = new MiniJavaSystem();
                console.println("O arquivo não foi achado!");
        } catch (IOException e) {
                MiniJavaSystem console = new MiniJavaSystem();
                console.println("Algum problema ocorreu.");
        }
   }
}
```

Escreve um objeto do tipo Conta

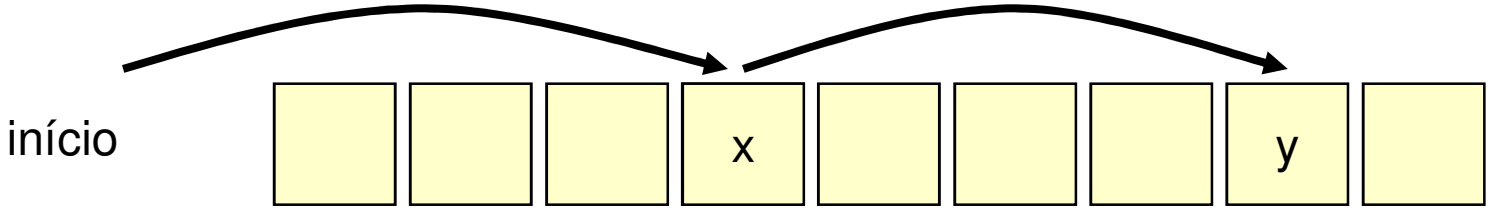


Modos de Acesso: Seqüencial x Randômico

- Acesso seqüencial



- Acesso randômico



Modos de Acesso: Seqüencial x Randômico

- TextFile e JavaFile → acesso seqüencial
  - Escrita contínua
- RandomJavaFile → Acesso randômico
  - Qualquer lugar do arquivo
    - `seek(long distancia)`
    - `skipBytes(int distancia)`
- Leitura (r) → ex: `readInt()`
  - boolean byte char double int String …
- escrita (rw) → ex: `writeInt(int dado)`
  - boolean char double int

Revisão do Tamanho dos Tipos Primitivos de Java

- Qual o tamanho de um :
  - **byte** e **boolean**: 1 byte (8 bits)
  - **char**(Unicode) e **short**: 2 bytes (16 bits)
  - **int** e **float**: 4 bytes (32 bits)
  - **double** e **long**: 8 bytes (64 bits)

Exemplo miniJava.io.RandomJavaFile

```
public class TesteRandomJavaFile {
   public static void main(String args[]) {
        Conta c1 = new Conta(123, "123-0");
        Conta c2 = new Conta(456, "456-7");
        MiniJavaSystem console = new MiniJavaSystem();
        try {
                RandomJavaFile arquivo = new RandomJavaFile("java.bin",
   "rw");
                arquivo.writeInt(c1.getSaldo());
                arquivo.writeInt(c2.getSaldo());
                arquivo.writeString(c1.getNumero());
                arquivo.writeString(c2.getNumero());
                arquivo.close();
```

Indica abertura para leitura e escrita

```
(...)
```

Exemplo miniJava.io.RandomJavaFile

```
(...)
                arquivo = new RandomJavaFile("java.bin", "r");
                console.println("c1.saldo: " + arquivo.readInt());
                console.println("c2.saldo: " + arquivo.readInt());
                console.println("c1.numero: " + arquivo.readString());
                console.println("c2.numero: " + arquivo.readString());
                arquivo.close();
        } catch (FileNotFoundException e) {
                console.println("O arquivo não foi achado!");
        } catch (IOException e) {
                console.println("Algum problema ocorreu.");
        }
    }
}
```

Pacote java.io

- java.io
  - InputStream OutputStream (abstratas)
    - Leitura e escrita de streams de bytes
  - FileInputStream FileOutputStream
    - leitura e escrita de bytes em arquivos
  - DataInputStream/DataOutputStream
    - Escrita e leitura de tipos primitivos
  - ObjectInputStream/ObjectOutputStream
    - Escrita e leitura de Objetos

Pacote java.io

- Reader/Writer
  - Leitura e escrita de texto
- FileReader/FileWriter
  - Leitura e escrita de texto em arquivos
- BufferedReader/PrintWriter
  - Leitura e escrita de Strings

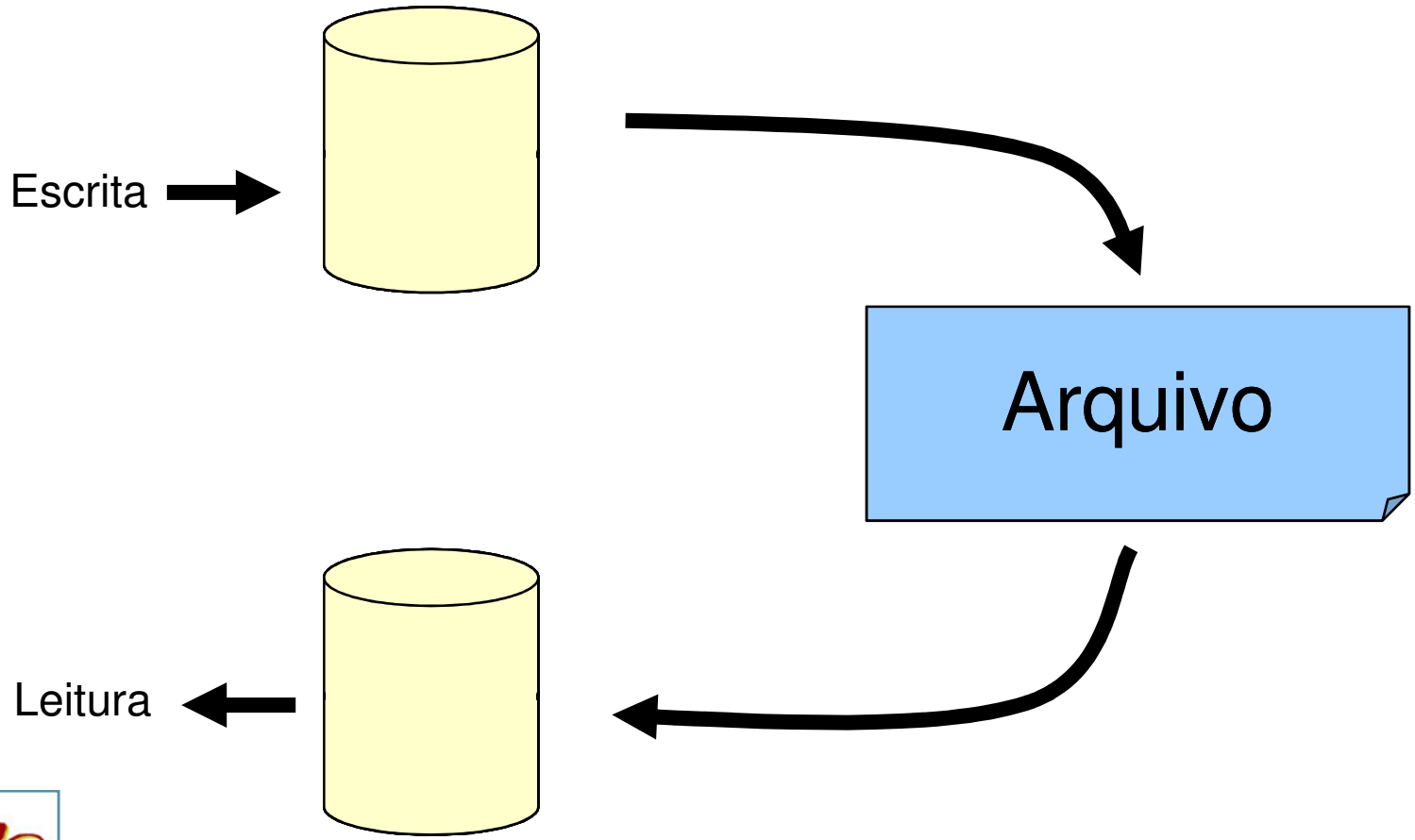
Filtros

- Ler dados como bytes brutos é rápido, mas grosseiro;
- Normalmente, lêem-se os agregados de bytes que formam `ints`, `floats`, `doubles`...
- Algumas classes são capazes de ler um conjunto de bytes e transformá-las para tipos primitivos. (Ex.: RandomJavaFile)

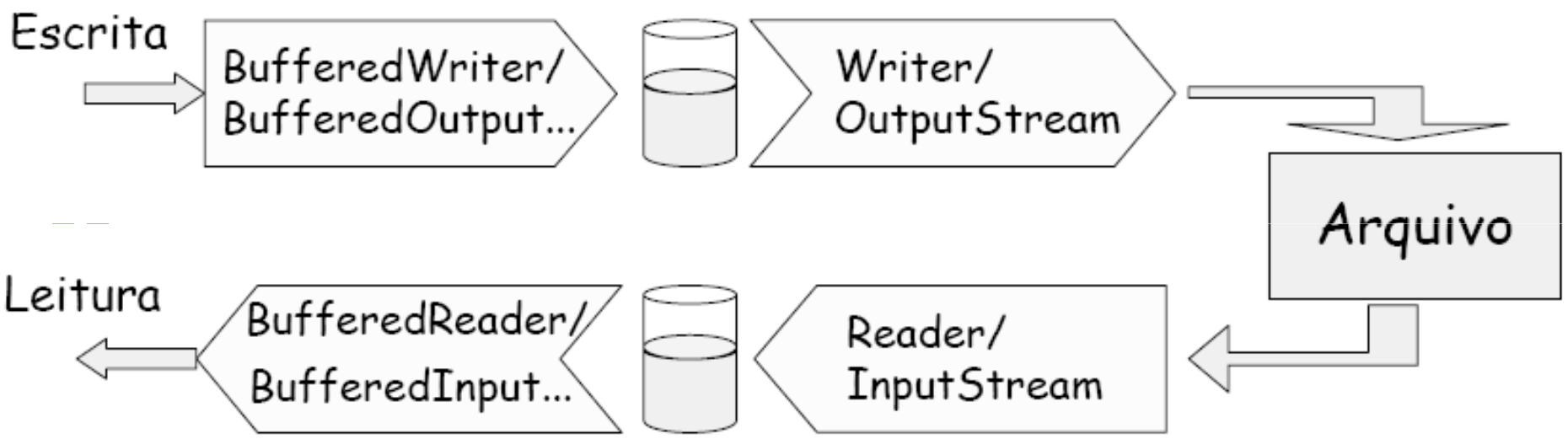
Buffering

- Transferência física de saída → operação lenta se comparada com as velocidades do processador e da memória principal;
- Armazenamento em buffer (*buffering)* → melhorias significativas de desempenho de entrada e saída;
- `flush()` → força o envio a qualquer momento;

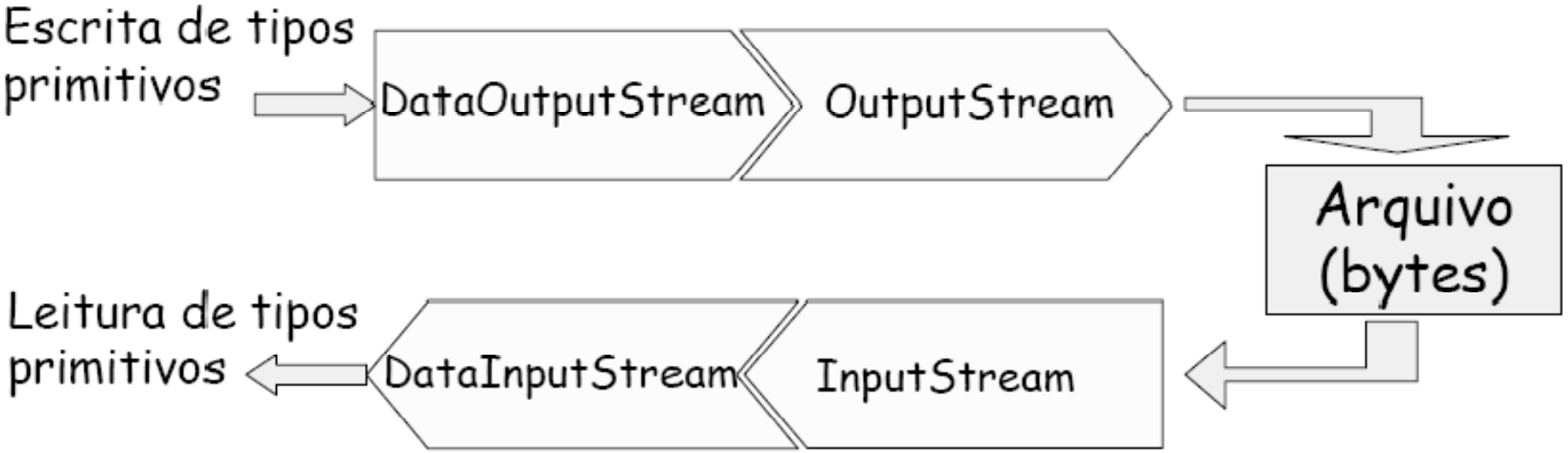
Buffering



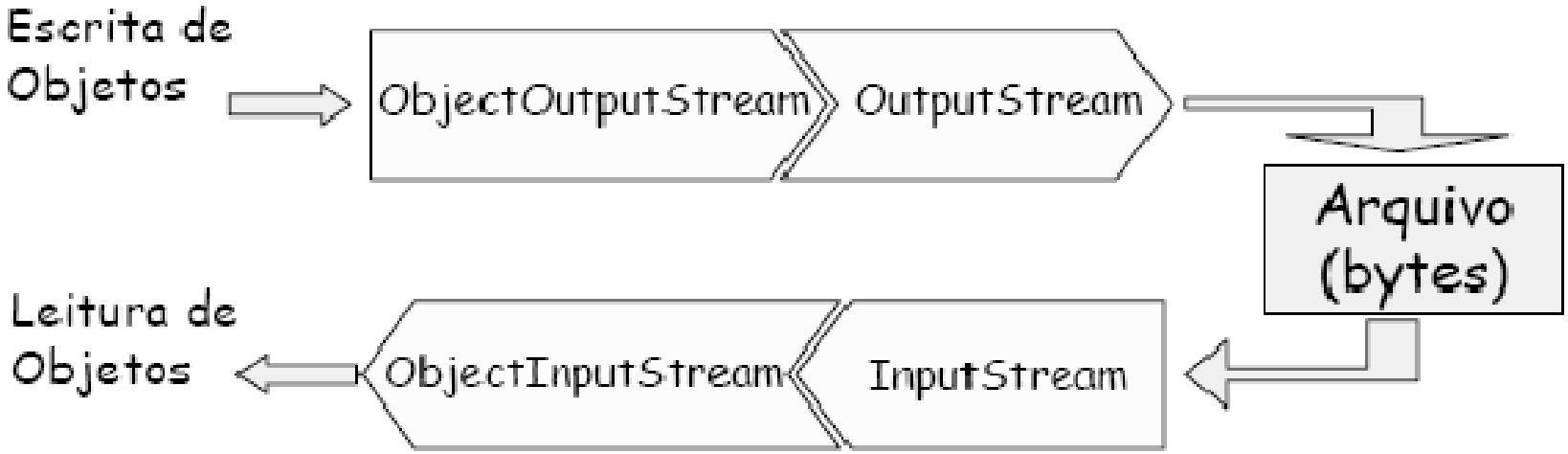
Buffering


Escrita
BufferedWriter/
BufferedOutput...
Writer/
OutputStream
Arquivo
Leitura
BufferedReader/
BufferedInput...
Reader/
InputStream

Tipos Primitivos



Objetos


Escrita de Objetos
ObjectOutputStream
OutputStream
Arquivo (bytes)
Leitura de Objetos
ObjectInputStream
InputStream

Considerações sobre arquivos

- Sistema operacional mantém um “cursor” que indica a posição de trabalho no arquivo
- Dados em muitos arquivos seqüenciais não podem ser modificados sem o risco de destruir parte desses dados →Arquivo geralmente é regravado por inteiro;
- Arquivos de acesso aleatório → Modificação mais fácil quando registros de tamanho fixo.

TextFile	**JavaFile**	**RandomJavaFile**
Arquivo de texto	Arquivo binário	Arquivo binário
Legível por humano	Não legível por humano	Não legível por humano
char	byte	byte
Acesso seqüencial	Acesso seqüencial	Acesso aleatório