Binary Input/Output

Slides derived from the work of
Dr. Amy McGovern and Dr. Deborah Trytten

File Abstraction

- Lowest level: Sequence of bytes
  - We refer to this as a ***stream***

- How we interpret each byte (or group of bytes) depends on the context

Data Types

Different primitive data types in Java require different amounts of space

- byte: 1 byte
- short: 2 bytes
- int/float: 4 bytes
- double: 8 bytes

Characters: several options

- ASCII char: 1 byte
- Unicode: 2 bytes
- Extended Unicode: 3 bytes

Dec	Hx	Oct	Char	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL (null)	32	20	040		Space	64	40	100	@	@	96	60	140	`	`
1	1	001	SOH (start of heading)	33	21	041	!	!	65	41	101	A	A	97	61	141	a	a
2	2	002	STX (start of text)	34	22	042	"	"	66	42	102	B	B	98	62	142	b	b
3	3	003	ETX (end of text)	35	23	043	#	#	67	43	103	C	C	99	63	143	c	c
4	4	004	EOT (end of transmission)	36	24	044	$	$	68	44	104	D	D	100	64	144	d	d
5	5	005	ENQ (enquiry)	37	25	045	%	%	69	45	105	E	E	101	65	145	e	e
6	6	006	ACK (acknowledge)	38	26	046	&	&	70	46	106	F	F	102	66	146	f	f
7	7	007	BEL (bell)	39	27	047	'	'	71	47	107	G	G	103	67	147	g	g
8	8	010	BS (backspace)	40	28	050	(	(	72	48	110	H	H	104	68	150	h	h
9	9	011	TAB (horizontal tab)	41	29	051	)	)	73	49	111	I	I	105	69	151	i	i
10	A	012	LF (NL line feed, new line)	42	2A	052	*	*	74	4A	112	J	J	106	6A	152	j	j
11	B	013	VT (vertical tab)	43	2B	053	+	+	75	4B	113	K	K	107	6B	153	k	k
12	C	014	FF (NP form feed, new page)	44	2C	054	,	,	76	4C	114	L	L	108	6C	154	l	l
13	D	015	CR (carriage return)	45	2D	055	-	-	77	4D	115	M	M	109	6D	155	m	m
14	E	016	SO (shift out)	46	2E	056	.	.	78	4E	116	N	N	110	6E	156	n	n
15	F	017	SI (shift in)	47	2F	057	/	/	79	4F	117	O	O	111	6F	157	o	o
16	10	020	DLE (data link escape)	48	30	060	0	0	80	50	120	P	P	112	70	160	p	p
17	11	021	DC1 (device control 1)	49	31	061	1	1	81	51	121	Q	Q	113	71	161	q	q
18	12	022	DC2 (device control 2)	50	32	062	2	2	82	52	122	R	R	114	72	162	r	r
19	13	023	DC3 (device control 3)	51	33	063	3	3	83	53	123	S	S	115	73	163	s	s
20	14	024	DC4 (device control 4)	52	34	064	4	4	84	54	124	T	T	116	74	164	t	t
21	15	025	NAK (negative acknowledge)	53	35	065	5	5	85	55	125	U	U	117	75	165	u	u
22	16	026	SYN (synchronous idle)	54	36	066	6	6	86	56	126	V	V	118	76	166	v	v
23	17	027	ETB (end of trans. block)	55	37	067	7	7	87	57	127	W	W	119	77	167	w	w
24	18	030	CAN (cancel)	56	38	070	8	8	88	58	130	X	X	120	78	170	x	x
25	19	031	EM (end of medium)	57	39	071	9	9	89	59	131	Y	Y	121	79	171	y	y
26	1A	032	SUB (substitute)	58	3A	072	:	:	90	5A	132	Z	Z	122	7A	172	z	z
27	1B	033	ESC (escape)	59	3B	073	;	;	91	5B	133	[	[	123	7B	173	{	{
28	1C	034	FS (file separator)	60	3C	074	<	<	92	5C	134	\	\	124	7C	174			|
29	1D	035	GS (group separator)	61	3D	075	=	=	93	5D	135	]	]	125	7D	175	}	}
30	1E	036	RS (record separator)	62	3E	076	>	>	94	5E	136	^	^	126	7E	176	~	~
31	1F	037	US (unit separator)	63	3F	077	?	?	95	5F	137	_	_	127	7F	177	&#127;	DEL

Source: www.LookupTables.com

Low Level Files

FileInputStream/FileOutputStream

- Byte-level interface to the file
- read() and write() “think” in terms of arrays of bytes to be read and written

Character-Level Interaction

BufferedReader/BufferedWriter (you already have experience with these):

- Read/write individual characters or entire Strings

Buffering

- Storage devices store data in ***blocks*** of bytes (512-2048 bytes in one block)
- Often more efficient to read/write entire blocks than the equivalent size a few bytes at a time
- BufferedReader will read an entire block at once and stores data temporarily in memory
- BufferedWriter stores written data temporarily in memory and then writes the data when a block is complete

- BufferedReader/BufferedWriter example

Data-Level Interaction

Want to store primitive types in the file without having to deal directly at the byte level

- DataInputStream/DataOutputStream

- readShort(), readLong(), readDouble()
- writeShort(), writeLong(), writeDouble()

- Your program must keep track of the sequence that these primitive values are stored in

DataInputStream/DataOutputStream example

Buffering for Primitive Data

We can wrap a BufferedOutputStream around a FileOutputStream

- Many variables may be written to the “file”, but only occasionally are these written out to the disk
- Added method: flush() – forces all buffered bytes to be written immediately

Object-Level Interaction

ObjectInputStream/ObjectOutputStream

- Can read/write entire objects in one call
- Class must implement the *Serializable* interface
  - Marker interface: we don't have to implement anything
- Read/Write is recursive
  - If an object contains references to other objects – they are read/written, too

Object-Level Interaction

Example…

Very Recursive

Collection demonstration

Very Recursive

When we write/read an object:

- All of the contents are written/read, including other objects
- Copy stops before we make multiple copies of objects

- Can keep a variable from being written using the *transient* keyword

More about Serializable

- To be written to an ObjectOutputStream, a class must implement Serializable
- All variables must also be primitive, Serializable or transient
  - Same for their children…
- Class data are not included with objects

End of File

- Finding when you've reached the end of the file is problematic in Java
- Many read() methods will return a special value to indicate that the end of file has been reached
- For DataInputStream and ObjectInputStream: the EOFException will be thrown

Critical Idea

- Must match data format in input and output operations
  - If you write it as a byte, you read it as a byte
  - If you write UTF-8, you read UTF-8
  - If you write characters, you read characters
- Order is important because streams are sequential

Writing/Reading Objects

Code version matters. Imagine:

- File is written using one class definition
- The definition then changes (e.g., add a new instance variable)
- ObjectInputStream will not know how to map the bytes from the old implementation to the new one.
  - Will most likely see an exception