Chapter 4 Java I/O

Xiang Zhang

javacose@qq.com

Content

- Java I/O Introduction

- File and Directory

- Byte-stream and Character-stream

- Bridge between b-s and c-s

- Random Access File

- Standard I/O

  ○ System.in

  ○ System.out

- java.nio Pilot

Java I/O Introduction

- I/O Target
  - File
  - Console
  - Network Connection

- I/O Manner
  - Text-based (char) / Data-based(byte)
  - Sequential / Random Access

- java.io Package

  - general classes

  - filtered / buffered / piped streams

  - data streams

  - File

  - object serialization

File and Directory

- java.io.File -   "A Path in a file system"
  - File
  - Directory

- File Construction

```
File file = new File("c:/Windows/explorer.exe");
File file = new File("c:/Windows","explorer.exe");
File file = new File(".");
...
System.out.println(file.exists());
```
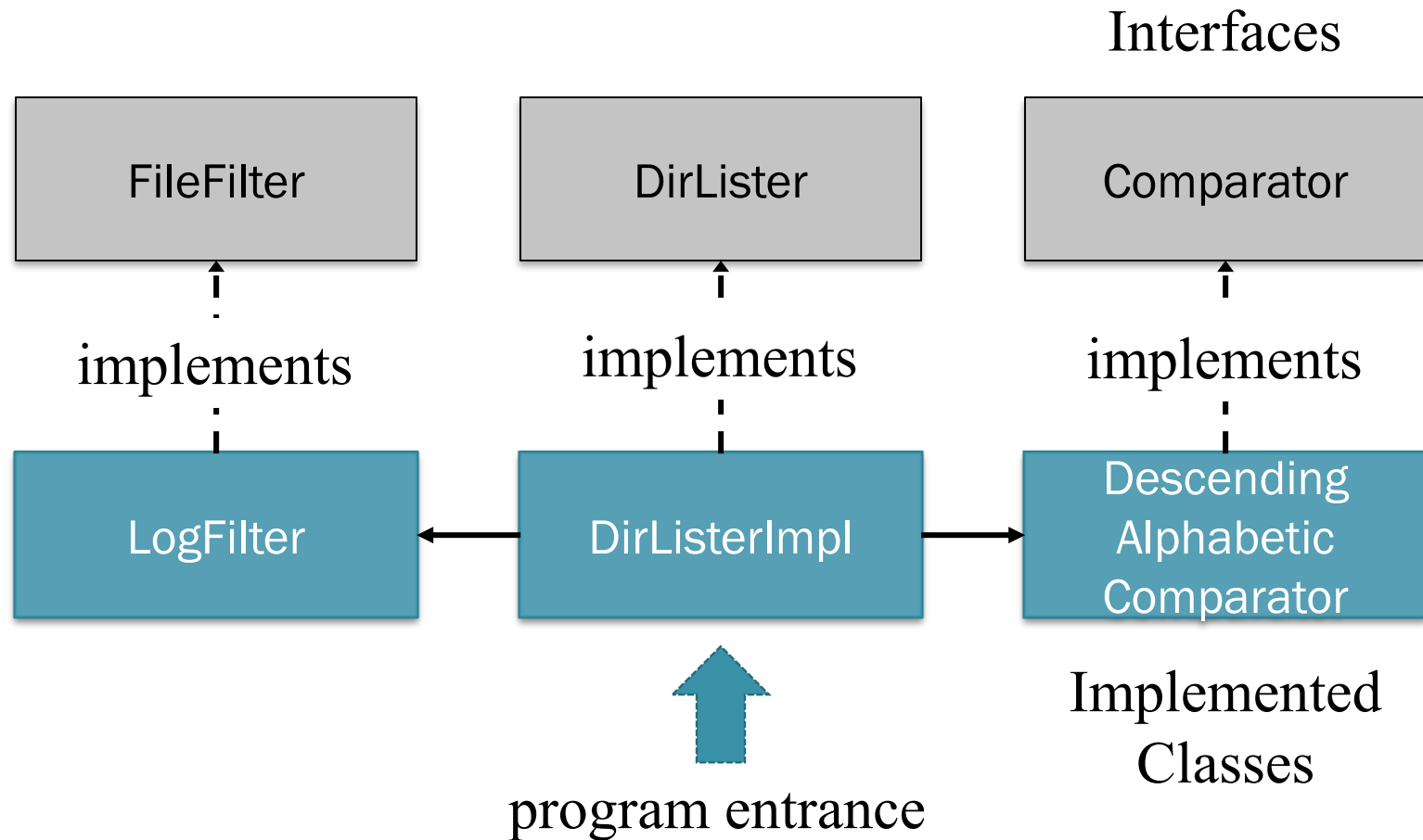
- Example: Directory Lister

  List *.log from "c:\Windows" in descending order

Interface: FileFilter ➡ java.io.FileFilter

Interface: Comparator ➡ java.util.Comparator

语法糖

- A Java grammar sugar from Java8;
  - Usually simplified inner anonymous class;
- For a concise and readable program;
- Allows a function working as a parameter;

Java Lambda Syntax
Concise
`n -> System.out.print(n)`
Expanded
`(String n) -> System.out.print(n)`
Verbose
`(String n) -> { System.out.print(n); }`

```java
public class ReverseSortAlgorithms {

    void reverseSort(List<String> list) {
        Collections.sort(list, (o1, o2)-> -o1.compareTo(o2));
    }
}
```

1. Lambda expression

```java
public class ReverseSortAlgorithms {

    void reverseSort(List<String> list) {
        Collections.sort(list, Collections.reverseOrder());
    }
}
```

2. Java Implemented

```java
public class ReverseSortAlgorithms {

    void reverseSort(List<String> list) {
        Collections.sort(list, new ReverseComparator());
    }

    public class ReverseComparator implements Comparator{

        @Override
        public int compare(Object o1, Object o2) {
            // TODO Auto-generated method stub
            return -o1.toString().compareTo(o2.toString());
        }
    }
}
```

3. Inner Class

```java
public class FileLister {
    // 由毛欣威、陈奕帆、杜晓欣、张祥共同完成
    ArrayList<File> files;
    File path;
    String suffix;
    // 构造方法
    public FileLister(File path, String suffix){
        this.files = new ArrayList<File>();
        this.path = path;
        this.suffix = suffix;
    }
    // 递归过滤出给定文件夹中所有满足条件的文件名
    public void filter() {
        if(path.isDirectory()) {
            File[] filesInThisPath = path.listFiles();
            for(File f: filesInThisPath) {
                if(f.isDirectory()) {path=f; filter();}
                else if(f.toString().endsWith(suffix)) {files.add(f);}
            }
        }
    }
    // 逆序排序
    public void reversedSort() {
        Collections.sort(files, (f1,f2)->-f1.compareTo(f2));
        System.out.println(files.toString());
    }

    public static void main(String[] args) {
        FileLister lister = new FileLister(new File("e:/temp/"), "log");
        lister.filter(); lister.reversedSort();
    }
}
```

A concise version
by three students

采集 Dec	Char	Dec	Char	Dec	Char	Dec	Char	Dec	Char	Dec	Char
33	!	49	1	65	A	81	Q	97	a	113	q
34	"	50	2	66	B	82	R	98	b	114	r
35	#	51	3	67	C	83	S	99	c	115	s
36	$	52	4	68	D	84	T	100	d	116	t
37	%	53	5	69	E	85	U	101	e	117	u
38	&	54	6	70	F	86	V	102	f	118	v
39	'	55	7	71	G	87	W	103	g	119	w
40	(	56	8	72	H	88	X	104	h	120	x
41	)	57	9	73	I	89	Y	105	i	121	y
42	*	58	:	74	J	90	Z	106	j	122	z
43	+	59	;	75	K	91	[	107	k	123	{
44	,	60	<	76	L	92	\	108	l	124	|
45	-	61	=	77	M	93	]	109	m	125	}
46	.	62	>	78	N	94	^	110	n	126	~
47	/	63	?	79	O	95	_	111	o	127	_
48	0	64	@	80	P	96	`	112	p		

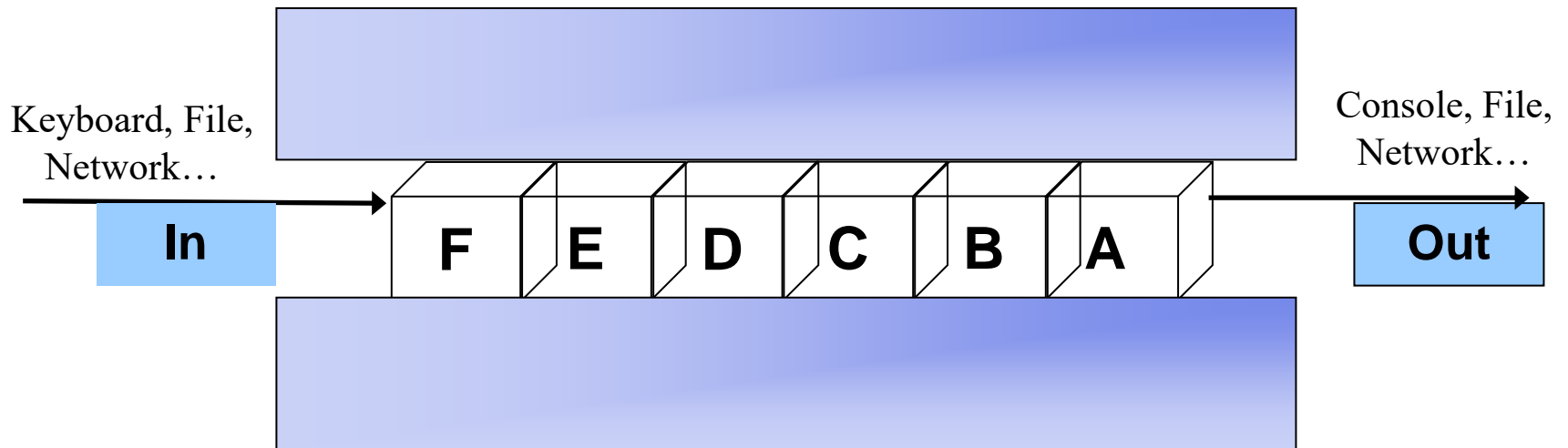In the results, why WindowsUpdate.log is behind setuperr.log ?

Can you write a case-insensitive DirLister?

- # The Notion of Stream

  - ○ A sequence of flowing byte / char

  - ○ A channel sending message in FIFO

Keyboard, File, Network…

In

F E D C B A

Console, File, Network…

Out

Stream

- Classification of Stream

  - Byte Stream

    - Byte as the unit   10010011 01010010 10100101 01010100

    - Used to read and write binary data

  - Character Stream

    - Char as the unit   Welcome to the CoSE!

    - Used to read and write text

- ## Abstract Stream Class in java.io
  - Byte stream
    - java.io.InputStream
      - int read()   //read a byte, something wrong?
    - java.io.OutputStream
      - void write(int b)   //write an int ?? Why not byte?
      - void write(byte[] b)
  - Character stream
    - java.io.Reader
      - int read()   //read a char, something wrong?
    - java.io.Writer
      - void write(int b)   //write an int ?? Why not char?
   - void write(char[] c)

- # Implemented Classes in java.io

  - ## Byte stream

    - FileInputStream、FileOutputStream

    - *PipedInputStream、PipedOutputStream

    - ByteArrayInputStream、 ByteArrayOutputputStream

    - BufferedInputStream、BufferedOutputStream

    - ObjectInputStream、ObjectOutputStream

  - ## Character stream

    - FileReader、FileWriter

    - *PipedReader、PipedWriter

    - BufferedReader、BufferedWriter

    - InputStreamReader、OutputStreamWriter

Byte Stream

- A Complex Hierarchy of Byte Stream
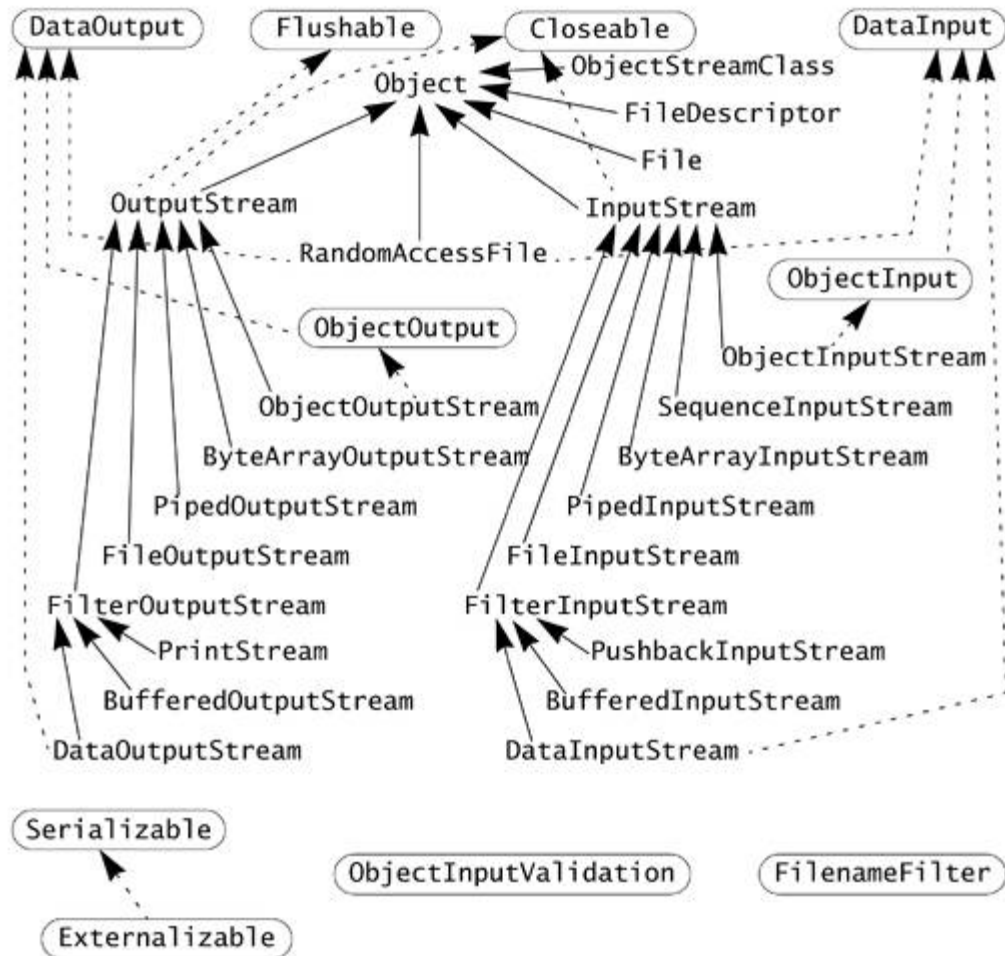
Byte Stream

- ## FileInputStream
  - Read bytes from file system
  - Used to read image or data

- ## FileOutputStream
  - Write bytes to file system
  - Used to write image or data

…10110100 10111001…

- Example:
  - Write following data into "c:\test.dat"
  - Read them out
    - byte 97
    - char 'b'
    - String "好"

```java
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;

public class FileStreamTester {

    private FileInputStream fis;
    private FileOutputStream fos;

    public FileStreamTester(File file) throws IOException{
        createFile(file);
        this.fis = new FileInputStream(file);
        this.fos = new FileOutputStream(file);
    }

    public static void createFile(File file) throws IOException{
        if(!file.exists()||!file.isFile()){
            file.createNewFile();
        }
    }
}
```

```java
public void close() throws IOException{
    fis.close();
    fos.close();
}

public int read() throws IOException{
    return fis.read();
}

public void write(int arg) throws IOException{
    fos.write(arg);
}

public void write(byte[] arg) throws IOException{
    fos.write(arg);
}

public int available() throws IOException{
    return fis.available();
}
```

```java
public static void main(String[] args){
    try {
        File file = new File("d:/test.dat");
        FileStreamTester tester = new FileStreamTester(file);
        tester.write(97);   // What will happen if we write 260?
        tester.write('b');
        tester.write(new String("好").getBytes());

        System.out.println(tester.available() + " size");
        int i = tester.read();
        while (i != -1) {
            System.out.println((char)i);
            i = tester.read();
        }    // if the file already exists and has data, what will happen?
        tester.close();   // Is there any better place for this close()?
    }catch(IOException e) {
        e.printStackTrace();
    }
}
```
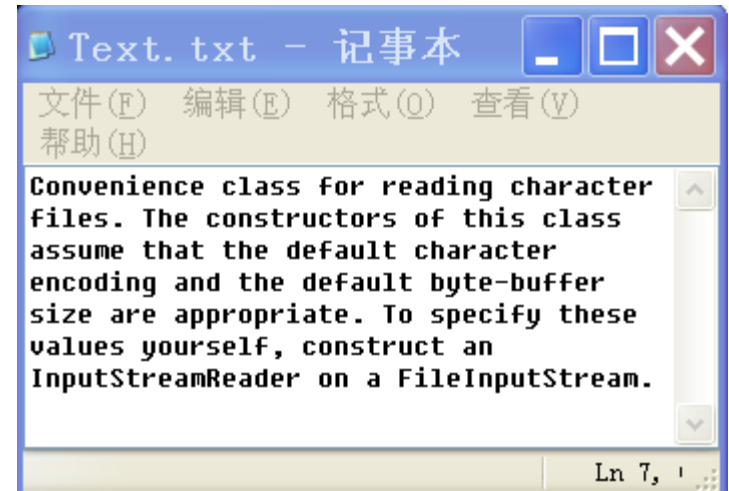
Character Stream

- FileReader
  - Read char from file
- FileWriter
  - Write char to file
- FileReader and FileWriter use system default encoding
- Use other encodings:
  - InputStreamReader
  - OutputStreamWriter

Text.txt – 记事本

文件(F)　编辑(E)　格式(O)　查看(V)　帮助(H)

Convenience class for reading character files. The constructors of this class assume that the default character encoding and the default byte-buffer size are appropriate. To specify these values yourself, construct an InputStreamReader on a FileInputStream.

Ln 7, '

'C' 'o' 'n' 'v' 'e' 'n' 'i' 'e' 'n' 'c' 'e'

- Example:
  - Write following chars
  - Read them out
    - 'C'   'o'   'S'   'E'
    - '软'   '件'   '学'   '院'

```java
import java.io.*;
public class TestWriter {
    public static void main(String[] args) throws IOException{
        File file = new File("c:/text.txt");
        FileWriter writer = new FileWriter(file, true);
        writer.write("CoSE".toCharArray());
        writer.write("软件学院".toCharArray());
        writer.flush();
        FileReader reader = new FileReader(file);
        int character = reader.read();
        while(character!=-1){
            System.out.println((char)character);
            character = reader.read();
        }
    }
}
```

- Byte Stream

  - An int or byte[] can be written to an OutputStream;

  - An int or byte[] can be read from an InputStream;

- Character Stream

  - An int or char[] or String can be written to an Writer;

  - An int or char[] or CharBuffer can be read from a Reader

- Think

  - How to input a student information into a file?

    - Student ID (int)

    - Name (String)

    - Age (short)

    - Sex (boolean)

  - How to read these information from file? (You can use get/put method in ByteBuffer, or …)

  - How to store these information in binary or text?

Self-study

- PrintStream(will be used in Chapter 11)

  o Inherited from OutputStream

- DataInputStream and DataOutputStream

  o Inherited from InputStream and OutputStream

- PrintWriter

  o Inherited from Writer

- Scanner

  o java.util.Scanner

Buffered Stream

○ Add buffer for input and output.

○ To improve the efficiency, read() and write() is not invoked immediately, but after the buffer is full.

○ Buffer is implemented using inner array.

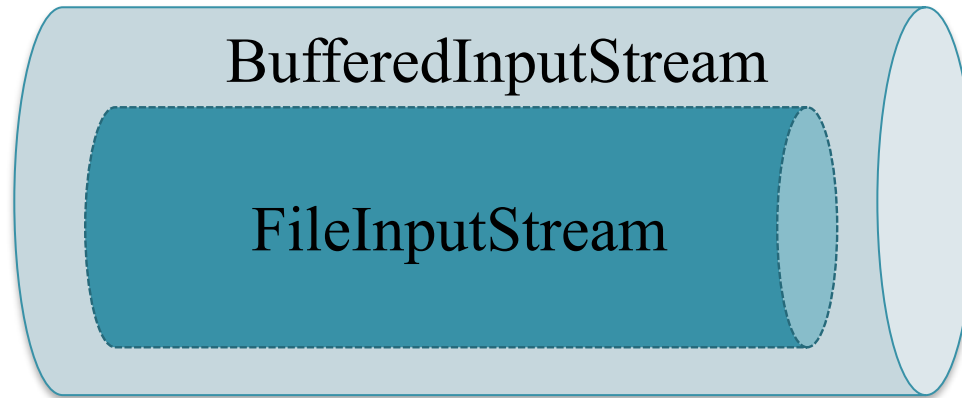○ Usually, buffered stream is connected to other streams (such as FileInputStream)

○ flush()

```
BufferedInputStream bis = new BufferedInputStream
                (new FileInputStream(new File("c:/test.dat")))
```
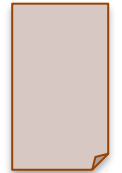
Buffer

Java
Program

block
of
bytes

BufferedInputStream

FileInputStream

bytes

File

- Example: Benchmark following classes
  - FileInputStream vs. BufferedInputStream;
  - FileOutputStream vs. BufferedOutputStream;

- Create a storage with MAX_STORE_SIZE (for example 500k random bytes)

- Write these bytes into a file one by one;

- Read these bytes from the file one by one;

- Benchmark the time efficiency;

```java
import java.io.*;
import java.util.ArrayList;

public class BufferTest {
    private ArrayList<Integer> intStoreSource;
    private ArrayList<Integer> intStoreTarget;
    private final int MAX_STORE_SIZE = 500000;
    private File file;
```

```java
public static int getRandomInt(){
    return (int)(Math.random()*255);
}

private void buildIntStore(){
    for(int i=0; i<MAX_STORE_SIZE; i++){
        intStoreSource.add(getRandomInt());
    }
}

public BufferTest(File file){
    this.intStoreSource = new ArrayList<Integer>();
    this.intStoreTarget = new ArrayList<Integer>();
    this.buildIntStore();
    this.file = file;
}
```

```java
private void useFileInputStream() throws IOException{
    FileInputStream fis = new FileInputStream(this.file);
    int intValue = fis.read();
    while(intValue!=-1){
        this.intStoreTarget.add(intValue);
        intValue = fis.read();
    }
    fis.close();
}

private void useFileOutputStream() throws IOException{
    FileOutputStream fos = new FileOutputStream(this.file);
    for(int i=0; i<this.intStoreSource.size();i++){
        fos.write(intStoreSource.get(i));
    }
    fos.close();
}
```

```java
private void useBufferedInputStream() throws IOException{
    BufferedInputStream bis = new BufferedInputStream
                                (new FileInputStream(this.file));
    int intValue = bis.read();
    while(intValue!=-1){
        this.intStoreTarget.add(intValue);
        intValue = bis.read();
    }
    bis.close();
}

private void useBufferedOutputStream() throws IOException{
    BufferedOutputStream bos = new BufferedOutputStream
                                (new FileOutputStream(this.file));
    for(int i=0; i<this.intStoreSource.size();i++){
        bos.write(intStoreSource.get(i));
    }
    bos.close();
}
```
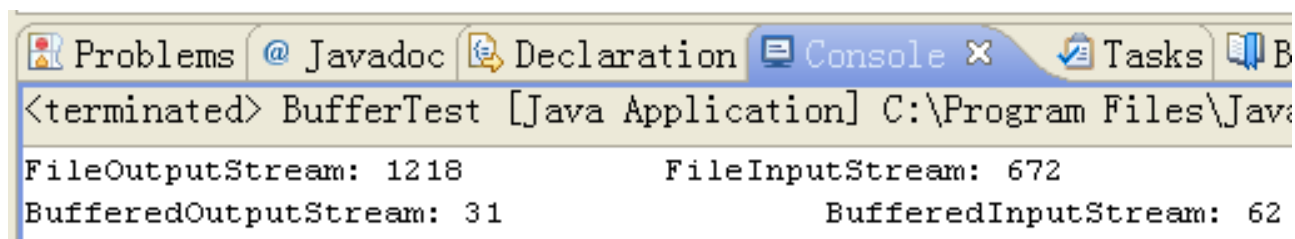
```java
private void test() throws IOException{
    System.gc();
    long time1 = System.currentTimeMillis();
    this.useFileOutputStream();
    long time2 = System.currentTimeMillis();
    this.useFileInputStream();
    long time3 = System.currentTimeMillis();

    this.intStoreTarget = new ArrayList<Integer>();
    System.gc();
    long time4 = System.currentTimeMillis();
    this.useBufferedOutputStream();
    long time5 = System.currentTimeMillis();
    this.useBufferedInputStream();
    long time6 = System.currentTimeMillis();

    System.out.println("FileOutputStream: " + (time2-time1) + "\t\t" +
            "FileInputStream: " + (time3-time2));
    System.out.println("BufferedOutputStream: " + (time5-time4) + "\t\t" +
            "BufferedInputStream: " + (time6-time5));
}
```

```java
public static void main(String[] args){
    BufferTest test = new BufferTest(new File("c:/test.dat"));
    try{
        test.test();
    }catch(Exception e){
        e.printStackTrace();
    }
}
}
```

Problems @ Javadoc Declaration Console × Tasks B

<terminated> BufferTest [Java Application] C:\Program Files\Java

FileOutputStream: 1218   FileInputStream: 672
BufferedOutputStream: 31   BufferedInputStream: 62

- InputStreamReader

- OutputStreamWriter

Using UTF-8

```java
FileInputStream fileInput = new FileInputStream(tempFile);
InputStreamReader inputStrReader = new InputStreamReader(fileInput, "UTF-8");
BufferedReader buffereReader = new BufferedReader(inputStrReader);

FileOutputStream fileOutput = new FileOutputStream(tempFile);
PrintWriter fileWrite = new PrintWriter(new OutputStreamWriter(fileOutput, "UTF-8"));
BufferedWriter buffereWriter = new BufferedWriter(fileWrite);
```

About Encoding in Java

- http://www.ibm.com/developerworks/cn/java/j-lo-chinesecoding/

Random Access File

- java.io.RandomAccessFile

  - RandomAccessFile is used for fixed length records

  - Using seek(long position) to locate

  - Nothing to do with InputStream and OutputStream

  - Can be used like DataInputStream and DataOutputStream

  - Often used for building index of Search Engines

file pointer

byte0	byte1	byte2	byte3	byte4	byte5	byte6	byte7	EOF

- # System.in

  o InputStream

  o Input from keyboard

- # System.out

  o PrintStream ->FilterOutputStream -> OutputStream

  o Show information in console

- # System.err

  o PrintStream ->FilterOutputStream -> OutputStream

  o Show error information in console

```java
import java.io.IOException;
public class TranslateByte {
    public static void main(String[] args) throws IOException
    {
        if(args.length<2){
            System.err.println("Usage: Java TranslationByte FROM TO");
            return;
        }
        byte from = (byte) args[0].charAt(0);
        byte to   = (byte) args[1].charAt(0);
        int b;
        System.out.print("Input the source: ");
        while ((b = System.in.read()) != -1)
            System.out.write(b == from ? to : b);
    }
}
```

```
c:\> java TranslateByte b B
aaabbb
```

Best Practice

- Common Programming Errors
  - Use FileOutputStream to write to an existing file – the existing content will be erased.
  - Path errors - \ and \\
- Good Programming Habits
  - Choose mode r for read-only for RandomAccessFile
  - Judge the existence of a file before using FileOuputStream
  - Use buffer as possible as you can
  - Remember to close the stream

- Motivation：High Speed I/O

- Mapped Memory File 内存映射文件
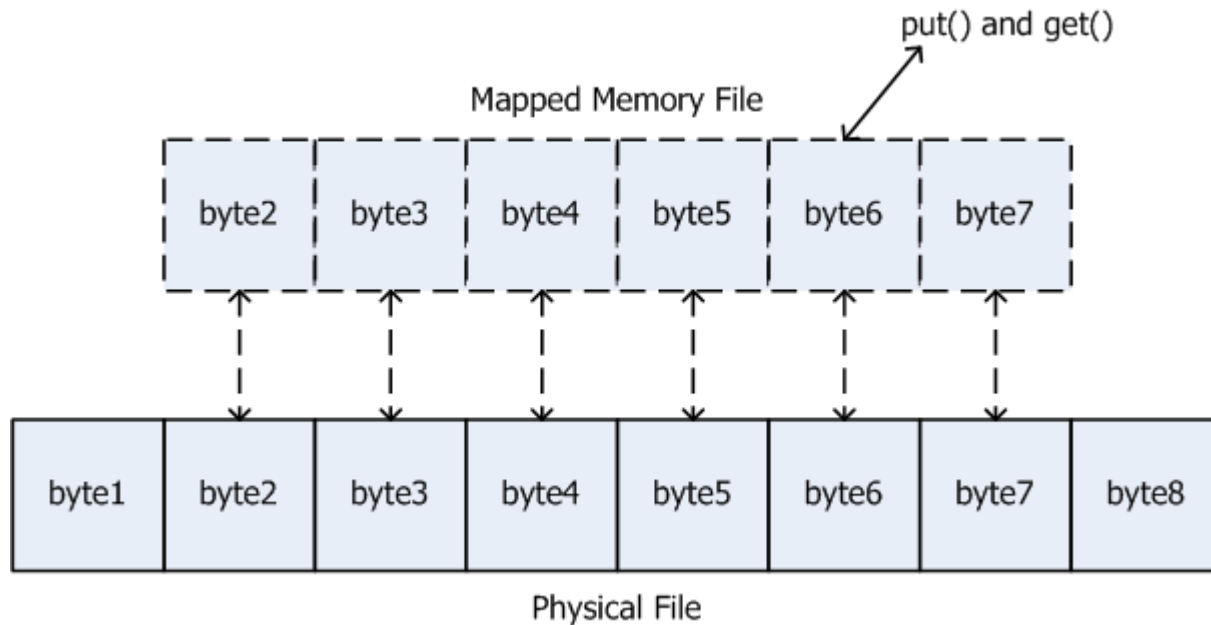
- ## Example:

  - Use MappedByteBuffer to create a 128M file, with each byte be a binary representation of char '*x*';

  - Read 6 bytes from the middle of this file;

```java
import java.io.*;
import java.nio.*;
import java.nio.channels.*;

public class LargeMappedFiles {
    static int length = 0x8FFFFFF; // 128 Mb
    public static void main(String[] args) throws Exception {
        MappedByteBuffer out = new RandomAccessFile("d:/test.dat", "rw")
                .getChannel().map(FileChannel.MapMode.READ_WRITE, 0, length);
        long begin = System.currentTimeMillis();
        for (int i = 0; i < length; i++)
            out.put((byte) 'x');
        long end = System.currentTimeMillis();
        System.out.println("Finished writing using " + (end-begin) + " ms.");
        for (int i = length / 2; i < length / 2 + 6; i++)
            System.out.print((char) out.get(i));
    }
}
```

- Benchmark these class:

  o Memory-mapped file vs. DataInputStream

  o Memory-mapped file vs. DataOutputStream

  o Memory-mapped file vs. RandomAccessFile

- Tips:

```java
public abstract class Benchmark{
    public int numOfInts = …
    public abstract void test() // using I/O classes
    public void run() { // benchmarking running time
        long startTime = System.currentTimeMillis();
        test();
        long endTime = System.currentTimeMillis();
        …
    }
}
```

- Tips
  - DataOutputStream

```java
DataOutputStream dos = new DataOutputStream(
        new BufferedOutputStream(
        new FileOutputStream(new File("temp.tmp"))));
```

  - DataInputStream

```java
DataInputStream dis = new DataInputStream(
        new BufferedInputStream(
        new FileInputStream(new File("temp.tmp"))));
```

- RandomAccessFile

```java
RandomAccessFile raf = new RandomAccessFile(file, "rw");
```
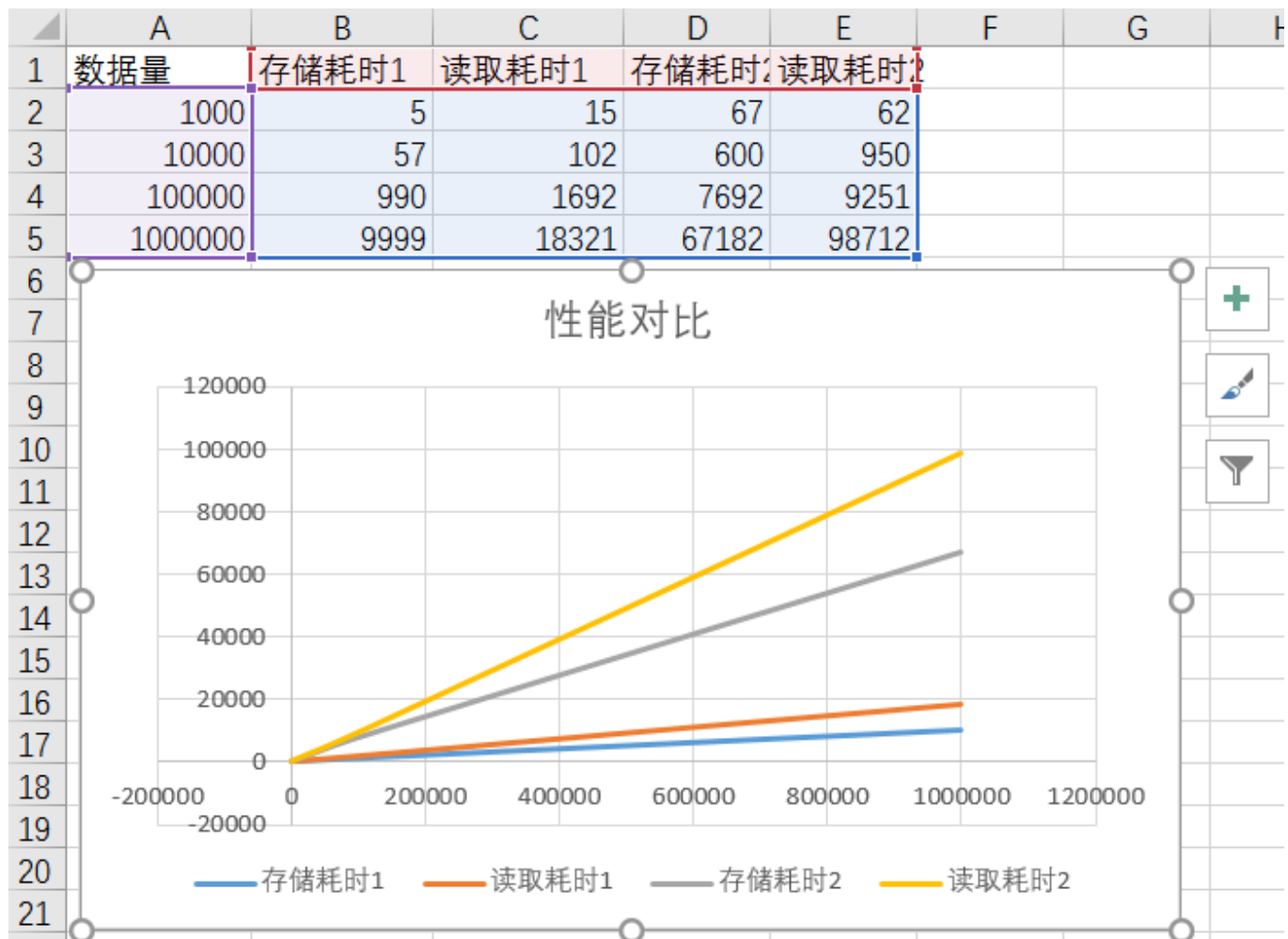
- Mapped File

```java
FileChannel fc = new RandomAccessFile("temp.tmp", "rw").getChannel();
MappedByteBuffer ib = fc.map(FileChannel.MapMode.READ_WRITE, 0, fc.size());
```

- Setup an Benchmark Environment

- Design a Benchmark Case

- Run and Gain the Efficient of Each I/O Class

- * Evaluate and Analysis the Performance Curve

using Excel to draw a scatter plot with curve

- Serializable Object

- Object serialization

```java
public class Person implements Serializable{
    ...
    public static void main(String[] args){
        Person tom = new Person();
        ...
        FileOutputStream fos = new FileOutputStream("person.dat");
        ObjectOutputStream oos = new ObjectOutputStream(fos);
        oos.writeObject(tom);
        ...
    }
}
```
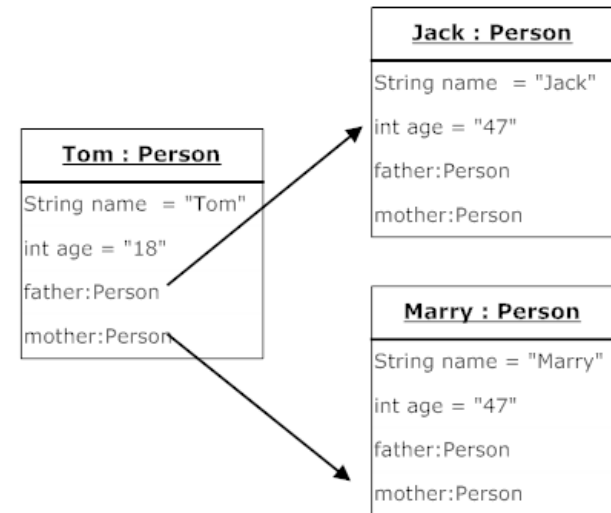
Tom : Person

String name = "Tom"

int age = "18"

father:Person

mother:Person

Jack : Person

String name = "Jack"

int age = "47"

father:Person

mother:Person

Marry : Person

String name = "Marry"

int age = "47"

father:Person

mother:Person

Lab Work

- ATM machine with persistency (持久化存储)

- Requirement

  - Single user;

  - Query / Deposit / Withdrawl / Adding Interest (annual 5%);

  - Using DataInput/OutputStream or ObjectInput/OutStream to store user data;

Self-study

- Reading

  - *The.Java.Programming.Language.4th.Edition,* Chapter 20;

  - *Thinking in Java, 3th Edition,* Chapter 12.

- Arrays

- Collection
  - ArrayList
  - LinkedList

- Map
  - HashMap

- Iterator