OPERAÇÕES DE ENTRADA E SAÍDA

ACH 2003 — COMPUTAÇÃO ORIENTADA A OBJETOS

Daniel Cordeiro

8 de abril de 2016

Escola de Artes, Ciências e Humanidades | EACH | USP

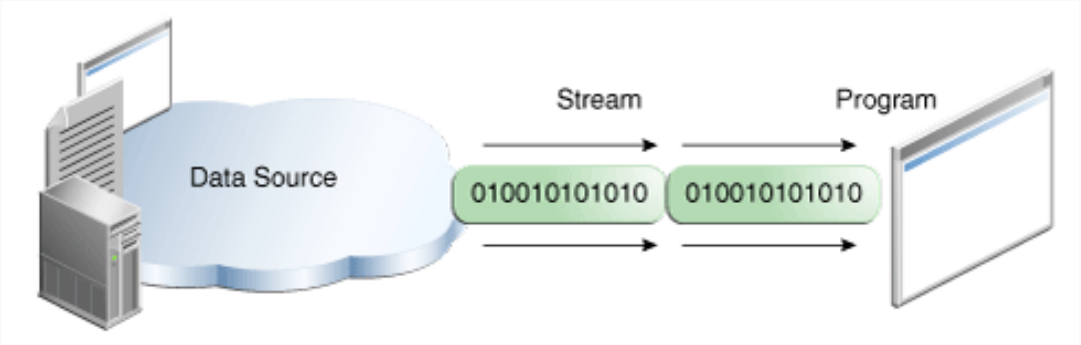
VISÃO GERAL

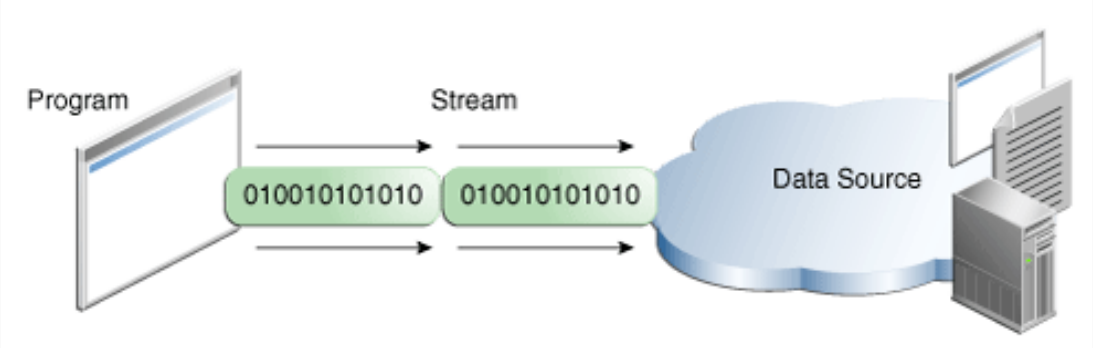
- operações de E/S
- fluxos de E/S (*I/O streams*)
- seriação[1] (*serialization*)

[1] ou serialização

FLUXOS DE E/S

- um fluxo de E/S representa uma fonte de dados ou um destino de saída
- pode representar dispositivos bem diferentes, como arquivos em disco, dispositivos, outros programas e vetores na memória
- abstração simples, um fluxo é uma sequência de dados

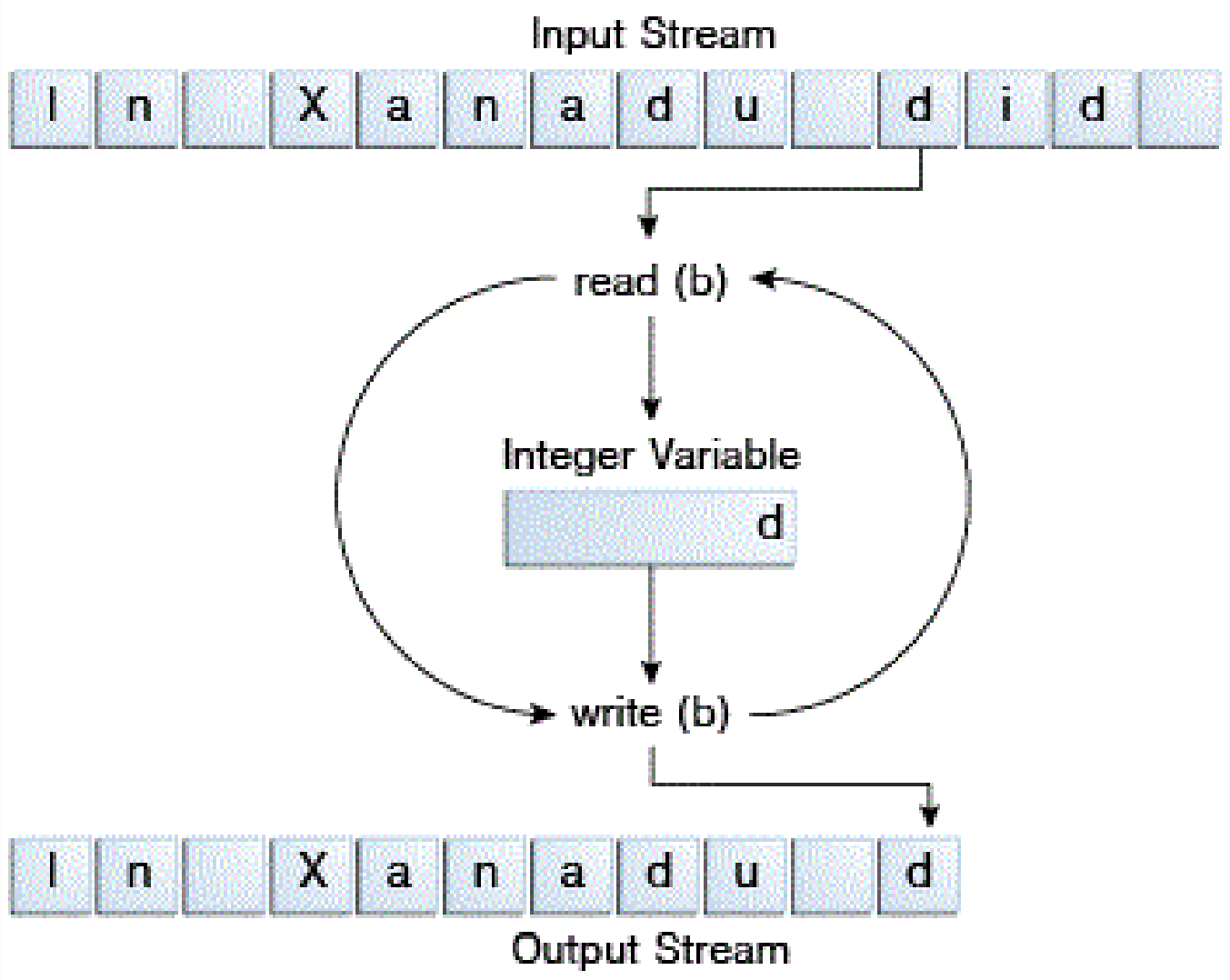




FLUXOS DE BYTES

- programas leem e gravam fluxos de 8-bits
- todos os fluxo de bytes são descendentes de `InputStream` ou `OutputStream`
- existem várias implementações de fluxos de dados, mas as principais talvez sejam os fluxos de E/S de arquivos: `FileInputStream` e `FileOutputStream`

EXEMPLO


Input Stream
I n X a n a d u d i d
read (b)
Integer Variable
d
write (b)
I n X a n a d u d
Output Stream

EXEMPLO

```java
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("outagain.txt");

            int c;
            while ((c = in.read()) != -1) {
                out.write(c);
            }
        } finally {
            if (in != null) { // null se arquivo não existir
                in.close();   // streams devem ser sempre fechados
            }
            if (out != null) {
                out.close();
            }}
}}
```

CHARACTER STREAMS

- byte streams devem ser usados apenas em E/S de baixo nível
- como a entrada é um arquivo texto, fluxos de E/S de caracteres são mais adequados

```
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("characteroutput.txt");

            int c;
            while ((c = inputStream.read()) != -1) {
                outputStream.write(c);
            }
        } finally {
            if (inputStream != null) {
                inputStream.close();
            }
            if (outputStream != null) {
                outputStream.close();
            }
        }}
}
```

Observações:

- `CopyCharacters` e `CopyBytes` são muito parecidos
- a diferença mais importante é o uso de `FileReader` e `FileWriter` ao invés de `FileInputStream` e `FileOutputStream`
- `CopyBytes` manipula um `int` que armazena 8 bits, quando que `CopyCharacters` armazena um caractere Unicode de 16 bits

CHARSET

- um *charset* é uma coleção de caracteres
- um *encoding* é um mapeamento de sequências de bits a um charset
- Java internamente usa o *encoding* UTF-16 em `char[]`, `String` e `StringBuffer`

Encoding	Descrição
US-ASCII	representação de 7 bits, contém os caracteres latinos básicos do Unicode
ISO-8859-1	Alfabeto Latino 1 ISO
UTF-8	Representação de 8 bits
UTF-16BE	Representação de 16 bits *big-endian*
UTF-16LE	Representação de 16 bits *little-endian*
UTF-16	Representação de 16 bits com identificador opcional da ordem do byte

US-ASCII – 7 bits

Dec	Hex		Dec	Hex		Dec	Hex		Dec	Hex		Dec	Hex		Dec	Hex		Dec	Hex	
0	00	NUL	16	10	DLE	32	20		48	30	0	64	40	@	80	50	P	96	60	`	112	70	p
1	01	SOH	17	11	DC1	33	21	!	49	31	1	65	41	A	81	51	Q	97	61	a	113	71	q
2	02	STX	18	12	DC2	34	22	"	50	32	2	66	42	B	82	52	R	98	62	b	114	72	r
3	03	ETX	19	13	DC3	35	23	#	51	33	3	67	43	C	83	53	S	99	63	c	115	73	s
4	04	EOT	20	14	DC4	36	24	$	52	34	4	68	44	D	84	54	T	100	64	d	116	74	t
5	05	ENQ	21	15	NAK	37	25	%	53	35	5	69	45	E	85	55	U	101	65	e	117	75	u
6	06	ACK	22	16	SYN	38	26	&	54	36	6	70	46	F	86	56	V	102	66	f	118	76	v
7	07	BEL	23	17	ETB	39	27	'	55	37	7	71	47	G	87	57	W	103	67	g	119	77	w
8	08	BS	24	18	CAN	40	28	(	56	38	8	72	48	H	88	58	X	104	68	h	120	78	x
9	09	HT	25	19	EM	41	29	)	57	39	9	73	49	I	89	59	Y	105	69	i	121	79	y
10	0A	LF	26	1A	SUB	42	2A	*	58	3A	:	74	4A	J	90	5A	Z	106	6A	j	122	7A	z
11	0B	VT	27	1B	ESC	43	2B	+	59	3B	;	75	4B	K	91	5B	[	107	6B	k	123	7B	{
12	0C	FF	28	1C	FS	44	2C	,	60	3C	<	76	4C	L	92	5C	\	108	6C	l	124	7C	|
13	0D	CR	29	1D	GS	45	2D	-	61	3D	=	77	4D	M	93	5D	]	109	6D	m	125	7D	}
14	0E	SO	30	1E	RS	46	2E	.	62	3E	>	78	4E	N	94	5E	^	110	6E	n	126	7E	~
15	0F	SI	31	1F	US	47	2F	/	63	3F	?	79	4F	O	95	5F	_	111	6F	o	127	7F	DEL

01100010	01101001	01110100	01110011
b	i	t	s

ISO-8859-1 (Latin-1) – 8 bits

- US-ASCII + conjunto de caracteres latinos
- Ex: ª, «, ü, ñ, Ð, etc.
- Ainda assim insuficiente:
  - Œ, œ (francês)
  - Ă, ă, Ș, ș, Ț, ț (romeno)
  - Ẽ, ẽ, Ĩ, ĩ, Ũ, ũ, Ỹ, ỹ, G̃, g̃ (guarani)
  - etc.
- Várias variações (ISO-8859-X) para vários países

Unicode

- *Unicode* é um formato padrão para codificação, representação e manipulação de textos
- formato multi-byte
- UTF-8 mantém compatibilidade com US-ASCII de 8 bits
- UTF-8, UTF-16, UTF-32 usam diferentes quantidades de bytes para representar os caracteres

caractere	encoding	bits
A	UTF-8	01000001
A	UTF-16	00000000 01000001
A	UTF-32	00000000 00000000 00000000 01000001
あ	UTF-8	11100011 10000001 10000010
あ	UTF-16	00110000 01000010
あ	UTF-32	00000000 00000000 00110000 01000010

BIBLIOGRAFIA

- The Java™ Tutorials – Basic I/O: `https://docs.oracle.com/javase/tutorial/essential/io/`
- What every programmer absolutely, positively needs to know about encodings and character sets to work with text: `http://kunststube.net/encoding/`