# Arquivos em Java

# A classe File

- A classe java.io.File representa um arquivo ou um diretório que pode ou não existir no sistema de arquivos da plataforma em uso
- Encapsula um String com o **nome** do arquivo/diretório
- **Não inclui o conteúdo** do arquivo associado

# A classe File: construtores

- File(String pathname)
- File(String dir,  String subpath)
- File(File dir, String subpath)

# File: propriedades de arquivos e pastas

- String getName()
- String getPath()
- String getAbsolutePath()
- String getParent()
- long length()
- boolean exists()
- boolean isFile()
- boolean isDirectory()
- boolean canWrite()
- boolean canRead()

# File: propriedades

```
public class DetalheArquivo {
  public static void main(java.lang.String[] args) {
    String nomeArquivo =  "c:\\winnt\\notepad.exe";
    File arq = new File(nomeArquivo);
    if (!arq.exists()) {
      System.out.print("Arquivo nao existe: "+nomeArquivo);
    } else {
      System.out.println("Nome: " + arq.getName());
      System.out.println("Caminho absoluto: " +
                              arq.getAbsolutePath());
      System.out.println("Tamanho:"+arq.length()+"bytes");
      long horaMod = arq.lastModified();
      System.out.println("Hora de modificacao: " +
                                          new Date(horaMod));
    }
   }
}
```

# File: ações comuns

- Listar o conteúdo de um diretório
  - String[] list()
  - File[] listFiles()
- Criar um novo arquivo
  - boolean createNewFile()
- Criar novos diretórios
  - boolean mkdir()
  - boolean mkdirs()

# File: ações comuns

- Mudar nome de um arquivo ou diretório
  - boolean renameTo(File arq)
- Remover um arquivo ou diretório
  - boolean delete()
  - Um diretório deve estar vazio para que possa ser removido!

# Exemplo: conteúdo de um diretório

```
public class ListarDiretorio {
  public static void main(java.lang.String[] args) {
    String nomeDiretorio = ".";

    File dir = new File(nomeDiretorio);
    if (!dir.exists()) {
      System.out.println("Diretorio " +
              dir.getAbsolutePath() + " nao encontrado");
    } else if (dir.isDirectory()) {
      String[] listaArquivos = dir.list();
      if (listaArquivos != null) {
        for(int i=0; i < listaArquivos.length; i++)
          System.out.println(listaArquivos[i]);
      }
    }
  }
}
```

# File: independência do sistema

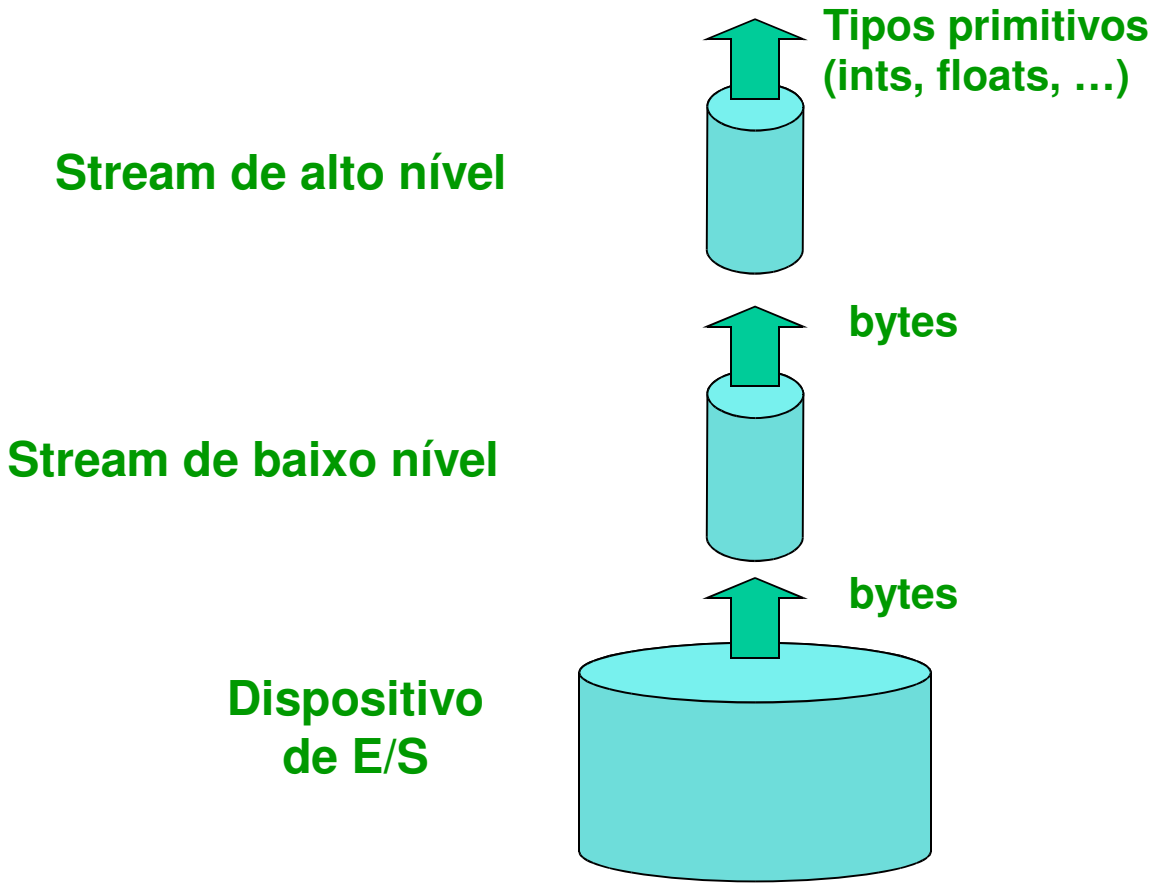
- O separador de arquivos pode não ser “\”
  - Use **File.separator**
- O diretório atual pode não ser “.”
  - use **System.getProperty("user.dir")**
- O diretório pai pode não ser “..”
  - use o método **getParent()** da classe File
- O diretório de arquivos temporários pode não ser “c:\windows\temp”:
  - Use **File.createTempFile()**

# Streams

# Streams

- Streams são seqüências de bytes
- São lidos de forma seqüencial
- Streams estão sempre associados a “fontes de dados”
  - Arquivos, outros streams, áreas de memória

# Tipos de streams



# Streams de baixo nível

- Leitura e escrita de bytes sem processamento
- Estendem diretamente **InputStream** ou **OutputStream**
- Mais importantes
  - FileInputStream
  - FileOutputStream

# Streams de alto nível

- Processamento dos bytes lidos/escritos
- Leitura e escrita de tipos primitivos e objetos
- Lêem e escrevem **em outros streams**
- Mais importantes:
  - BufferedInputStream, BufferedOutputStream
  - DataOutputStream, DataInputStream
  - ObjectOutputStream, ObjectInputStream
- Streams de alto nível não lêem nem escrevem diretamente em dispositivos de entrada e saída

# Streams com filtros

- Fazem algum processamento nos bytes lidos/escritos
- Subclasses de **FilterInputStream** e **FilterOutputStream**
- Mais importantes
  - DataInputStream, DataOutputStream
    - Leitura e escrita de tipos primitivos
  - BufferedInputStream, BufferedOutputStream
    - Leitura e escrita de bytes com buffering (para eficiência)
  - ObjectInputStream, ObjectOutputStream
    - Leitura e escrita de objetos

# Exemplo com DataOutputStream

```
Cliente[] clientes = { new Cliente("111","Joao da Silva"),
                new Cliente("222","Carlos Filho"),
                new Cliente("333","Paulo Cornelli")};
   try {
     String tempDir = System.getProperty("user.home");
     FileOutputStream file = new FileOutputStream(tempDir
           + File.separator + "clientes.txt");
     DataOutputStream dos = new DataOutputStream(file);

     for (int i=0; i < clientes.length; i++) {
       dos.writeBytes(clientes[i].getCpf());
       dos.writeByte('\n');
       dos.writeBytes(clientes[i].getNome());
       dos.writeByte('\n');
     }
     dos.close();
     file.close();
     System.out.println("Arquivo salvo em: " + tempDir);
```

# Exemplo com DataInputStream

```
FileInputStream fileIn = new FileInputStream(tempDir
          + File.separator + "clientes.txt");
DataInputStream dis = new DataInputStream(fileIn);
String line = dis.readLine();

while(line != null) {
  System.out.println("CPF: " + line
            + " - Nome: " + dis.readLine());
  line = dis.readLine();
}
```

# Exemplo: leitura de um arquivo de propriedades

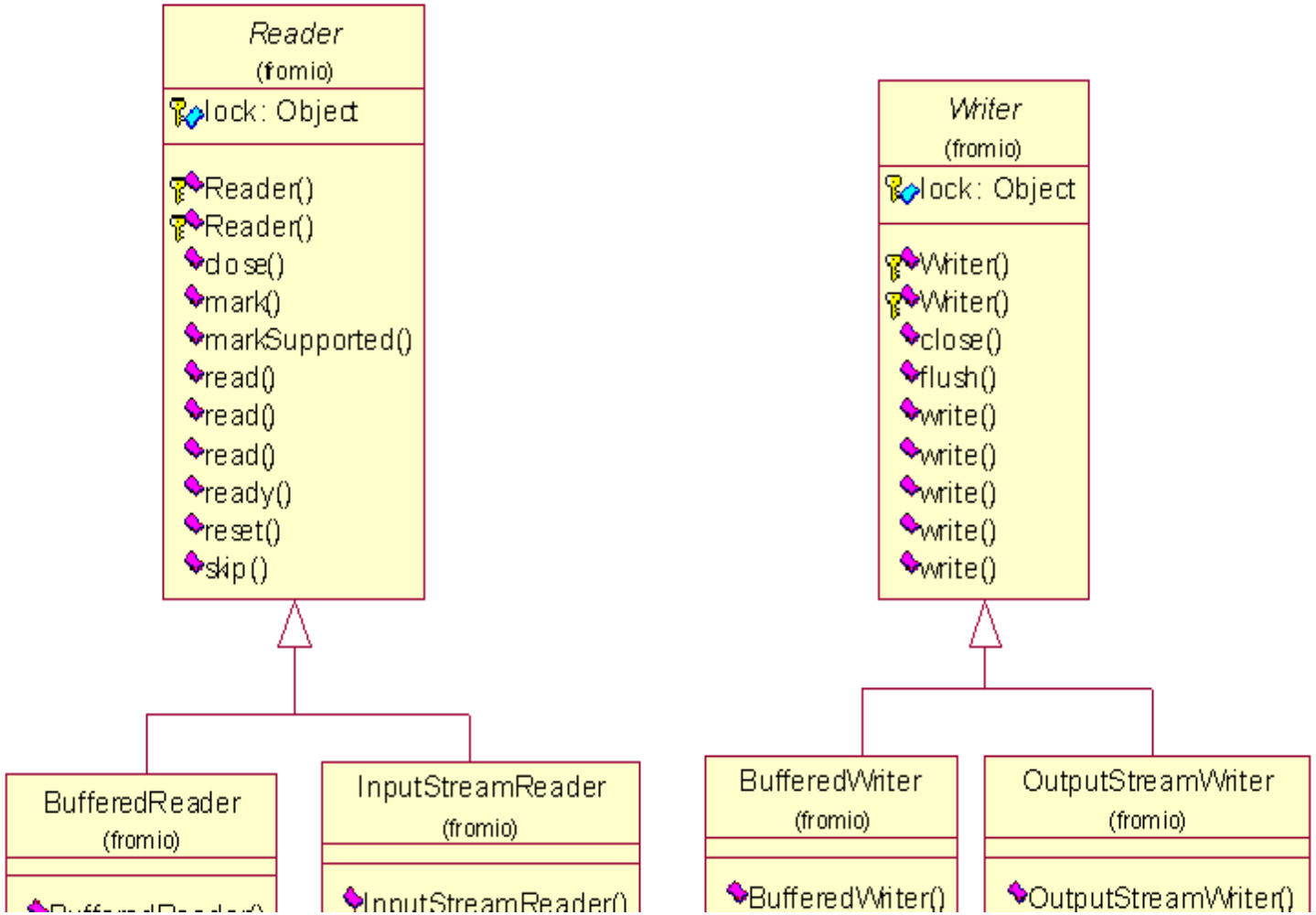
```
public class LerPropriedades {
  public static void main(String[] args) {
    try{
      String arquivo = "sistema.properties";
      Properties p = new Properties();
      FileInputStream fin =
              new FileInputStream(arquivo);
      p.load(fin);
      p.list(System.out);
    }catch(Exception e){
      e.printStackTrace();
    }
  }
}
```

# Readers e Writers

# Readers e Writers

- São streams que se diferenciam dos demais por lidar com caracteres Unicode
- Devem ser usados para leitura de caracteres (mais completos e eficientes)
- Readers/Writers de baixo nível
  - FileReader
  - FileWriter
- Readers/Writers de alto nível
  - BufferedReader
  - BufferedWriter
  - PrintWriter

# Hierarquia de Readers e Writers



# Exemplo: listar um arquivo texto

```
public static void lerArquivo(InputStream is)
      throws IOException{

  InputStreamReader ir = new InputStreamReader(is);
  //Buffer para leitura linha a linha
  BufferedReader br = new BufferedReader(ir);

  String linha = br.readLine();
  while (linha != null) {
    System.out.println(linha);
    linha = br.readLine();
  }
}
```

# Exemplo: escrita em arquivo texto

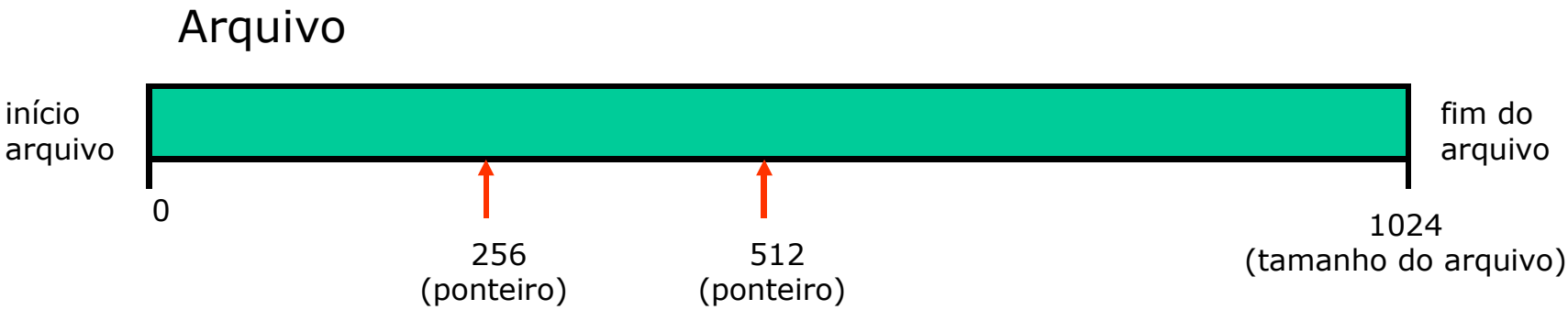
```
public static void escreverArquivo(String name)
       throws IOException{

     FileWriter fout = new FileWriter(name);
     BufferedWriter bw = new BufferedWriter(fout);
     bw.write("texto");
     bw.close();
     fout.close();
 }
```

# Acesso aleatório

# RandomAccessFile

- Define um ponteiro para acessar posições aleatórias do arquivo, relativas ao início



# RandomAccessFile

- Construtores
  - RandomAccessFile(String file, String mode)
  - RandomAccessFile(File file, String mode)
- A criação de um objeto RandomAccessFile pode gerar uma FileNotFoundException (modo de leitura) ou criar o arquivo (modo de escrita) caso o arquivo não exista

# RandomAccessFile: navegação

- long getFilePointer
  - retorna a posição atual do ponteiro
- long length()
  - retorna o tamanho do arquivo em bytes
- void seek(long position)
  - posiciona o ponteiro no arquivo

# RandomAccessFile: leitura e escrita

- Métodos para leitura e escrita de tipos de dados comuns

| Tipo de dado | Métodos | |
|---|---|---|
| | Leitura | Escrita |
| boolean | boolean readBoolean() | void writeBoolean(boolean b) |
| byte | byte readByte() | void writeByte(byte b) |
| char | char readChar() | void writeChar(char c) |
| double | double readDouble() | void writeDouble(double d) |
| float | float readFloat() | void writeFloat(float f) |
| int | int readInt() | void writeInt(int i) |
| long | long readLong() | void writeLong(long l) |
| short | short readShort() | void writeShort(short s) |
| String | String readLine() | - |

# RandomAccessFile: exemplo

```
//escrever
RandomAccessFile rf =  new RandomAccessFile("teste.dat", "rw");
for(int i = 0; i < 10; i++) {
  double num = i*3.00;
  rf.writeDouble(num);
  System.out.println(num);
}

//ler
int bytesLidos = 0;
long totalBytes = rf.length();
rf.seek(bytesLidos);
while(bytesLidos*8 < totalBytes) {
  double num = rf.readDouble();
  System.out.println(num);
  bytesLidos++;
  rf.seek(bytesLidos*8);
}
```

# Serialização de Objetos

# Serialização de Objetos

- Objetos podem ser lidos e escritos no disco
  - **ObjectOutputStream**
    - void writeObject(Object objeto)
  - **ObjectInputStream**
    - Object readObject()
- Classes devem implementar a interface **Serializable** para que seus objetos possam ser escritos/lidos

# Exemplo de Serialização

```
public static void gravarCliente(Cliente c, String arquivo)
   throws FileNotFoundException, IOException {

   FileOutputStream fos = new FileOutputStream(arquivo);
   ObjectOutputStream oos = new ObjectOutputStream(fos);
   oos.writeObject(c);
 }

 public static Cliente lerCliente(String arquivo)
   throws     FileNotFoundException,
              IOException,ClassNotFoundException {

   FileInputStream fis = new FileInputStream(arquivo);
   ObjectInputStream ois = new ObjectInputStream(fis);
   Cliente c = (Cliente) ois.readObject();
   return c;
 }
```

# Mais sobre serialização

- Se um objeto tem referências a outros objetos, esses objetos também são serializados
- Um atributo de um objeto pode ser marcado como não-serializável com a palavra-chave **transient**

```
import java.io.*;

public class Usuario implements Serializable
{
  private String ID;
  private HistoricoAcessos historico;
  private transient String senha;
}
```

`historico` será serializado somente se implementar `Serializable`

O atributo `senha` não será serializado

# Mais sobre serialização

- Objetos e suas referências formam um grafo
- Para que um objeto possa ser serializado, ele e o resto dos objetos do grafo de objetos devem implementar Serializable
  - java.io.NotSerializableException