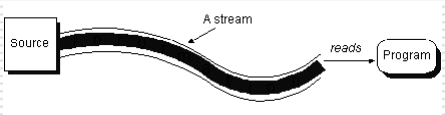
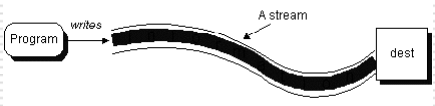
Ficheiros

- Os objectos e tipos básicos até agora utilizados têm uma característica comum: são armazenadas na memória central do computador
- Isto significa que só existem durante a execução do programa, pelo que os dados que armazenam apenas estão acessíveis durante esse período
- Esta situação é inaceitável em muitas situações
- É necessário dispor e utilizar dispositivos de armazenamento fixo (por exemplo discos duros) que mantenham a informação (em ficheiros) para além do final da execução do programa que a manipula.

Ficheiros



Stream para leitura de dados



Stream para escrita de dados

Ficheiros

- O Java inclui diversas classes destinadas a manipular ficheiros (incluídas na package java.io)
- A classe File serve para modelar ficheiros em disco
- Para criar um objecto File:

```
File f1 = new File (nomeDoFicheiro);
```

- O nomeDoFicheiro deve incluir toda a estrutura de directórios necessária para o localizar
- A criação de um objecto File não garante a criação de um ficheiro correspondente em disco, serve apenas para o representar logicamente
- Se o ficheiro correspondente existir, o objecto File fornece alguns métodos para o manipular

Ficheiros

- Um deles permite apagar um ficheiro:

```
File f = new File ("lixo");
f.delete();
```

- Outro permite verificar se um dado ficheiro existe:

```
File f1 = new File("c:/Exemplos/UmFicheiro.txt");
if (f1.exists()) {
    System.out.print("Ficheiro existe");
} else {
    System.out.print("Ficheiro não existe");
}
```

- Esta classe não possui métodos para criar um ficheiro ou para ler (escrever) de (para) um ficheiro

Ficheiros de texto

- Para que isso seja possível (ou para re-inicializar um ficheiro já existente) é necessário estabelecer um caminho de saída dirigido para esse ficheiro
- No caso dos ficheiros de texto, podem ser utilizadas as classes FileReader ou FileWriter, consoante a operação desejada seja de leitura (*reader*) ou de escrita (*writer*)

Ficheiros de texto

- Para criar objectos destas classes, é necessário fornecer como parâmetro um objecto da classe File correspondente ao ficheiro pretendido:

```
FileReader frd = new FileReader(new File(nomeDoFicheiro));
FileWriter fwt = new FileWriter(new File(nomeDoFicheiro));
```

- Ou, de forma abreviada:

```
FileReader frd = new FileReader(nomeDoFicheiro);
FileWriter fwt = new FileWriter(nomeDoFicheiro);
```

- Assim que a stream é construída o ficheiro de texto é criado se não existia antes ou os seus conteúdos removidos se o ficheiro já existia

Ficheiros de texto

- Estas classes são específicas de ficheiros de caracteres, permitindo a sua leitura ou escrita carácter a carácter
- A leitura ou escrita de um carácter de cada vez não é muito conveniente na maior parte das vezes
- Por esse motivo, faz-se apelo a um terceiro objecto, de forma a conseguir a leitura e a escrita linha a linha
- As classes em causa são a BufferedReader e a BufferedWriter, que recebem um objecto da classe FileReader e FileWriter, respectivamente, como parâmetro:

```
BufferedReader fR = new BufferedReader(frd);
BufferedWriter fW = new BufferedWriter(fwt);
```

- Estas classes têm métodos convenientes para a leitura e escrita de linhas de caracteres, tornando-se, por isso, mais convenientes para a manipulação de ficheiros.

Ficheiros de texto

- Utilizando as classes anteriores, é possível criar uma nova classe que permite manipular este tipo de ficheiros através de uma interface simplificada.
- A utilização de ficheiros de texto envolve essencialmente quatro operações: abertura, leitura, escrita e fecho.
- Esta nova classe deve então implementar estes quatro comportamentos.

Ficheiros de texto

- Para poder funcionar devidamente, esta classe deve ter como atributos uma referência para um objecto da classe BufferedReader e outra para um objecto da classe BufferedWriter:

```
import java.io.*;
public class FicheiroDeTexto
{
    private BufferedReader fR;
    private BufferedWriter fW;
}
```


Ficheiros de texto

- Podem agora ser adicionados dois métodos de abertura do ficheiro

```
public void abreLeitura(String nomeDoFicheiro) throws
    IOException
{
    fR = new BufferedReader(new
    FileReader(nomeDoFicheiro));
}
public void abreEscrita(String nomeDoFicheiro) throws
    IOException
{
    fW = new BufferedWriter(new
    FileWriter(nomeDoFicheiro));
}
```

Ficheiros de texto

- As palavras reservadas throws IOException são obrigatórias no cabeçalho de todos os métodos que incluam operações de entrada/saída ou que chamem métodos que as incluam
- A sua função é indicar ao compilador que o método pode gerar ou propagar um erro do tipo IOException, que se verifica, por exemplo, quando se tenta abrir para leitura um ficheiro inexistente.

Ficheiros de texto

- A leitura de uma linha a partir de um ficheiro de texto pode ser feita utilizando o método readLine() da classe BufferedReader:

```
//Método para ler uma linha do ficheiro
//Devolve a linha lida
public String lerLinha() throws IOException
{
    return fR.readLine();
}
```

Ficheiros de texto

- É comum utilizar ficheiros de texto para armazenar representações de números, um em cada linha.
- Internamente os números são armazenados como cadeias de caracteres, pelo que após a leitura de uma linha é necessário converter a cadeia de caracteres obtida para um número.
- Para o caso de números inteiros:

```
//Método para ler um número do ficheiro
//Devolve o número lido
public int[] lerNumeroInt() throws IOException  {
    int[] result = new int[2];
    String st = fR.readLine();
    if (st != null) {
            result[0] = 0;
            result[1] = Integer.parseInt(st);
    } else {
            result[0] = -1;}
    return result;
}
```


Ficheiros de texto

- A escrita de uma cadeia de caracteres num ficheiro de texto pode ser obtida com o método:

```
//Método para escrever uma linha no ficheiro
//Recebe a linha a escrever
public void escreverLinha(String linha) throws
    IOException
{
    fW.write(linha,0,linha.length());
    fW.newLine();
}
```

Ficheiros de texto

- Tal como para a leitura, é útil criar um método que permita escrever um número inteiro num ficheiro:

```
//Método para escrever um número inteiro no ficheiro
//Recebe o número a escrever
public void escreverNumero(int num) throws
    IOException
{
    String st = "";
    st = st.valueOf(num);
    escreverLinha(st);
}
```


Ficheiros de texto

- Os métodos para fechar ficheiros são muito simples, pois limitam-se a utilizar o método close() das classes BufferedReader e BufferedWriter.
- Incluem-se na classe FicheiroDeTexto apenas para manter a consistência da sua interface:

```
//Método para fechar um ficheiro aberto em modo leitura
public void fechaLeitura() throws IOException {
    fR.close();
}
//Método para fechar um ficheiro aberto em modo escrita
public void fechaEscrita() throws IOException {
    fW.close();
}
```

Ficheiros de texto

- Vamos criar um programa que crie um ficheiro de texto contendo os quadrados de todos os números menores que 10. A execução do programa resulta na criação, no directório corrente, de um ficheiro de nome “teste.txt”.

```
public class ExemploFicheiros
{
    public static void main(String args[]) throws IOException
    {
            FicheiroDeTexto f = new FicheiroDeTexto();
            f.abreEscrita("teste.txt");
            for (int i=1;i<10;i++)
                    f.escreverNumero(i*i);
            f.fechaEscrita();
    }

}
```


Ficheiros de objectos

- A leitura e a escrita de objectos de e para ficheiros são asseguradas pelas classes ObjectInputStream e ObjectOutputStream
- A classe ObjectOutputStream, através do seu método writeObject(), organiza os dados do objecto, de modo a que possam ser enviados sequencialmente para o ficheiro através do fluxo de saída de dados
- A classe ObjectInputStream, através do método readObject(), recolhe os dados do fluxo de entrada e reorganiza-os, de forma a reconstruir um objecto igual ao inicialmente escrito
- Estes métodos podem trabalhar com qualquer classe de objectos predefinidos na linguagem, como sejam as cadeias de caracteres ou os ArrayList

Ficheiros de objectos

- Para armazenar em ficheiro objectos de classes não pré-definidas, é necessário indicar que se autoriza a sua reorganização para armazenamento em ficheiro.
- Para isso é preciso declarar que essas classes implementam a interface Serializable, por exemplo:

  public class **Turma** implements Serializable

- Este cabeçalho permite que os objectos da classe Turma possam ser fornecidos ao método writeObject() para serem enviados para um ficheiro

Ficheiros de objectos

- O método readObject() pode ser usado para ler dados do ficheiro e construir o objecto correspondente em memória central.
- É importante notar que apenas as variáveis de instância (atributos) de um objecto são incluídas no ficheiro. As variáveis globais ou de classe que um objecto utilize não são incluídas no ficheiro.

Ficheiros de objectos

- Nos ficheiros de texto, foram utilizadas as classes FileReader e FileWriter para estabelecer os fluxos de dados, uma vez que se tratava apenas de caracteres
- Neste caso, a utilização destas classes não é adequada. Em sua substituição devem ser utilizadas as classes FileInputStream e FileOutputStream:

```
FileInputStream is = new FileInputStream(new
    File(nomeDoFicheiro));
FileOutputStream os = new FileOutputStream(new
    File(nomeDoFicheiro));
```

- Ou:

```
FileInputStream is = new
    FileInputStream(nomeDoFicheiro);
FileOutputStream os = new
    FileOutputStream(nomeDoFicheiro);
```


Ficheiros de objectos

- Agora é possível criar objectos para manipular os ficheiros de objectos:

```
ObjectInputStream ois = new ObjectInputStream(is);
ObjectOutputStream oos = new ObjectOutputStream(os);
```

- Tal como para os ficheiros de texto, pode ser desenvolvida uma classe que simplifique a utilização de ficheiros de objectos
  - Esta classe, que vai ser denominada FicheiroDeObjectos, incluirá métodos para abrir um ficheiro, escrever um objecto num ficheiro, ler um objecto a partir de um ficheiro e fechar um ficheiro

Ficheiros de objectos

- A classe terá como atributos referências para um objecto da classe ObjectInputStream e para um objecto da classe ObjectOutputStream:

```
import java.io.*;
public class FicheiroDeObjectos
{
    private ObjectInputStream iS;
    private ObjectOutputStream oS;
}
```

- Os métodos de abertura do ficheiro para leitura e para escrita podem ser:

```
//Método para abrir um ficheiro para leitura
public void abreLeitura(String nomeDoFicheiro) throws
    IOException {
    iS = new ObjectInputStream(new
    FileInputStream(nomeDoFicheiro));
}
```


Ficheiros de objectos

```
//Método para abrir um ficheiro para escrita
//Recebe o nome do ficheiro
public void abreEscrita(String nomeDoFicheiro) throws
    IOException
{
    oS = new ObjectOutputStream(new
    FileOutputStream(nomeDoFicheiro));
}
```

- A leitura de um objecto a partir de um ficheiro pode ser efectuada através do método readObject() da classe ObjectInputStream:

```
//Método para ler um objecto do ficheiro
//Devolve o objecto lido
public Object leObjecto() throws IOException,
    ClassNotFoundException
{
    return iS.readObject();
}
```

Ficheiros de objectos

- Este método devolve um resultado Object, de modo a poder ler qualquer tipo de objecto. Caberá ao método que o chamar concretizar, através de um *cast*, qual a classe a que esse objecto deve pertencer.
- Notar a indicação de que o método pode propagar um erro do tipo ClassNotFoundException. Este erro será gerado pelo método readObject() se o ficheiro não contiver objectos.

Ficheiros de objectos

- A escrita de um objecto num ficheiro pode ser obtida com o método:

```
//Método para escrever um objecto no ficheiro
//Recebe o objecto a escrever
public void escreveObjecto(Object o) throws IOException
{
    oS.writeObject(o);
}
```

Ficheiros de objectos

- Os métodos para fechar os ficheiros limitam-se a utilizar o método close() das classes ObjectInputStream e ObjectOutputStream:

```
//Método para fechar um ficheiro aberto em modo leitura
public void fechaLeitura() throws IOException
{
    iS.close();
}

//Método para fechar um ficheiro aberto em modo escrita
public void fechaEscrita() throws IOException
{
    oS.close();
}
```


Tratamento de excepções

- As palavras reservadas throws IOException no cabeçalho de um método servem para indicar ao compilador que ele pode gerar ou propagar um erro do tipo IOException.
- Essa utilização pode ser evitada se o método tratar internamente as excepções (erros) que possam ocorrer, evitando assim a sua propagação
- Para isso enquadram-se as instruções potencialmente geradoras de excepções num bloco try ao qual se segue um ou mais blocos catch que indicam o procedimento a seguir caso o erro ocorra.

Tratamento de excepções

- O bloco try ... catch tem a seguinte sintaxe:

```
try {
    //Código que faz a acção desejada, mas pode gerar um erro
} catch (ExceptionType1 e1){
    //Instruções a executar se ocorrer uma excepção do tipo ExceptionType1
} catch (ExceptionType2 e2){
    //Instruções a executar se ocorrer uma excepção do tipo ExceptionType2
}
```


Tratamento de excepções

- Qualquer método que contenha instruções potencialmente geradoras de erros ou que chame métodos que possam propagar erros deve:
  - Declarar a possibilidade de propagar erros (através da inclusão de throws no seu cabeçalho)
  - Tratar as eventuais excepções, utilizando try ... catch
- O tratamento de excepções deve ser feito em algum ponto do programa (o tratamento feito pelo sistema operativo consiste em terminar o programa abruptamente)

Tratamento de excepções

- Exemplo:

```
public static int readInt() {
  while (true) {
    try {
       return
    Integer.valueOf(readString().trim()).intValue();
    } catch (Exception e) {
            System.out.println ("!!!Not an integer !!!");
    }
  }
}
```