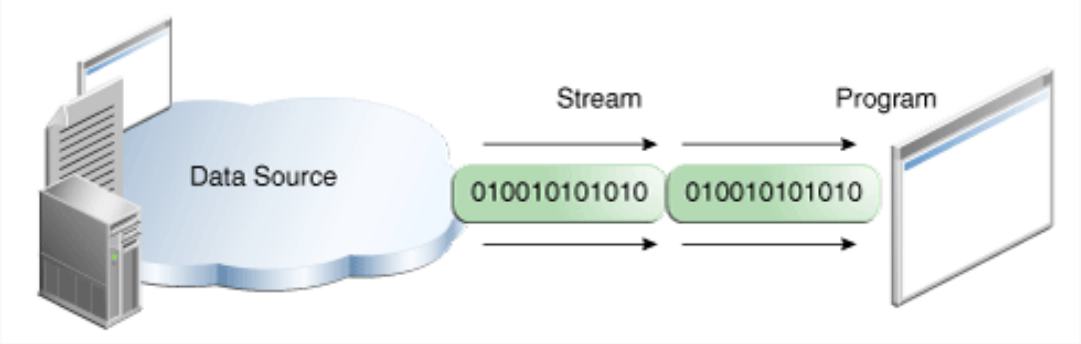
OPERAÇÕES DE ENTRADA E SAÍDA

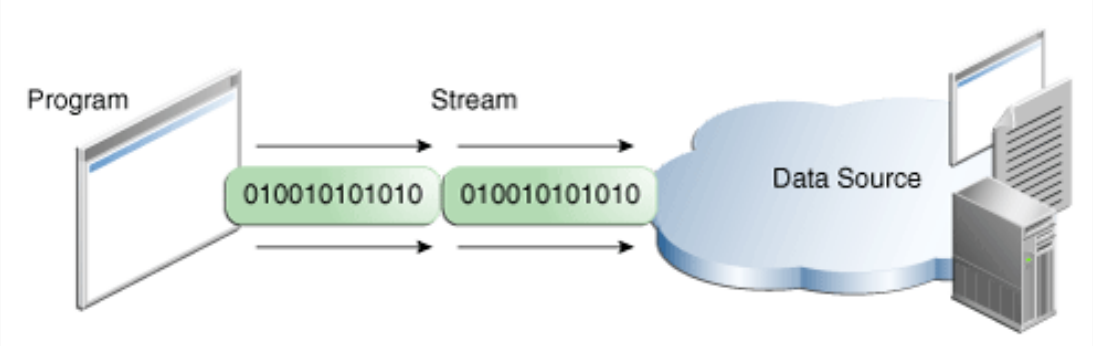
ACH 2003 — COMPUTAÇÃO ORIENTADA A OBJETOS

Daniel Cordeiro

Escola de Artes, Ciências e Humanidades | EACH | USP

FLUXOS DE E/S





Vimos:

- fluxos de bytes (*byte streams*)
- fluxos de caracteres (*character streams*)

E/S POR LINHAS

- É incomum manipularmos textos caractere a caractere
- Normalmente o fazemos por *linhas*
- O término de uma linha[1] é indicado por: uma sequência de *carriage-return* + *line-feed* (`"\r\n"`), um único *carriage-return* (`"\r"`), ou um único *line-feed* (`"\n"`).

[1] Recomendo a leitura de `https://en.wikipedia.org/wiki/Newline` sobre as diferentes representações de uma quebra de linha usadas por sistemas operacionais.

CÓPIA LINHA A LINHA

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

public class CopyLines {
    public static void main(String[] args) throws IOException {
        BufferedReader inputStream = null;
        PrintWriter outputStream = null;

        try {
            inputStream = new BufferedReader(new FileReader("xanadu.txt"));
            outputStream = new PrintWriter(new FileWriter("characteroutput.txt")

            String l;
            while ((l = inputStream.readLine()) != null) {
                outputStream.println(l);
            }
        } finally {
            if (inputStream != null) {
                inputStream.close();
            }
            if (outputStream != null) {
                outputStream.close();
            }
    }   }
}
```

BUFFERED STREAMS

- Operações de E/S com e sem uso de *buffer*
  - **sem uso** cada operação de leitura e escrita é realizada diretamente pelo sistema operacional
  - **com uso** dados são lidos de/escritos para uma área de memória (o *buffer* e só quando o *buffer* estiver vazio/cheio é que uma chamada ao sistema operacional é realizada
- Um programa pode transformar um fluxo sem *buffer* em um com *buffer* usando uma expressão idiomática (*idiom*) de orientação a objetos chamada *wrapping* (invólucro)

Para usar buffers no CopyCharacter:

```
inputStream = new BufferedReader(
                      new FileReader("xanadu.txt"));
outputStream = new BufferedWriter(
                      new FileWriter("charoutput.txt"));
```

BUFFERED STREAMS

Há quatro classes para embrulhar fluxos sem *buffers*:

- `BufferedInputStream` e `BufferedOutputStream` para criar fluxos de byte com *buffers*
- `BufferedReader` e `BufferedWriter` para criar fluxos de caracteres com *buffers*

Flushing

- às vezes faz sentido querer gravar os dados do *buffer* antes que ele encha; chamamos isso de *flush*
- alguns fluxos com *buffer* possuem uma opção de `autoflush`, especificada no construtor, que realiza o *flush* após certas ações (ex: após o uso de `println` em `PrintWriter`)
- para realizar manualmente o *flush* de um fluxo, execute seu método `flush()`

SCANNING

- Objetos do tipo `Scanner` são usados para quebrar uma entrada bem definida em "pedaços"[2] (*tokens*)
- Por padrão, cada pedaço é separado por espaços em branco (o que inclui os caracteres de espaço, tabs e de quebras de linha)

[2] Ao pé da letra, a tradução de *token* seria símbolo, ficha, código, etc.

```java
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")));
          // o código abaixo parece familiar?
          while (s.hasNext()) {
              System.out.println(s.next());
          } // Scanner implementa Iterator<String>
      } finally {
          if (s != null) {
              s.close();
          }
  }   }
}
```

Note que:

- você deve chamar o método `close` do `Scanner`; mesmo que ele não seja um fluxo, você deve indicar que o `Scanner` pode fechar o fluxo que utilizou.
- para usar um delimitador diferente, use o método `useDelimiter()`.

  Ex: `s.useDelimiter(",\\s*");` (vírgula seguida de um ou mais espaços)

CONVERSÃO DE TOKENS EM VALORES

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

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("usnumbers.txt")));
      s.useLocale(Locale.US); // lá 32,767 é um inteiro, aqui é decimal

      while (s.hasNext()) {
        if (s.hasNextDouble()) {
          sum += s.nextDouble(); // um String é convertido para um double
        } else {
          s.next();
        }
      }
    } finally {
        s.close();
    }

    System.out.println(sum);
  }
}
```

FORMATAÇÃO

- Formatação é a transformação de um tipo primitivo em uma representação como String
- Fluxos que implementam formatação são instâncias de `PrintWriter`, um fluxo de caracteres, ou `PrintStream`, fluxo de bytes
- `System.out` e `System.err` são provavelmente os únicos `PrintStream` que vocês irão utilizar
- Use sempre `PrintWriter` para formatar sua saída

Métodos para formatação

- `print` and `println` formatam valores individuais
- `format` formata vários valores usando uma string com o formato preciso

PRINT E PRINTLN

```java
public class Root {
    public static void main(String[] args) {
        int i = 2;
        double r = Math.sqrt(i);

        System.out.print("A raiz quadrada de ");
        System.out.print(i);
        System.out.print(" é ");
        System.out.print(r);
        System.out.println(".");

        i = 5;
        r = Math.sqrt(i);
        System.out.println("A raiz quadrada de " + i + " é " + r + ".");
    }
}
```

Saída:

```
A raiz quadrada de 2 é 1.4142135623730951.
A raiz quadrada de 5 é 2.23606797749979.
```

FORMAT

```java
public class Root2 {
    public static void main(String[] args) {
        int i = 2;
        double r = Math.sqrt(i);

        System.out.format("A raiz quadrada de %d é %f.%n", i, r);
    }
}
```

Saída:
A raiz quadrada de 2 é 1.414214.

- **d** formata um valor inteiro como decimal
- **f** formata um valor de ponto flutuante como decimal
- **n** imprime a quebra de linha da plataforma
- **x** formata um inteiro como hexadecimal
- **s** formata qualquer valor como String
- **tB** formata um inteiro como o nome de um mês no idioma definido

`https://docs.oracle.com/javase/8/docs/api/java/util/Formatter.html#syntax`

Fluxos padrão:

entrada padrão: `System.in`
saída padrão: `System.out`
saída de erro padrão: `System.err`

- por razões históricas, `System.out` e `System.err` são fluxos de bytes e não de caracteres, apesar de se comportarem como fluxos de caracteres internamente
- `System.in` não se comporta como fluxo de caracteres, para usá-lo é preciso embrulhá-lo[3]:

```
InputStreamReader cin =
        new InputStreamReader(System.in);
```

[3] As classes `InputStreamReader` e `OutputStreamWriter` adaptam fluxos de bytes a fluxos de caracteres e fluxos de caracteres a fluxos de bytes, respectivamente.

CONSOLE

- Em Java $\geq$ 1.6 a classe `Console` provê métodos para iteração com usuários
- Possui métodos para leitura segura de senhas na linha de comando
- Fornece fluxos **de caracteres** de entrada e saída obtidos pelos métodos `reader()` e `writer()`
- Uma instância do console deve ser obtida pelo método `System.console()`, que pode devolver `null` caso as operações no console não sejam permitidas

CONSOLE – USO

```
import java.io.Console; import java.util.Arrays; import java.io.IOException;

public class Password {   // Troca a senha do usuário
  public static void main (String args[]) throws IOException {
    Console c = System.console();
    if (c == null) { System.err.println("No console."); System.exit(1); }

    String login = c.readLine("Enter your login: ");
    char [] oldPassword = c.readPassword("Enter your old password: ");

    if (verify(login, oldPassword)) {
      boolean noMatch;
      do {
        char [] newPassword1 = c.readPassword("Enter your new password: ");
        char [] newPassword2 = c.readPassword("Enter new password again: ");
        noMatch = ! Arrays.equals(newPassword1, newPassword2);
        if (noMatch) {
            c.format("Passwords don't match. Try again.%n");
        } else {
            change(login, newPassword1);
            c.format("Password for %s changed.%n", login);
        }
        Arrays.fill(newPassword1, ' ');
        Arrays.fill(newPassword2, ' ');  } while (noMatch);
    }
    Arrays.fill(oldPassword, ' ');   }}
```

BIBLIOGRAFIA

- The Java™ Tutorials – Basic I/O: `https://docs.oracle.com/javase/tutorial/essential/io/`