Input/Output

Dasar – Dasar Pemrograman 2

Pudy Prima

References

- Liang, Introduction to Java Programming, 11th Edition, Ch. 12 & 17

Text I/O

The File Class

The **File** class is intended to provide an abstraction that deals with most of the machine-dependent complexities of files and path names in a machine-independent fashion.
The filename is a string.
The File class is a wrapper class for the file name and its directory path.

Obtaining file properties and manipulating file

java.io.File	
`+File(pathname: String)`	Creates a `File` object for the specified path name. The path name may be a directory or a file.
`+File(parent: String, child: String)`	Creates a `File` object for the child under the directory parent. The child may be a file name or a subdirectory.
`+File(parent: File, child: String)`	Creates a `File` object for the child under the directory parent. The parent is a `File` object. In the preceding constructor, the parent is a string.
`+exists(): boolean`	Returns true if the file or the directory represented by the `File` object exists.
`+canRead(): boolean`	Returns true if the file represented by the `File` object exists and can be read.
`+canWrite(): boolean`	Returns true if the file represented by the `File` object exists and can be written.
`+isDirectory(): boolean`	Returns true if the `File` object represents a directory.
`+isFile(): boolean`	Returns true if the `File` object represents a file.
`+isAbsolute(): boolean`	Returns true if the `File` object is created using an absolute path name.
`+isHidden(): boolean`	Returns true if the file represented in the `File` object is hidden. The exact definition of *hidden* is system-dependent. On Windows, you can mark a file hidden in the File Properties dialog box. On Unix systems, a file is hidden if its name begins with a period(.) character.
`+getAbsolutePath(): String`	Returns the complete absolute file or directory name represented by the `File` object.
`+getCanonicalPath(): String`	Returns the same as `getAbsolutePath()` except that it removes redundant names, such as "." and "..", from the path name, resolves symbolic links (on Unix), and converts drive letters to standard uppercase (on Windows).
`+getName(): String`	Returns the last name of the complete directory and file name represented by the `File` object. For example, new `File("c:\\book\\test.dat").getName()` returns `test.dat`.
`+getPath(): String`	Returns the complete directory and file name represented by the `File` object. For example, new `File("c:\\book\\test.dat").getPath()` returns `c:\book\test.dat`.
`+getParent(): String`	Returns the complete parent directory of the current directory or the file represented by the `File` object. For example, new `File("c:\\book\\test.dat").getParent()` returns `c:\book`.
`+lastModified(): long`	Returns the time that the file was last modified.
`+length(): long`	Returns the size of the file, or 0 if it does not exist or if it is a directory.
`+listFile(): File[]`	Returns the files under the directory for a directory `File` object.
`+delete(): boolean`	Deletes the file or directory represented by this `File` object.The method returns true if the deletion succeeds.
`+renameTo(dest: File): boolean`	Renames the file or directory represented by this `File` object to the specified name represented in dest. The method returns true if the operation succeeds.
`+mkdir(): boolean`	Creates a directory represented in this `File` object. Returns true if the the directory is created successfully.
`+mkdirs(): boolean`	Same as `mkdir()` except that it creates directory along with its parent directories if the parent directories do not exist.

Text I/O

- ✓ A File object encapsulates the properties of a file or a path, but does not contain the methods for reading/writing data from/to a file.
- ✓ In order to perform I/O, you need to create objects using appropriate Java I/O classes.
- ✓ The objects contain the methods for reading/writing data from/to a file.

Writing Data Using PrintWriter

java.io.PrintWriter	
+PrintWriter(filename: String)	Creates a PrintWriter for the specified file.
+print(s: String): void	Writes a string.
+print(c: char): void	Writes a character.
+print(cArray: char[]): void	Writes an array of character.
+print(i: int): void	Writes an int value.
+print(l: long): void	Writes a long value.
+print(f: float): void	Writes a float value.
+print(d: double): void	Writes a double value.
+print(b: boolean): void	Writes a boolean value.
Also contains the overloaded println methods.	A println method acts like a print method; additionally it prints a line separator. The line separator string is defined by the system. It is \r\n on Windows and \n on Unix.
Also contains the overloaded printf methods.	The printf method was introduced in §3.6, "Formatting Console Output and Strings."

Writing Data Using PrintWriter

```
File file = new File("scores.txt");

// Create a file
PrintWriter output = new PrintWriter(file);

// Write formatted output to the files
output.print("John T Smith ");
output.println(90);

// Close the file
output.close();
```

Catatan:
Constructor PrintWriter mengandung deklarasi **throws suatu checked exception**, sehingga instansiasinya harus disertai penanganan exception.

Try-with-resources

Programmers often forget to close the file. JDK 7 provides the followings new try-with-resources syntax that automatically closes the files.

try (declare and create resources) {
  Use the resource to process the file;
}

```
...
try (PrintWriter output = new PrintWriter(file)) {
    output.print("John T Smith ");
    output.println(90);
} catch (FileNotFoundException e) {
    e.printStackTrace();
}
...
```

Reading Data Using Scanner

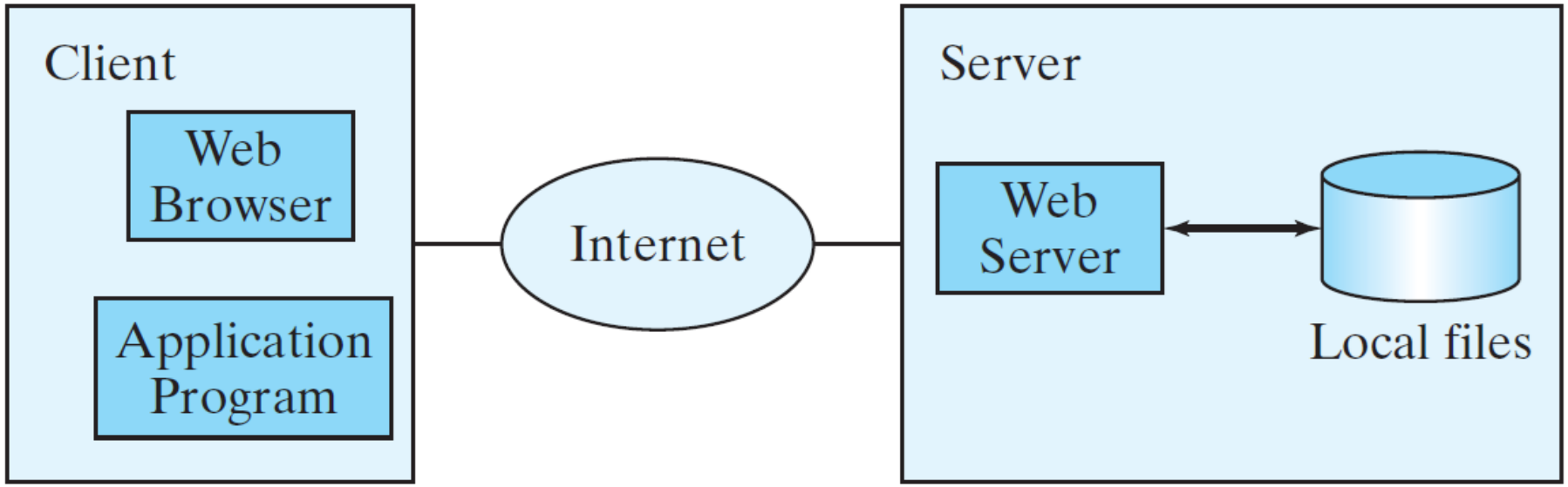
java.util.Scanner	
+Scanner(source: File)	Creates a Scanner object to read data from the specified file.
+Scanner(source: String)	Creates a Scanner object to read data from the specified string.
+close()	Closes this scanner.
+hasNext(): boolean	Returns true if this scanner has another token in its input.
+next(): String	Returns next token as a string.
+nextByte(): byte	Returns next token as a byte.
+nextShort(): short	Returns next token as a short.
+nextInt(): int	Returns next token as an int.
+nextLong(): long	Returns next token as a long.
+nextFloat(): float	Returns next token as a float.
+nextDouble(): double	Returns next token as a double.
+useDelimiter(pattern: String): Scanner	Sets this scanner's delimiting pattern.

Reading Data Using Scanner

```
File file = new File("scores.txt");

// Create a Scanner for the file
Scanner input = new Scanner(file);

// Read data from file
String text = input.next();

// Close the file
input.close();
```


Reading Data from the Web

Just like you can read data from a file on your computer, you can read data from a file on the Web.



Reading Data from the Web

URL url = **new** URL(**"www.google.com/index.html"**);

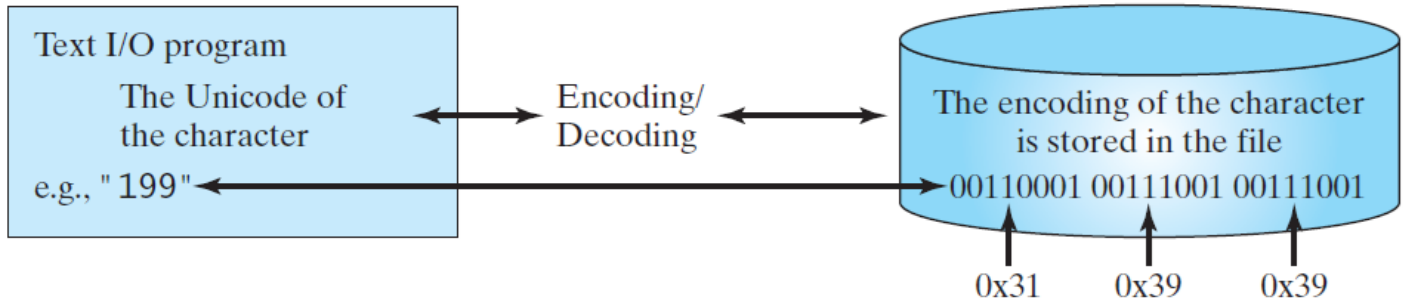
After a **URL** object is created, you can use the **openStream()** method defined in the **URL** class to open an input stream and use this stream to create a **Scanner** object as follows:

Scanner input = **new** Scanner(url.openStream());

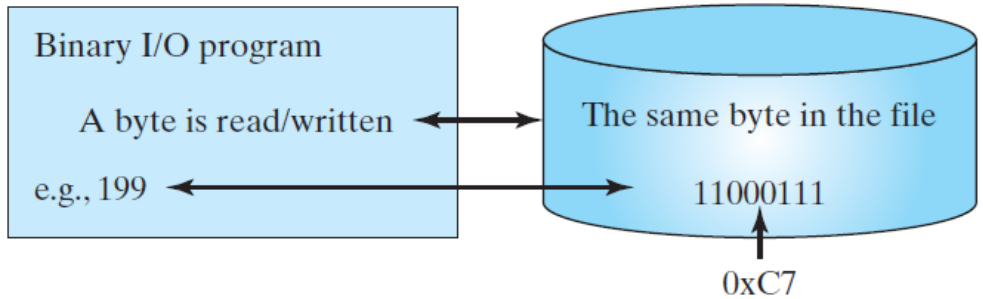
Binary I/O

Text File vs. Binary File

- Data stored in a text file are represented in human-readable form. Data stored in a binary file are represented in binary form (not human-readable).
- For example, the Java source programs are stored in text files and can be read by a text editor, but the Java classes are stored in binary files and are read by the JVM.
- Binary files processing is more efficient because it does not require encoding/decoding process.

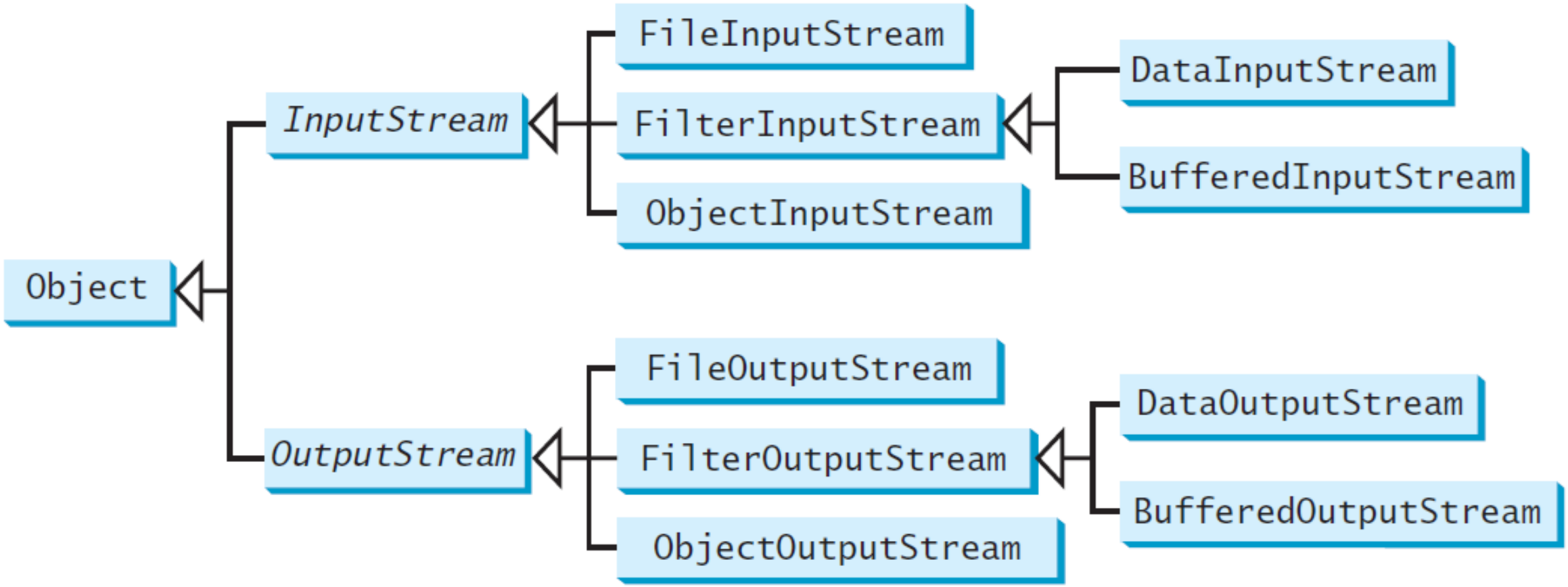


(a)



(b)

Binary I/O Classes



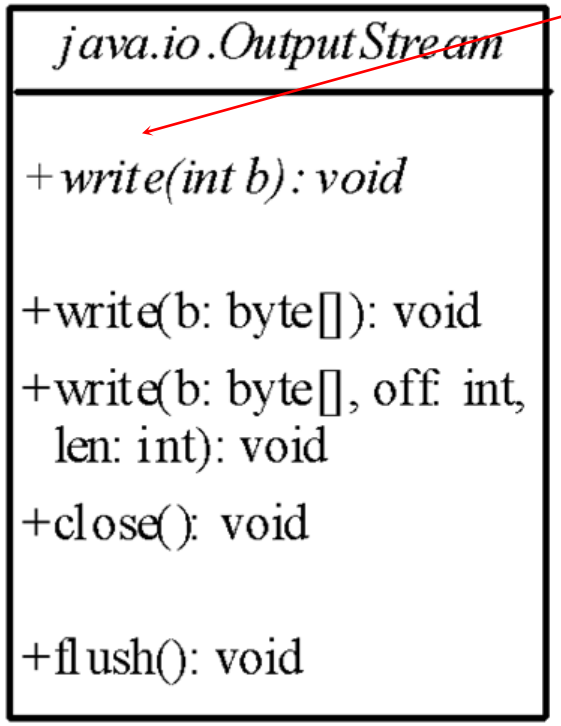
InputStream

The value returned is a byte as an int type.

java.io.InputStream	
+read(): int	Reads the next byte of data from the input stream. The value byte is returned as an int value in the range 0 to 255. If no byte is available because the end of the stream has been reached, the value –1 is returned.
+read(b: byte[]): int	Reads up to b.length bytes into array b from the input stream and returns the actual number of bytes read. Returns -1 at the end of the stream.
+read(b: byte[], off: int, len: int): int	Reads bytes from the input stream and stores into b[off], b[off+1], …, b[off+len-1]. The actual number of bytes read is returned. Returns -1 at the end of the stream.
+available(): int	Returns the number of bytes that can be read from the input stream.
+close(): void	Closes this input stream and releases any system resources associated with the stream.
+skip(n: long): long	Skips over and discards n bytes of data from this input stream. The actual number of bytes skipped is returned.
+markSupported(): boolean	Tests if this input stream supports the mark and reset methods.
+mark(readlimit: int): void	Marks the current position in this input stream.
+reset(): void	Repositions this stream to the position at the time the mark method was last called on this input stream.

OutputStream

The value is a byte as an int type.



Writes the specified byte to this output stream. The parameter b is an int value. (byte)b is written to the output stream.

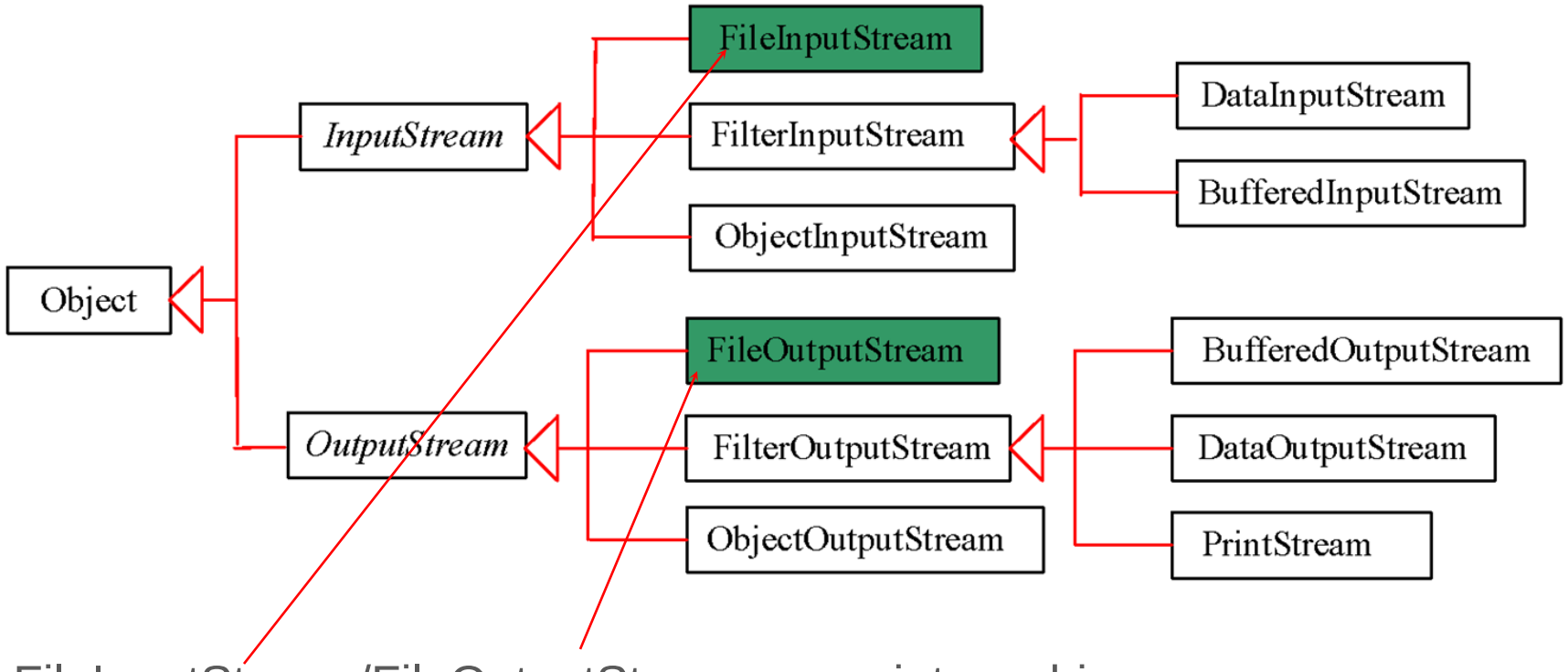
Writes all the bytes in array b to the output stream.

Writes b[off], b[off+1], …, b[off+len-1] into the output stream.

Closes this output stream and releases any system resources associated with the stream.

Flushes this output stream and forces any buffered output bytes to be written out.

FileInputStream/FileOutputStream



FileInputStream/FileOutputStream associates a binary input/output stream with an external file.
All the methods in FileInputStream/FileOuptputStream are inherited from its superclasses.

Writing Data Using PrintOutputStream

```
String fileName = "temp.dat";

// Create a file
FileOutputStream fos = new FileOutputStream(fileName);

// Write output to the files
fos.write(1000);
fos.write(900);

// Close the file
fos.close();
```

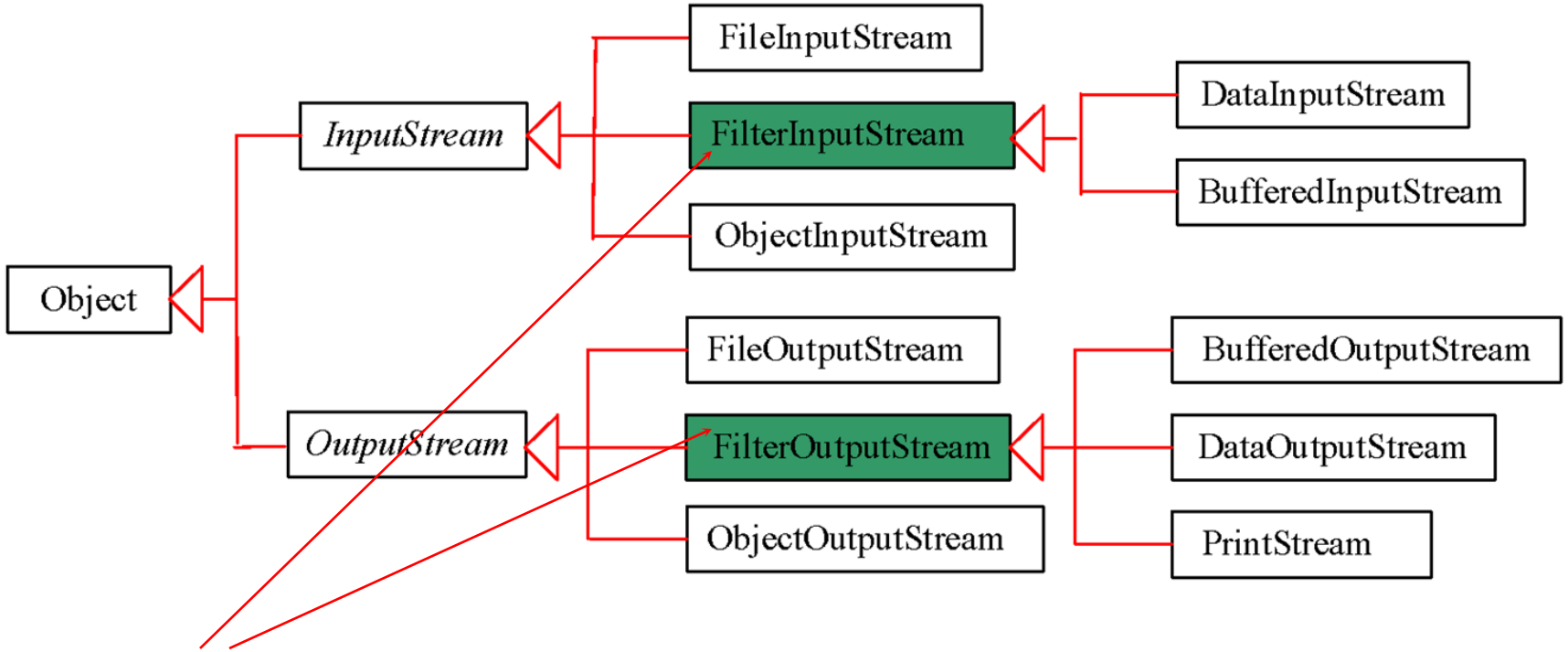
Catatan:
Constructor dan method write pada FileOutputStream mengandung deklarasi **throws suatu checked exception**, sehingga instansiasinya harus disertai penanganan exception.

Method write() hanya dapat menerima parameter berupa byte[] atau int (nilai int akan dikonversi menjadi byte).

Reading Data Using PrintInputStream

```
String fileName = "temp.dat";

// Create a file
FileInputStream fis = new FileInputStream(fileName);

// Read data from file
System.out.println(fis.read());
System.out.println(fis.read());

// Close the file
fis.close();
```

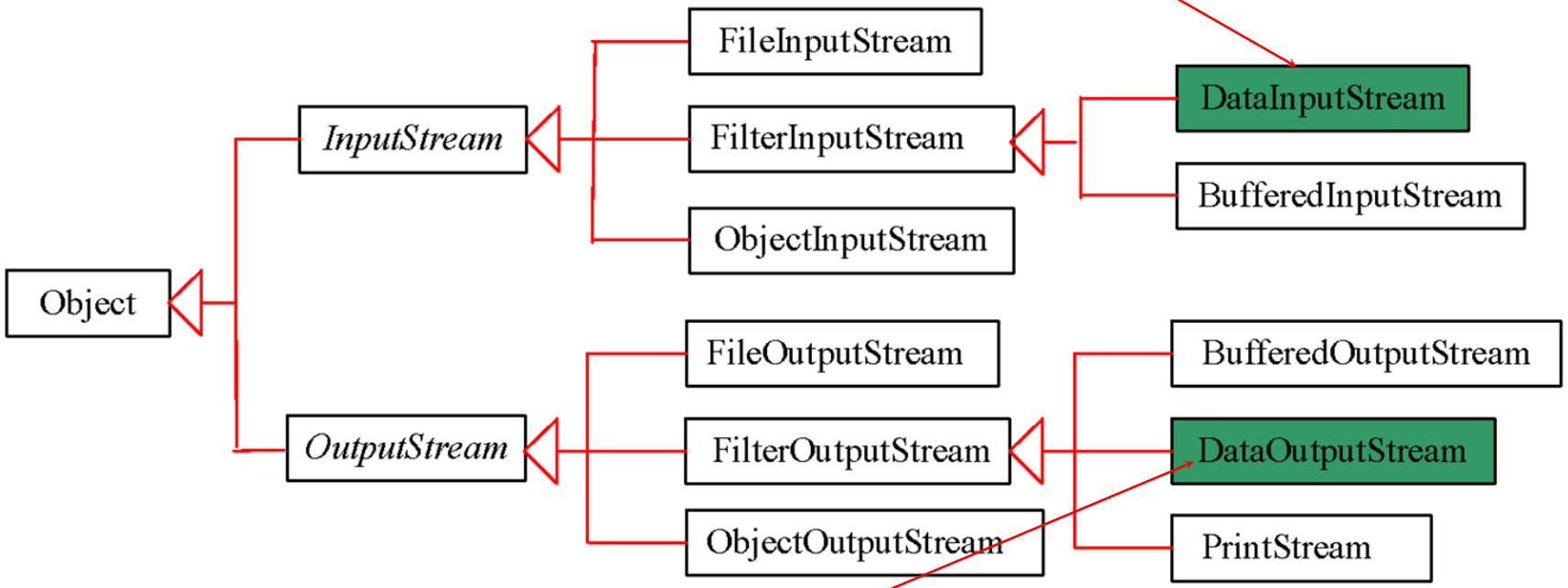
FilterInputStream/FilterOutputStream



Filter streams are streams that filter bytes for some purpose. The basic byte input stream provides a read method that can only be used for reading bytes. If you want to read integers, doubles, or strings, you need a filter class to wrap the byte input stream. Using a filter class enables you to read integers, doubles, and strings instead of bytes and characters. FilterInputStream and FilterOutputStream are the base classes for filtering data. When you need to process primitive numeric types, use DatInputStream and DataOutputStream to filter bytes.

DataInputStream/DataOutputStream

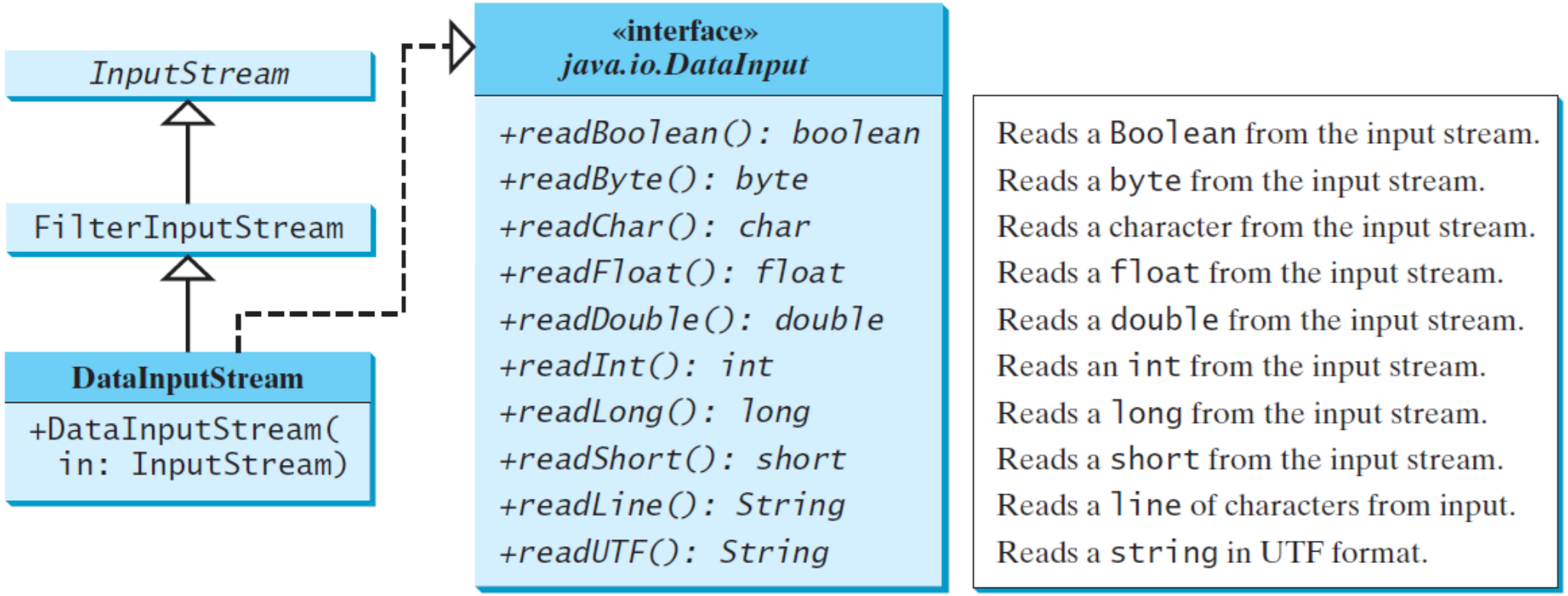
DataInputStream reads bytes from the stream and converts them into appropriate primitive type values or strings.



DataOutputStream converts primitive type values or strings into bytes and output the bytes to the stream.

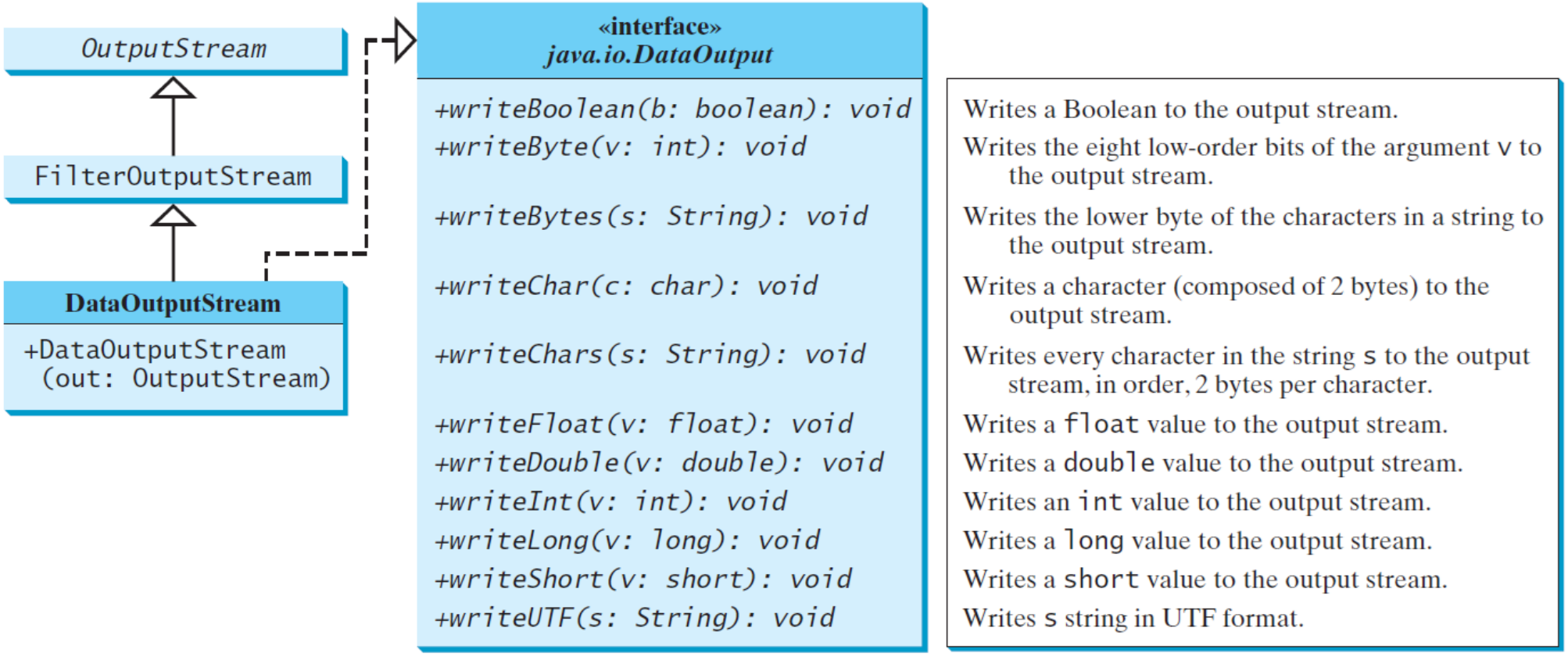
DataInputStream

DataInputStream extends FilterInputStream and implements the DataInput interface.



DataOutputStream

DataOutputStream extends FilterOutputStream and implements the DataOutput interface.



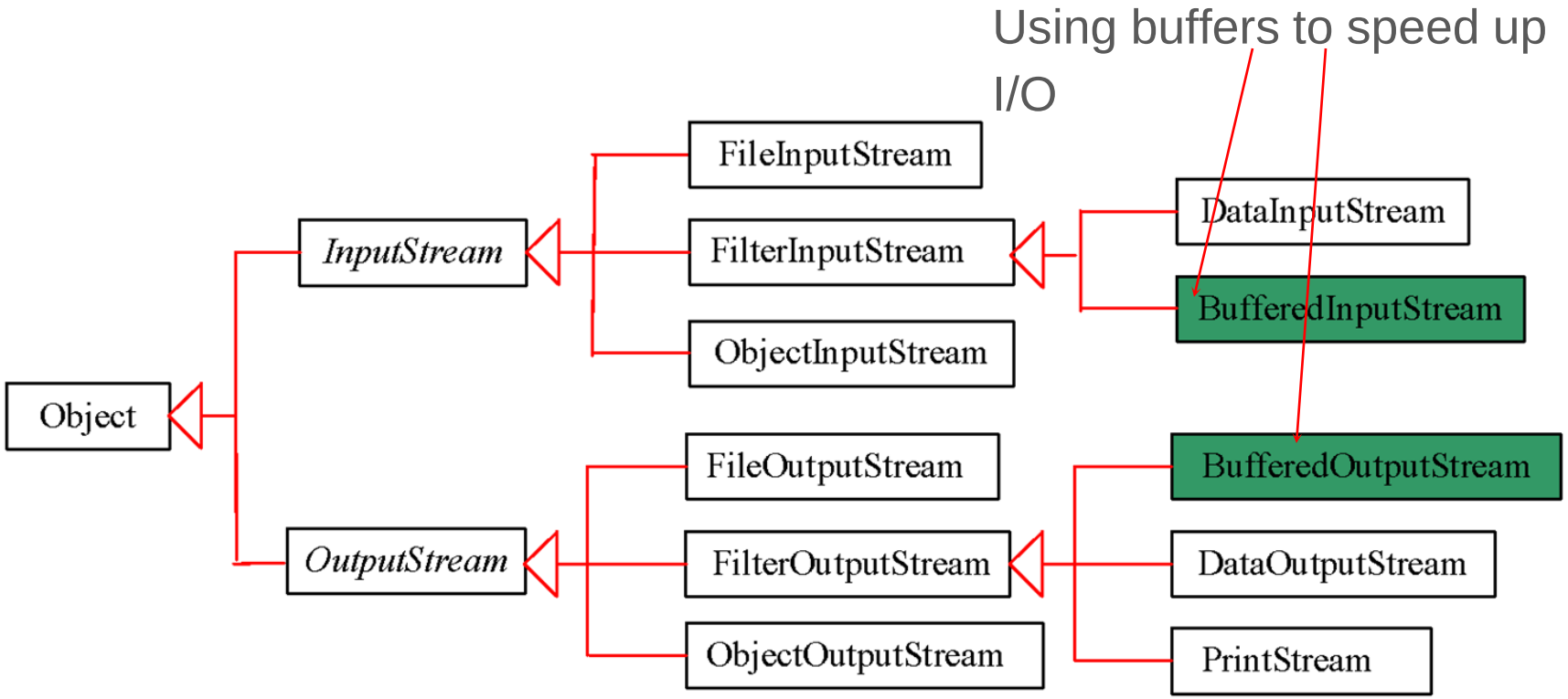
Writing Data Using DataOutputStream

```
String fileName = "temp.dat";

// Create a file
FileOutputStream fos = new FileOutputStream(fileName);
DataOutputStream output = new DataOutputStream(fos);

// Write output to the files
fos.writeUTF("John");
fos.writeDouble(85.5);

// Close the file
output.close();
```


Reading Data Using DataInputStream

```
String fileName = "temp.dat";

// Create a file
FileInputStream fis = new FileInputStream(fileName);
DataInputStream input = new DataInputStream(fis);

// Read data from file
System.out.println(input.readUTF());
System.out.println(fis.readDouble());

// Close the file
input.close();
```


BufferedInputStream/BufferedOutputStream

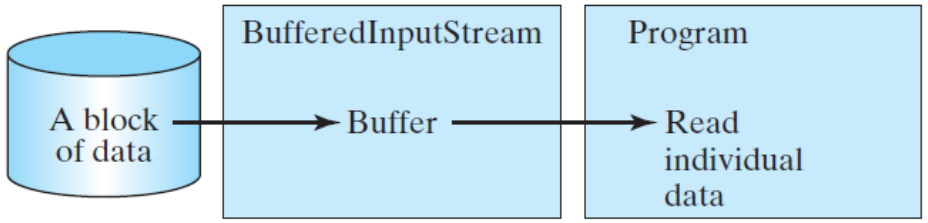


BufferedInputStream/BufferedOutputStream does not contain new methods. All the methods BufferedInputStream/BufferedOutputStream are inherited from the InputStream/OutputStream classes.

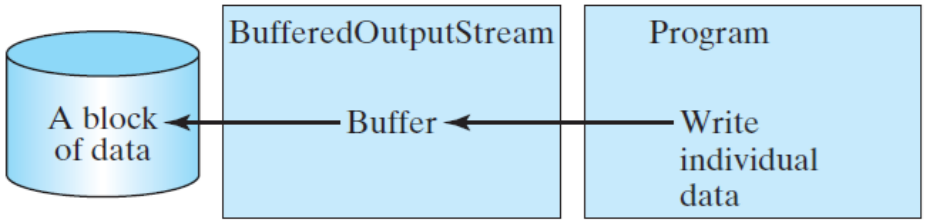
Constructing BufferedInputStream/BufferedOutputStream

```
// Create a BufferedInputStream
public BufferedInputStream(InputStream in)
public BufferedInputStream(InputStream in, int bufferSize)

// Create a BufferedOutputStream
public BufferedOutputStream(OutputStream out)
public BufferedOutputStream(OutputStreamr out, int bufferSize)
```

Catatan:
BufferedInputStream dan BufferedOutputStream dapat digunakan untuk mempercepat proses penulisan dan pembacaan data dengan cara mengurangi jumlah pembacaan langsung dari atau penulisan langsung dari disk.



(a)



(b)