

*Unit 9:*

# **File I/O and File Manipulation**

---

Object-Oriented Programming (OOP)  
CCIT4023, 2025-2026

# U9: File I/O and File Manipulation

- File Input and Output
  - File Stream
  - Text vs. Binary File
- Common File Manipulation
  - With The `JFileChooser` Class
  - Copy, Rename and Delete Files
- Text File Input and Output
- Low-Level File I/O (in Bytes)
- High-Level File I/O (in Primitive Types)
- Object-Level File I/O

# File Input and Output

- Data stored in memory for processing during program execution are not permanent. They will be deleted / released when they are out of scope (e.g. a local variable within a method will be deleted after the method execution) or the program terminates.
- To permanently retain data even after the program execution, we store data in files, such as in secondary storage: hard disks, optical disks
- File data are often categorized into two forms when processing:
  - *Character-based data* stored in a **text file**, in human-readable form
    - Numeric data could be mapped to character symbols with character coding schemes (e.g. ASCII and Unicode), such as most \*.txt files
  - *Byte-based data* stored in a **binary file**, which is better read by programs
    - Data are stored as raw numeric values, often could not be directly mapped to characters symbols, such as image JPEG files
- The advantage of binary files is that very often they are more efficient to process than text files

# File Input and Output

Reference Only

- Example of ASCII code table (e.g. code value 65 is 'A')

Dec	Hex	Oct	Char	Dec	Hex	Oct	Char	Dec	Hex	Oct	Char	Dec	Hex	Oct	Char
0	0	0		32	20	40	[space]	64	40	100	@	96	60	140	`
1	1	1		33	21	41	!	65	41	101	A	97	61	141	a
2	2	2		34	22	42	"	66	42	102	B	98	62	142	b
3	3	3		35	23	43	#	67	43	103	C	99	63	143	c
4	4	4		36	24	44	\$	68	44	104	D	100	64	144	d
5	5	5		37	25	45	%	69	45	105	E	101	65	145	e
6	6	6		38	26	46	&	70	46	106	F	102	66	146	f
7	7	7		39	27	47	'	71	47	107	G	103	67	147	g
8	8	10		40	28	50	(	72	48	110	H	104	68	150	h
9	9	11		41	29	51	)	73	49	111	I	105	69	151	i
10	A	12		42	2A	52	*	74	4A	112	J	106	6A	152	j
11	B	13		43	2B	53	+	75	4B	113	K	107	6B	153	k
12	C	14		44	2C	54	,	76	4C	114	L	108	6C	154	l
13	D	15		45	2D	55	-	77	4D	115	M	109	6D	155	m
14	E	16		46	2E	56	.	78	4E	116	N	110	6E	156	n
15	F	17		47	2F	57	/	79	4F	117	O	111	6F	157	o
16	10	20		48	30	60	0	80	50	120	P	112	70	160	p
17	11	21		49	31	61	1	81	51	121	Q	113	71	161	q
18	12	22		50	32	62	2	82	52	122	R	114	72	162	r
19	13	23		51	33	63	3	83	53	123	S	115	73	163	s
20	14	24		52	34	64	4	84	54	124	T	116	74	164	t
21	15	25		53	35	65	5	85	55	125	U	117	75	165	u
22	16	26		54	36	66	6	86	56	126	V	118	76	166	v
23	17	27		55	37	67	7	87	57	127	W	119	77	167	w
24	18	30		56	38	70	8	88	58	130	X	120	78	170	x
25	19	31		57	39	71	9	89	59	131	Y	121	79	171	y
26	1A	32		58	3A	72	:	90	5A	132	Z	122	7A	172	z
27	1B	33		59	3B	73	;	91	5B	133	[	123	7B	173	{
28	1C	34		60	3C	74	<	92	5C	134	\	124	7C	174	
29	1D	35		61	3D	75	=	93	5D	135	]	125	7D	175	}
30	1E	36		62	3E	76	>	94	5E	136	^	126	7E	176	~
31	1F	37		63	3F	77	?	95	5F	137	_	127	7F	177	

# File Input and Output

Reference Only

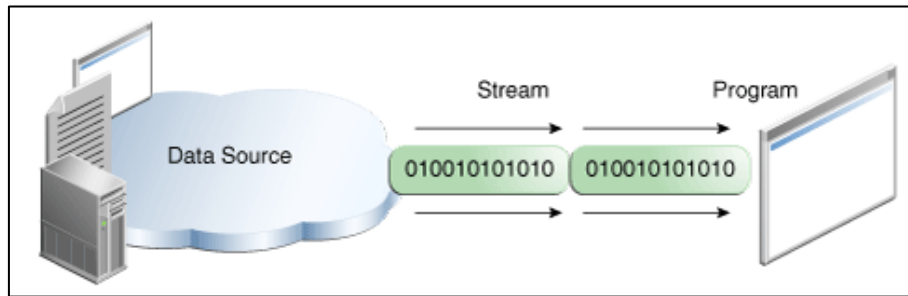
- Example of *partial* Unicode code table

0000	0001	0002	0003	0004	0005	0006	0007	0008	0009	000A	000B	000C	000D	000E	000F	0010	0011	0012	0013	0014	0015	0016	0017	0018	0019	001A	001B	001C	001D	001E	001F	0020	0021	0022	0023	0024	0025	0026	0027	0028	0029	002A	002B	002C	002D	002E	002F	0030	0031	0032	0033	0034	0035	0036	0037	0038	0039	003A	003B	003C	003D	003E	003F	0040	0041	0042	0043	0044	0045	0046	0047	0048	0049	004A	004B	004C	004D	004E	004F	0050	0051	0052	0053	0054	0055	0056	0057	0058	0059	005A	005B	005C	005D	005E	005F	0060	0061	0062	0063	0064	0065	0066	0067	0068	0069	006A	006B	006C	006D	006E	006F	0070	0071	0072	0073	0074	0075	0076	0077	0078	0079	007A	007B	007C	007D	007E	007F	0080	0081	0082	0083	0084	0085	0086	0087	0088	0089	008A	008B	008C	008D	008E	008F	0090	0091	0092	0093	0094	0095	0096	0097	0098	0099	009A	009B	009C	009D	009E	009F	00A0	00A1	00A2	00A3	00A4	00A5	00A6	00A7	00A8	00A9	00AA	00AB	00AC	00AD	00AE	00AF	00B0	00B1	00B2	00B3	00B4	00B5	00B6	00B7	00B8	00B9	00BA	00BB	00BC	00BD	00BE	00BF	00C0	00C1	00C2	00C3	00C4	00C5	00C6	00C7	00C8	00C9	00CA	00CB	00CC	00CD	00CE	00CF	00D0	00D1	00D2	00D3	00D4	00D5	00D6	00D7	00D8	00D9	00DA	00DB	00DC	00DD	00DE	00DF	00E0	00E1	00E2	00E3	00E4	00E5	00E6	00E7	00E8	00E9	00EA	00EB	00EC	00ED	00EE	00EF	00F0	00F1	00F2	00F3	00F4	00F5	00F6	00F7	00F8	00F9	00FA	00FB	00FC	00FD	00FE	00FF	0100	0101	0102	0103	0104	0105	0106	0107	0108	0109	010A	010B	010C	010D	010E	010F	0110	0111	0112	0113	0114	0115	0116	0117	0118	0119	011A	011B	011C	011D	011E	011F	0120	0121	0122	0123	0124	0125	0126	0127	0128	0129	012A	012B	012C	012D	012E	012F	0130	0131	0132	0133	0134	0135	0136	0137	0138	0139	013A	013B	013C	013D	013E	013F	0140	0141	0142	0143	0144	0145	0146	0147	0148	0149	014A	014B	014C	014D	014E	014F	0150	0151	0152	0153	0154	0155	0156	0157	0158	0159	015A	015B	015C	015D	015E	015F	0160	0161	0162	0163	0164	0165	0166	0167	0168	0169	016A	016B	016C	016D	016E	016F	0170	0171	0172	0173	0174	0175	0176	0177	0178	0179	017A	017B	017C	017D	017E	017F	0180	0181	0182	0183	0184	0185	0186	0187	0188	0189	018A	018B	018C	018D	018E	018F	0190	0191	0192	0193	0194	0195	0196	0197	0198	0199	019A	019B	019C	019D	019E	019F	01A0	01A1	01A2	01A3	01A4	01A5	01A6	01A7	01A8	01A9	01AA	01AB	01AC	01AD	01AE	01AF	01B0	01B1	01B2	01B3	01B4	01B5	01B6	01B7	01B8	01B9	01BA	01BB	01BC	01BD	01BE	01BF	01C0	01C1	01C2	01C3	01C4	01C5	01C6	01C7	01C8	01C9	01CA	01CB	01CC	01CD	01CE	01CF	01D0	01D1	01D2	01D3	01D4	01D5	01D6	01D7	01D8	01D9	01DA	01DB	01DC	01DD	01DE	01DF	01E0	01E1	01E2	01E3	01E4	01E5	01E6	01E7	01E8	01E9	01EA	01EB	01EC	01ED	01EE	01EF	01F0	01F1	01F2	01F3	01F4	01F5	01F6	01F7	01F8	01F9	01FA	01FB	01FC	01FD	01FE	01FF	0200	0201	0202	0203	0204	0205	0206	0207	0208	0209	020A	020B	020C	020D	020E	020F	0210	0211	0212	0213	0214	0215	0216	0217	0218	0219	021A	021B	021C	021D	021E	021F	0220	0221	0222	0223	0224	0225	0226	0227	0228	0229	022A	022B	022C	022D	022E	022F	0230	0231	0232	0233	0234	0235	0236	0237	0238	0239	023A	023B	023C	023D	023E	023F	0240	0241	0242	0243	0244	0245	0246	0247	0248	0249	024A	024B	024C	024D	024E	024F	0250	0251	0252	0253	0254	0255	0256	0257	0258	0259	025A	025B	025C	025D	025E	025F	0260	0261	0262	0263	0264	0265	0266	0267	0268	0269	026A	026B	026C	026D	026E	026F	0270	0271	0272	0273	0274	0275	0276	0277	0278	0279	027A	027B	027C	027D	027E	027F	0280	0281	0282	0283	0284	0285	0286	0287	0288	0289	028A	028B	028C	028D	028E	028F	0290	0291	0292	0293	0294	0295	0296	0297	0298	0299	029A	029B	029C	029D	029E	029F	02A0	02A1	02A2	02A3	02A4	02A5	02A6	02A7	02A8	02A9	02AA	02AB	02AC	02AD	02AE	02AF	02B0	02B1	02B2	02B3	02B4	02B5	02B6	02B7	02B8	02B9	02BA	02BB	02BC	02BD	02BE	02BF	02C0	02C1	02C2	02C3	02C4	02C5	02C6	02C7	02C8	02C9	02CA	02CB	02CC	02CD	02CE	02CF	02D0	02D1	02D2	02D3	02D4	02D5	02D6	02D7	02D8	02D9	02DA	02DB	02DC	02DD	02DE	02DF	02E0	02E1	02E2	02E3	02E4	02E5	02E6	02E7	02E8	02E9	02EA	02EB	02EC	02ED	02EE	02EF	02F0	02F1	02F2	02F3	02F4	02F5	02F6	02F7	02F8	02F9	02FA	02FB	02FC	02FD	02FE	02FF	0300	0301	0302	0303	0304	0305	0306	0307	0308	0309	030A	030B	030C	030D	030E	030F	0310	0311	0312	0313	0314	0315	0316	0317	0318	0319	031A	031B	031C	031D	031E	031F	0320	0321	0322	0323	0324	0325	0326	0327	0328	0329	032A	032B	032C	032D	032E	032F	0330	0331	0332	0333	0334	0335	0336	0337	0338	0339	033A	033B	033C	033D	033E	033F	0340	0341	0342	0343	0344	0345	0346	0347	0348	0349	034A	034B	034C	034D	034E	034F	0350	0351	0352	0353	0354	0355	0356	0357	0358	0359	035A	035B	035C	035D	035E	035F	0360	0361	0362	0363	0364	0365	0366	0367	0368	0369	036A	036B	036C	036D	036E	036F	0370	0371	0372	0373	0374	0375	0376	0377	0378	0379	037A	037B	037C	037D	037E	037F	0380	0381	0382	0383	0384	0385	0386	0387	0388	0389	038A	038B	038C	038D	038E	038F	0390	0391	0392	0393	0394	0395	0396	0397	0398	0399	039A	039B	039C	039D	039E	039F	03A0	03A1	03A2	03A3	03A4	03A5	03A6	03A7	03A8	03A9	03AA	03AB	03AC	03AD	03AE	03AF	03B0	03B1	03B2	03B3	03B4	03B5	03B6	03B7	03B8	03B9	03BA	03BB	03BC	03BD	03BE	03BF	03C0	03C1	03C2	03C3	03C4	03C5	03C6	03C7	03C8	03C9	03CA	03CB	03CC	03CD	03CE	03CF	03D0	03D1	03D2	03D3	03D4	03D5	03D6	03D7	03D8	03D9	03DA	03DB	03DC	03DD	03DE	03DF	03E0	03E1	03E2	03E3	03E4	03E5	03E6	03E7	03E8	03E9	03EA	03EB	03EC	03ED	03EE	03EF	03F0	03F1	03F2	03F3	03F4	03F5	03F6	03F7	03F8	03F9	03FA	03FB	03FC	03FD	03FE	03FF	0400	0401	0402	0403	0404	0405	0406	0407	0408	0409	040A	040B	040C	040D	040E	040F	0410	0411	0412	0413	0414	0415	0416	0417	0418	0419	041A	041B	041C	041D	041E	041F	0420	0421	0422	0423	0424	0425	0426	0427	0428	0429	042A	042B	042C	042D	042E	042F	0430	0431	0432	0433	0434	0435	0436	0437	0438	0439	043A	043B	043C	043D	043E	043F	0440	0441	0442	0443	0444	0445	0446	0447	0448	0449	044A	044B	044C	044D	044E	044F	0450	0451	0452	0453	0454	0455	0456	0457	0458	0459	045A	045B	045C	045D	045E	045F	0460	0461	0462	0463	0464	0465	0466	0467	0468	0469	046A	046B	046C	046D	046E	046F	0470	0471	0472	0473	0474	0475	0476	0477	0478	0479	047A	047B	047C	047D	047E	047F	0480	0481	0482	0483	0484	0485	0486	0487	0488	0489	048A	048B	048C	048D	048E	048F	0490	0491	0492	0493	0494	0495	0496	0497	0498	0499	049A	049B	049C	049D	049E	049F	04A0	04A1	04A2	04A3	04A4	04A5	04A6	04A7	04A8	04A9	04AA	04AB	04AC	04AD	04AE	04AF	04B0	04B1	04B2	04B3	04B4	04B5	04B6	04B7	04B8	04B9	04BA	04BB	04BC	04BD	04BE	04BF	04C0	04C1	04C2	04C3	04C4	04C5	04C6	04C7	04C8	04C9	04CA	04CB	04CC	04CD	04CE	04CF	04D0	04D1	04D2	04D3	04D4	04D5	04D6	04D7	04D8	04D9	04DA	04DB	04DC	04DD	04DE	04DF	04E0	04E1	04E2	04E3	04E4	04E5	04E6	04E7	04E8	04E9	04EA	04EB	04EC	04ED	04EE	04EF	04F0	04F1	04F2	04F3	04F4	04F5	04F6	04F7	04F8	04F9	04FA	04FB	04FC	04FD	04FE	04FF	0500	0501	0502	0503	0504	0505	0506	0507	0508	0509	050A	050B	050C	050D	050E	050F	0510	0511	0512	0513	0514	0515	0516	0517	0518	0519	051A	051B	051C	051D	051E	051F	0520	0521	0522	0523	0524	0525	0526	0527	0528	0529	052A	052B	052C	052D	052E	052F	0530	0531	0532	0533	0534	0535	0536	0537	0538	0539	053A	053B	053C	053D	053E	053F	0540	0541	0542	0543	0544	0545	0546	0547	0548	0549	054A	054B	054C	054D	054E	054F	0550	0551	0552	0553	0554	0555	0556	0557	0558	0559	055A	055B	055C	055D	055E	055F	0560	0561	0562
------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------

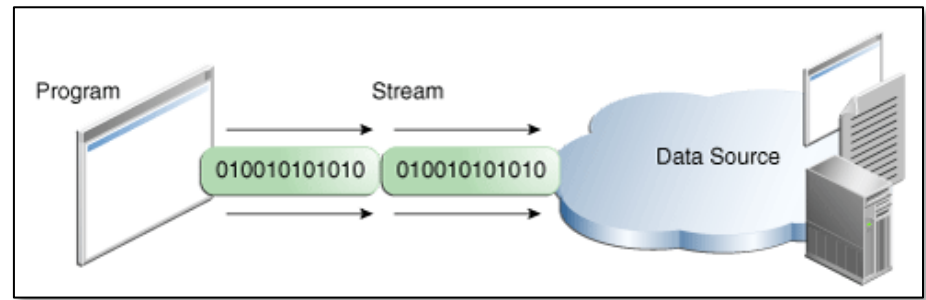
# File Input and Output

## (File and File Streams)

- Java views file as a sequential **stream** of bytes or characters
- File streams can be used to input and output data as bytes or characters



*Reading* information into a program,  
via an *input* stream



*Writing* information from a program,  
via an *output* stream

# File Input and Output

## (Text vs. Binary File I/O)

- Java Text File I/O requires encoding and decoding (via Character-based streams)
  - Write : Unicode → File
  - Read : File → Unicode
- Java Binary File I/O does not require conversions (via Byte-based streams)
  - Write : original byte is copied into the file
  - Read : exact byte in the file is returned
- Many basic File handling resources could be found in the `java.io` package (and in newer version, `java.nio.file` package)
- File manipulation in Java often involves *exception handling*, e.g.
  - `IOException`, `FileNotFoundException`



# File Input and Output

## (The `File` Class)

- To operate on a file, we first create a **File object** (from package `java.io`). For example:

```
File inFile = new File("sample.txt");
```

Opens the file `sample.txt` in the current directory

```
File inFile = new File  
("C:/OOP/test.txt");
```

Opens the file `test.txt` in the directory `C:\OOP` using the generic file separator `'/'` and providing the full pathname



# Some File Methods

Reference  
Only

```
if ( inFile.exists() ) {
```

To see if `inFile` is associated to a real file correctly

```
if ( inFile.isFile() ) {
```

To see if `inFile` is associated to a file (`true`) or a directory (`false`)

List the name of all files in the directory `C:/OOP`

# Example: TestMain.java

Reference  
Only

```
// import statements
import javax.swing.*;
import java.io.*;

public class TestMain {
    public static void main(String[] args) {

        String fileName = "sample.txt";
        File inFile = new File(fileName);
        // If inFile is associated to a real file correctly,
        // show a message dialog "sample.txt exists"
        if ( inFile.exists() ) {
            JOptionPane.showMessageDialog(null, fileName + " exists.");
        } else {
            JOptionPane.showMessageDialog(null, fileName + " doesn't exist!");
        }

        // List the name of all files in the directory "D:/"
        File directory = new File("D:/");
        String fileName2[] = directory.list();
        for (int i = 0; i < fileName2.length; i++) {
            System.out.println(fileName2[i]);
        }
    }
}
```

# Common File Manipulation

- Apart from basic reading and writing data, other file manipulation functions may support better file handling, including:
  - Choosing files
  - Copying, Renaming, and Deleting files
- It is common to use classes `java.io.File` (and in newer version, `java.nio.file.Files`) for basic file manipulation such as copying, renaming, and deleting files
- We may also use `javax.swing.JFileChooser` to choose/select a file via Graphical User Interface

# Simple Copy, Rename and Delete File

Reference  
Only

With `java.io.File` (and in newer version, `java.nio.file.Files`), and the string name of source and target files, we can:

- **Copy** (from original file name String `srcFName` to `targetFName`)

```
java.nio.file.Files.copy(new File(srcFName).toPath(),  
    new File(targetFName).toPath());
```

- **Rename** (from file name String `srcFName` to `targetFName`)

```
new File(srcFName).renameTo(new File(targetFName))
```

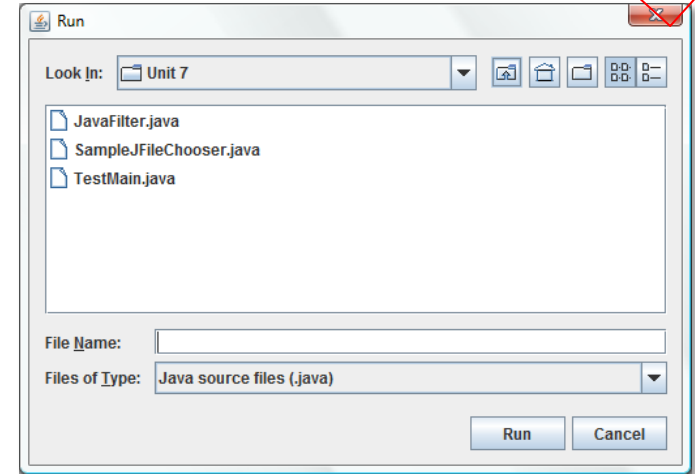
- **Delete** (the file name String `srcFName`)

```
new File(srcFName).delete();
```

# The JFileChooser Class

Reference  
Only

- JFileChooser allows the user to choose/select a file via Graphical User Interface
  - In package `javax.swing`



Default directory:

```
JFileChooser chooser = new JFileChooser( );  
chooser.showOpenDialog(null);
```

To start the listing from a specific directory with customized label:

```
JFileChooser chooser = new JFileChooser("D:/Unit7");  
chooser.showDialog(null, "Run");
```

# Getting Info from JFileChooser

Reference  
Only

```
int status = chooser.showOpenDialog(null);
if (status == JFileChooser.APPROVE_OPTION) {
    JOptionPane.showMessageDialog(null, "Open is clicked");

} else { //== JFileChooser.CANCEL_OPTION
    JOptionPane.showMessageDialog(null, "Cancel is clicked");
}
```

```
File selectedFile = chooser.getSelectedFile();
```

```
String fileName = selectedFile.getName();
```

```
File currentDirectory = chooser.getCurrentDirectory();
```

# Example: SampleJFileChooser.java

Reference  
Only

```
import javax.swing.*;
import java.io.*;

public class SampleJFileChooser {
    public static void main(String[] args) {
        JFileChooser chooser;
        File file;
        int status;

        // start the listing from a specific directory "D:/"
        chooser = new JFileChooser("./");

        // show the dialog with customized label "Browse"
        status = chooser.showDialog(null, "Browse");

        if (status == JFileChooser.APPROVE_OPTION) {
            file = chooser.getSelectedFile();

            // show the message dialog "Open File: <Your selected file name>"
            JOptionPane.showMessageDialog(null, "Open File: " + file.getName());
        } else {
            JOptionPane.showMessageDialog(null, "Open File dialog cancelled");
        }
    }
}
```



# Text File Input and Output

- **Handling data as string text**
  - This allows us to view the file content using any text editor
- To output/write data as a string to file, we may use `PrintWriter` or `FileWriter` classes (in package `java.io`)
- To input/read data from a text file, we may use `FileReader` and `BufferedReader` classes (in package `java.io`)
- From Java 5.0 (SDK 1.5), we can also use the `Scanner` class for inputting text files

- `java.io.Writer` (implements `java.lang`)
  - `java.io.BufferedWriter`
  - `java.io.CharArrayWriter`
  - `java.io.FilterWriter`
  - `java.io.OutputStreamWriter`
    - `java.io.FileWriter`
  - `java.io.PipedWriter`
  - `java.io.PrintWriter`
  - `java.io.StringWriter`

- `java.io.Reader` (implements `java.io.Closeable`)
  - `java.io.BufferedReader`
    - `java.io.LineNumberReader`
  - `java.io.CharArrayReader`
  - `java.io.FilterReader`
    - `java.io.PushbackReader`
  - `java.io.InputStreamReader`
    - `java.io.FileReader`
  - `java.io.PipedReader`
  - `java.io.StringReader`

Reference  
Only

# Text File Input and Output

(Simple Sample Codes, for Writing / Appending Text File)

- To **write/append text data as strings into a text file**, we may use `PrintWriter` or `FileWriter` classes, with methods below:

1) To **create and open** an output stream **for writing** text, e.g.:

```
String fileName = "myFile.txt"; // a file name (string)
PrintWriter outputStream = new PrintWriter(fileName);
```

– To create and open an output stream **for appending** text, e.g.:

```
String fileName = "myFile.txt";
PrintWriter outputStream = new PrintWriter(
    new FileWriter(fileName, true)); // true for appending
```

2) To **write a string line** to text output stream (with newline), e.g.:

```
String aStr = "This is a string line.";
outputStream.println(aStr); // end with a newline '\n' added
```

3) To **close the stream**, e.g.:

```
outputStream.close(); //close the stream
```

# Text File Input and Output

## (Simple Sample Codes , for Reading Text File)

- To **input/read string data from a text file**, we may use `FileReader` and `BufferedReader` classes, with methods below:

- 1) To **create and open** an input stream for **reading text**, e.g.:

```
BufferedReader bufferReader =  
    new BufferedReader(new FileReader(fileName));
```

- 2) To **read a string line** from text input stream, e.g.:

```
// below read a line, NOT including the newline '\n'  
String strLine = bufferReader.readLine();
```

- 3) To **close the stream**, e.g.:

```
bufferReader.close(); //close the stream
```

# Example: Text File Input / Output

## (main() Method for Testing)

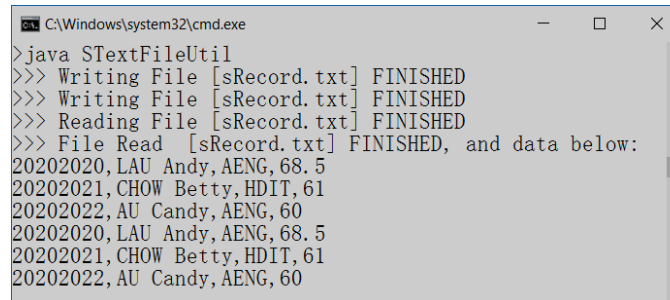
Reference  
Only

STextFileUtil.java

```
// STextFileUtil.java
import java.io.*;
import java.util.*;
public class STextFileUtil {
    public static void main(String[] args){ // main, for testing
        String fName = "sRecord.txt";
        String [] fData = {"20202020,LAU Andy,AENG,68.5",
                           "20202021,CHOW Betty,HDIT,61",
                           "20202022,AU Candy,AENG,60"};

        STextFileUtil.writeTextFile(fName, fData);
        STextFileUtil.appendTextFile(fName, fData);
        String [] fDataStrs = STextFileUtil.readTextFile(fName);
        System.out.println(">>> File Read [" + fName
                           + "] FINISHED, and data below:" );
        for (int i=0; i<fDataStrs.length; i++)
            System.out.println(fDataStrs[i]);
    }

    // continue ...
    // More coming
}
```



```
C:\Windows\system32\cmd.exe
>java STextFileUtil
>>> Writing File [sRecord.txt] FINISHED
>>> Writing File [sRecord.txt] FINISHED
>>> Reading File [sRecord.txt] FINISHED
>>> File Read [sRecord.txt] FINISHED, and data below:
20202020,LAU Andy,AENG,68.5
20202021,CHOW Betty,HDIT,61
20202022,AU Candy,AENG,60
20202020,LAU Andy,AENG,68.5
20202021,CHOW Betty,HDIT,61
20202022,AU Candy,AENG,60
```

Text file, e.g.  
sRecord.txt

```
20202020,LAU Andy,AENG,68.5
20202021,CHOW Betty,HDIT,61
20202022,AU Candy,AENG,60
20202020,LAU Andy,AENG,68.5
20202021,CHOW Betty,HDIT,61
20202022,AU Candy,AENG,60
```

# Example: Text File Output

(Write – may overwrite all current data, if file exists)

Reference  
Only

STextFileUtil.java

```
// continue ...
```

```
public static boolean writeTextFile(String fileName,
                                    String [] fileContent){
    try{
        PrintWriter outputStream = new PrintWriter(fileName);
        for (int i=0; i<fileContent.length; i++)
            outputStream.println(fileContent[i]); //write into file, in a line
        outputStream.close(); //close the stream
        System.out.println(">>> Writing File [" + fileName + "] FINISHED" );
    } catch (FileNotFoundException fnfE){
        System.out.println(">>> Exception, FileNotFoundException");
        return false;
    }
    return true;
}
// continue ...
```

Text file, e.g.  
sRecord.txt

```
20202020, LAU Andy, AENG, 68.5
20202021, CHOW Betty, HDIT, 61
20202022, AU Candy, AENG, 60
```

# Example: Text File Output

(Append – Append data at file end, if file exists)

Reference  
Only

STextFileUtil.java

```
// continue ...
```

```
public static boolean appendTextFile // class method to append
    (String fileName, String [] fileContent) {
    try{
        //Creates a new PrintWriter, with specified file name
        PrintWriter outputStream = new PrintWriter(
            new FileWriter(fileName, true)); // true for appending
        for (int i=0; i<fileContent.length; i++)
            outputStream.println(fileContent[i]); // write into file, in a line
        outputStream.close(); //close the stream
        System.out.println(">>> Writing File [" + fileName + "] FINISHED" );
    } catch (IOException ioE){
        System.out.println(">>> Exception, FileNotFoundException");
        return false;
    }
    return true;
}
```

```
// continue ...
```

Text file, e.g.  
sRecord.txt

```
20202020, LAU Andy, AENG, 68.5
20202021, CHOW Betty, HDIT, 61
20202022, AU Candy, AENG, 60
20202020, LAU Andy, AENG, 68.5
20202021, CHOW Betty, HDIT, 61
20202022, AU Candy, AENG, 60
```

# Text File Input (Read) (Use FileReader)

Reference  
Only

STextFileUtil.java

```
// continue ...
```

```
public static String[] readTextFile(String fileName){
    ArrayList<String> retAList = new ArrayList<String>(); // arraylist
    try{
        String strLine;
        BufferedReader bufferReader =
            new BufferedReader(new FileReader(fileName));
        // read file to string until end, one line a time
        while ((strLine = bufferReader.readLine()) != null) {
            retAList.add(strLine); // add the read line data to arraylist
        }
        bufferReader.close(); //close the stream
        System.out.println(">>> Reading File [" + fileName + "] FINISHED" );
    } catch (FileNotFoundException fnfE){
        System.out.println(">>> Exception, FileNotFoundException");
        return null;
    } catch (IOException ioE){
        System.out.println(">>> Exception, IOException");
        return null;
    }
    // ArrayList to String array with method toArray()
    return retAList.toArray(new String[retAList.size()]);
}
```



# Low-Level File I/O (in Bytes)

- Java offers many Byte-based stream classes to handle binary file I/O:
  - **Byte Streams** (`FileInputStream` / `FileOutputStream`) handle I/O of raw binary data.
  - **Data Streams** (`DataInputStream` / `DataOutputStream`) handle binary I/O of primitive data type and string values
  - **Object Streams** (`ObjectInputStream` / `ObjectOutputStream`) handle binary I/O of objects

- `java.io.InputStream` (implements `java.io.Closeable`)
  - `java.io.ByteArrayInputStream`
  - `java.io.FileInputStream`
  - `java.io.FilterInputStream`
    - `java.io.BufferedInputStream`
    - `java.io.DataInputStream` (implements `java.io.DataInput`)
    - `java.io.LineNumberInputStream`
    - `java.io.PushbackInputStream`
  - `java.io.ObjectInputStream` (implements `java.io.ObjectInput`, `java.io.ObjectStreamConstants`)

Reference  
Only

- `java.io.OutputStream` (implements `java.io.Closeable`, `java.io.Flushable`)
  - `java.io.ByteArrayOutputStream`
  - `java.io.FileOutputStream`
  - `java.io.FilterOutputStream`
    - `java.io.BufferedOutputStream`
    - `java.io.DataOutputStream` (implements `java.io.DataOutput`)
    - `java.io.PrintStream` (implements `java.lang.Appendable`, `java.io.Closeable`)
  - `java.io.ObjectOutputStream` (implements `java.io.ObjectOutput`, `java.io.ObjectStreamConstants`)

Reference  
Only

# Low-Level File I/O (in Bytes)

- To read data from or write data to a file, we must create one of the Java stream objects and attach it to the file
- A ***stream*** (byte stream) is a sequence of data items, usually 8-bit bytes
- Java has two types of streams: an *input stream* and an *output stream*
- An *input stream* has a source from which the data items come, and an *output stream* has a destination to which the data items are going

# Streams for Low-Level File I/O (in Bytes)

- `FileOutputStream` and `FileInputStream` are two stream objects that facilitate file access
- `FileOutputStream` allows us to output ***a sequence of bytes*** (values of data type `byte`)
- `FileInputStream` allows us to read in an array of bytes.
  - \* A `java.io.FileNotFoundException` would occur if you attempt to create a `FileInputStream` with a nonexistent file

API references:

<https://docs.oracle.com/en/java/javase/25/docs/api/java.base/java/io/FileInputStream.html>

<https://docs.oracle.com/en/java/javase/25/docs/api/java.base/java/io/FileOutputStream.html>

# Sample: Low-Level File Output (in Bytes)

Reference  
Only

TestFileOutputStream.java

Binary File  
sample1.dat:

(2<FP

```
//set up file and stream
FileOutputStream outStream = null;
try {
    outStream = new FileOutputStream ("sample1.dat");
    //data to save
    byte[] byteArray = {10, 20, 30, 40, 50, 60, 70, 80};
    //write data to the stream
    outStream.write( byteArray );
    //output done, so close the stream
    outStream.close();
} catch (IOException ioe){
    System.out.println("ERR: IOException");
}
```

# Sample: Low-Level File Input (in Bytes)

Reference  
Only

TestFileInputStream.java

```
//set up file and stream
File inFile = new File( "sample1.dat" );
FileInputStream inStream = null;
try {
    inStream = new FileInputStream(inFile);
    // we may also create directly, with specified file name
    // outStream = new FileInputStream("sample1.txt");
    // set up an array to read data in
    int    fileSize = (int) inFile.length();
    byte[] byteArray = new byte[fileSize];
    //read data in and display them
    inStream.read(byteArray);
    for (int i = 0; i < fileSize; i++){
        System.out.println(byteArray[i]);
    }
    //input done, so close the stream
    inStream.close();
} catch (IOException ioe){
    System.out.println("ERR: IOException");
}
```

Console  
output:

10  
20  
30  
40  
50  
60  
70  
80

# High-Level File I/O (in Primitive Types)

- `DataOutputStream` are used to output ***primitive data values***
  - take care of the details of converting the primitive data type values to a sequence of bytes
- `DataInputStream` is the reverse operation that read the data back from the files

- Set up a `DataOutputStream`, to write data to a file "aFile.dat"

```
dataOutS = new DataOutputStream(  
    new FileOutputStream("aFile.dat"));
```

- Set up a `DataInputStream`, to read data from a file "cFile.dat"

```
dataInS = new DataInputStream(  
    new FileInputStream("aFile.dat"));
```

# Reading Data Back in Right Order

- The order of write and read operations must match in order to read the stored primitive data back correctly, e.g.

```
# writing data from an array of Circle, to DataOutputStream File
for (int i = 0; i < circleArr.length; i ++) {
    dataOutS.writeInt(circleArr[i].x);
    dataOutS.writeInt(circleArr[i].y);
    dataOutS.writeDouble(circleArr[i].radius);
}
```

Circle
x: int y: int radius: double

Writing different  
primitive type data to a  
DataOutputStream  
object - dataOutS:



Reading different  
primitive type data from  
a DataInputStream  
object - dataInS:

```
# reading data from DataInputStream file, to an array of Circle
for (int i = 0; i < cInArr.length; i ++) {
    cInArr[i].x = dataInS.readInt();
    cInArr[i].y = dataInS.readInt();
    cInArr[i].radius = dataInS.readDouble();
}
```



# Example: High-Level File Output (in Primitive Types)

Reference  
Only

TestDataOutputStream.java

Binary File  
sample2.dat:

:Ph±K@ŠÇAInj€

```
import java.io.*;

public class TestDataOutputStream {
    public static void main (String[] args) throws IOException {
        //set up DataOutputStream object to write data to sample2.txt
        DataOutputStream outDataStream = new DataOutputStream(
            new FileOutputStream("sample2.dat") );

        //write values of primitive data types to the stream
        outDataStream.writeInt(987654321);
        outDataStream.writeFloat(22222222F);
        outDataStream.writeDouble(33333333D);

        //output done, so close the stream
        outDataStream.close();
    }
}
```

# Example: High-Level File Input (in Primitive Types)

Reference  
Only

TestDataInputStream.java

Console  
output:

987654321  
2.2222222E7  
3333333.0

```
import java.io.*;

public class TestDataInputStream {
    public static void main (String[] args) throws IOException {
        // set up the DataInputStream object
        DataInputStream inDataStream = new DataInputStream(
            new FileInputStream("sample2.dat") );

        //read values back from the stream and display them
        System.out.println(inDataStream.readInt());
        System.out.println(inDataStream.readFloat());
        System.out.println(inDataStream.readDouble());

        //input done, so close the stream
        inDataStream.close();
    }
}
```

# Object-Level File I/O

- It is possible to store objects just as easily as you store primitive data values
- We use `ObjectOutputStream` / `ObjectInputStream` to save to and load **objects** from a file
  - Object File I/O also includes exception handling and closing the stream
- To save objects from a given class, the class declaration must include the phrase `implements Serializable` (in the `java.io` package).

E.g.

```
public class Person implements java.io.Serializable {  
    //..  
}
```

\* *There are certain restrictions of object stream I/O in Java. Therefore, sometimes it is a bit complicated and tricky to use this Object Stream with `Serializable`. Details should be reference to: <http://docs.oracle.com/javase/25/docs/api/java/io/Serializable.html>*

# Saving and Reading Object Arrays

- Instead of processing array elements individually, it is even possible to save and read the whole array of objects at once

```
Person[] people = new Person[ N ];  
                //assume N already has a value  
  
//build the people array  
//...  
//save the array  
outObjectStream.writeObject ( people );
```

```
//read the array  
  
Person[] people = (Person[]) inObjectStream.readObject( );
```

# Saving Objects

Reference  
Only

```
FileOutputStream    outFileStream  
                    = new FileOutputStream("objects.obj");  
ObjectOutputStream outObjectStream  
                    = new ObjectOutputStream(outFileStream);
```

```
Person person = new Person("Mr. Espresso", 20, 'M');  
outObjectStream.writeObject( person );
```

```
account1    = new Account();  
bank1       = new Bank();  
  
outObjectStream.writeObject( account1 );  
outObjectStream.writeObject( bank1    );
```

Could save objects  
from the different  
classes

# Example: Object Output

Reference  
Only

TestObjectOutputStream.java

```
import java.io.*;

public class TestObjectOutputStream {
    public static void main (String[] args) throws IOException {
        //set up file and stream
        FileOutputStream outFileStream
            = new FileOutputStream("objects.obj");
        ObjectOutputStream outObjectStream
            = new ObjectOutputStream(outFileStream);
        //write serializable Person objects
        Person person;
        for (int i = 0; i < 3; i++) {
            person = new Person("Mr. Espresso" + i, 20+i, 'M');
            outObjectStream.writeObject(person);
        }
        //input done, so close the stream
        outObjectStream.close();
    }
}
```

Object File

Objects.obj:

```
秒 sr Person 漢
-? I ageC genderL namet Ljava/lang/String;xp Mt
Mr. Espresso0sq ~ Mt
Mr. Espresso1sq ~ Mt
Mr. Espresso2
```

# Reading Objects

Reference  
Only

TestObjectInputStream.java

```
FileInputStream    inFileStream
                  = new FileInputStream("objects.obj");
ObjectInputStream inObjectStream
                  = new ObjectInputStream(inFileStream);
```

```
Person person
    = (Person) inObjectStream.readObject();
```

Must type cast  
to the correct  
object type.

```
Account account1
    = (Account) inObjectStream.readObject();
Bank    bank1
    = (Bank) inObjectStream.readObject();
```

Must read in  
the correct  
order.

Console  
output:

Mr. Espresso0	20	M
Mr. Espresso1	21	M
Mr. Espresso2	22	M



# References

- This set of slides is only for educational purpose.
- Part of this slide set is referenced, extracted, and/or modified from the followings:
  - Deitel, P. and Deitel H. (2017) “Java How To Program, Early Objects”, 11ed, Pearson.
  - Liang, Y.D. (2017) “Introduction to Java Programming and Data Structures”, Comprehensive Version, 11ed, Prentice Hall.
  - Wu, C.T. (2010) “An Introduction to Object-Oriented Programming with Java”, 5ed, McGraw Hill.
  - Oracle Corporation, “Java Language and Virtual Machine Specifications” <https://docs.oracle.com/javase/specs/>
  - Oracle Corporation, “The Java Tutorials” <https://docs.oracle.com/javase/tutorial/>
  - Wikipedia, Website: <https://en.wikipedia.org/>