

Програмування-1

Лекція 14
ІО (Input-Output)

Бібліотеки IO

- **IO**
 - blocking, stream-oriented
 - `java.io.*`
- **NIO**
 - Java 1.4 +
 - non-blocking, buffer-oriented
 - `java.nio.*`
- **NIO.2**
 - Java 7 +
 - asynchronous, improved filesystem operations
 - `java.nio.*`

java.io

	Байти	Символи
Input	InputStream	Reader
Output	OutputStream	Writer

InputStream, OutputStream, Reader, Writer

- Абстракті
- «Знають» як читати, але «не вміють» працювати з конкретними джерелами
- Мають купу конкретних нащадків, які вміють працювати з конкретними джерелами

InputStream

```
abstract int read()
int read(byte[] b)
int read(byte[] b, int off, int len)
```

```
void close()
```

Не забувайте викликати **close()**

або

використовувати конструкцію
try-with-resources

```
boolean markSupported()
void mark(int readlimit)
void reset()
```

```
long skip(long n)
```

```
int available()
```

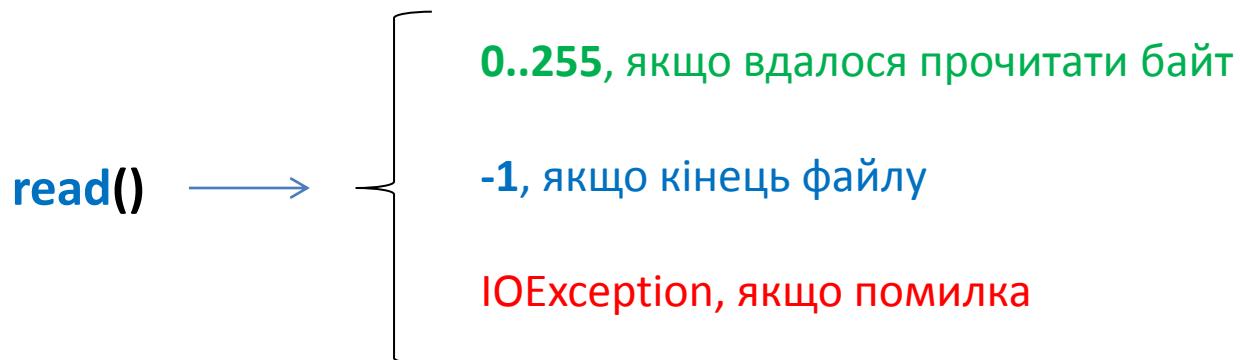
available робить зовсім не те, що
можна подумати з його назви
(see JavaDoc)

Скоріше за все він вам не потрібен

InputStream

```
public abstract int read()  
throws IOException
```

Чому повертає **int** якщо ми читаємо **байт**?



InputStream.read() Example

```
8  [-] import java.io.*;
9
10 [-] public class InputStreamExample {
11
12     [-]     void print(InputStream inputStream) throws IOException {
13         while(true) {
14             int value = inputStream.read();
15
16             // check EOF
17             if(value == -1)
18                 return; // EOF: End Of File - exit
19
20             byte b = (byte) value;
21
22             // Do something with byte
23             System.out.println(b);
24         }
25     }
26 }
```

InputStream.read() Example (short version)

```
8  import java.io.*;
9
10 public class InputStreamExample {
11
12     void print(InputStream inputStream) throws IOException {
13         int value;
14         while ((value = inputStream.read()) != -1) {
15             System.out.println((byte) value);
16         }
17     }
18 }
```

Performance

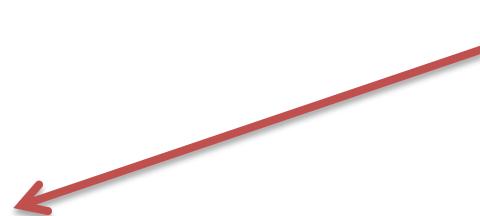
- Не використовуйте `InputStream.read()` у циклі для обробки великих об'ємів даних
 - це дуже просто, але працює дуже повільно
- Використовуйте `InputStream.read(byte[] b)` та `InputStream.read(byte[] b, int off, int len)`
- Використовуйте буферізацію

InputStream.read(byte[] buf)

- Читає дані з потоку
 - не більше ніж **buf.length** байт
- записує їх у **buf[]**
 - починаючи з **buf[0]**
- повертає кількість прочитаних байтів
 - **1..buf.length** у разі успіху
 - **-1** у разі кінця файлу
 - **IOException** у разі помилки

InputStream.read(byte[] buf) Example

```
19  void print2(InputStream inputStream) throws IOException {  
20      byte[] buf = new byte[512];  
21  
22      while (true) {  
23          int readed = inputStream.read(buf);  
24          if (readed == -1) {  
25              return;  
26          }  
27  
28          for (int i = 0; i < readed; i++) {  
29              System.out.println(buf[i]);  
30          }  
31      }  
32  }
```



!!!

InputStream.read(byte[] buf) Example short version

```
19     void print2(InputStream inputStream) throws IOException {
20         byte[] buf = new byte[512];
21
22         int readed;
23         while ((readed = inputStream.read(buf)) != -1) {
24             for (int i = 0; i < readed; i++) {
25                 System.out.println(buf[i]);
26             }
27         }
28     }
```

BufferedInputStream Example

```
12 void print(InputStream inputStream) throws IOException {  
13     int value;  
14     while ((value = inputStream.read()) != -1) {  
15         System.out.println((byte) value);  
16     }  
17 }  
18  
19 void bufAndPrint(InputStream inputStream) throws IOException {  
20     try (BufferedInputStream bis = new BufferedInputStream(inputStream)) {  
21         print(bis);  
22     }  
23 }
```

May be slow

Will work faster

The diagram illustrates two methods for reading from an input stream. The first method, 'print', reads one byte at a time using 'inputStream.read()' and prints it using 'System.out.println()'. This process is highlighted with a red bracket and labeled 'May be slow', indicating it may be inefficient for large files. The second method, 'bufAndPrint', uses a 'BufferedInputStream' to read multiple bytes into a buffer before printing. This process is highlighted with a green bracket and labeled 'Will work faster', indicating it is more efficient for large files.

InputStream.available()

- Returns an estimate of the number of bytes that can be read (or skipped over) from this input stream without blocking by the next invocation of a method for this input stream
- Note that while some implementations of InputStream will return the total number of bytes in the stream, many will not
 - It is never correct to use the return value of this method to allocate a buffer intended to hold all data in this stream

FileInputStream Example

```
public static void main(String[] args) {
    FileInputStream fileInputStream = null;
    try {
        fileInputStream = new FileInputStream("MyFile.bin");

        int value;
        while ((value = fileInputStream.read()) != -1) {
            System.out.println((byte) value);
        }
    } catch (FileNotFoundException ex) {
        System.out.println("Sorry, no such file");
    } catch (IOException ex) {
        System.out.println("Ooops, IO error");
    } finally {
        // finally blocks are guaranteed to be executed
        // close() can throw an IOException too, so we got to wrap that too
        try {
            if (fileInputStream != null) {
                fileInputStream.close();
            }
        } catch (IOException e) {
            // handle an exception or log it
            System.out.println("Ooops, IO error");
        }
    }
}
```

FileInputStream Example 2

try-with-resources

```
public static void main(String[] args) {
    try (FileInputStream fileInputStream = new FileInputStream("f:\\\\MyFile.bin")) {
        int value;
        while ((value = fileInputStream.read()) != -1) {
            System.out.println((byte) value);
        }
    } catch (FileNotFoundException ex) {
        System.out.println("Sorry, no such file");
    } catch (IOException ex) {
        System.out.println("Ooops, IO error");
    }
}
```

FileInputStream Example 3

try-with-resources + throws

```
public static void main(String[] args) throws IOException {
    try (FileInputStream fileInputStream = new FileInputStream("f:\\MyFile.bin")) {
        int value;
        while ((value = fileInputStream.read()) != -1) {
            System.out.println((byte) value);
        }
    }
}
```

OutputStream

abstract void **write**(int b)

void **write**(byte[] b)

void **write**(byte[] b, int off, int len)

void **flush**()

void **close**()

Reader

```
int read()
int read(char[] cbuf)
abstract int read(char[] cbuf, int off, int len)
int read(CharBuffer target)
```

```
boolean markSupported()
void mark(int readAheadLimit)
void reset()
```

```
long skip(long n)
```

```
boolean ready()
```

```
abstract void close()
```

Writer

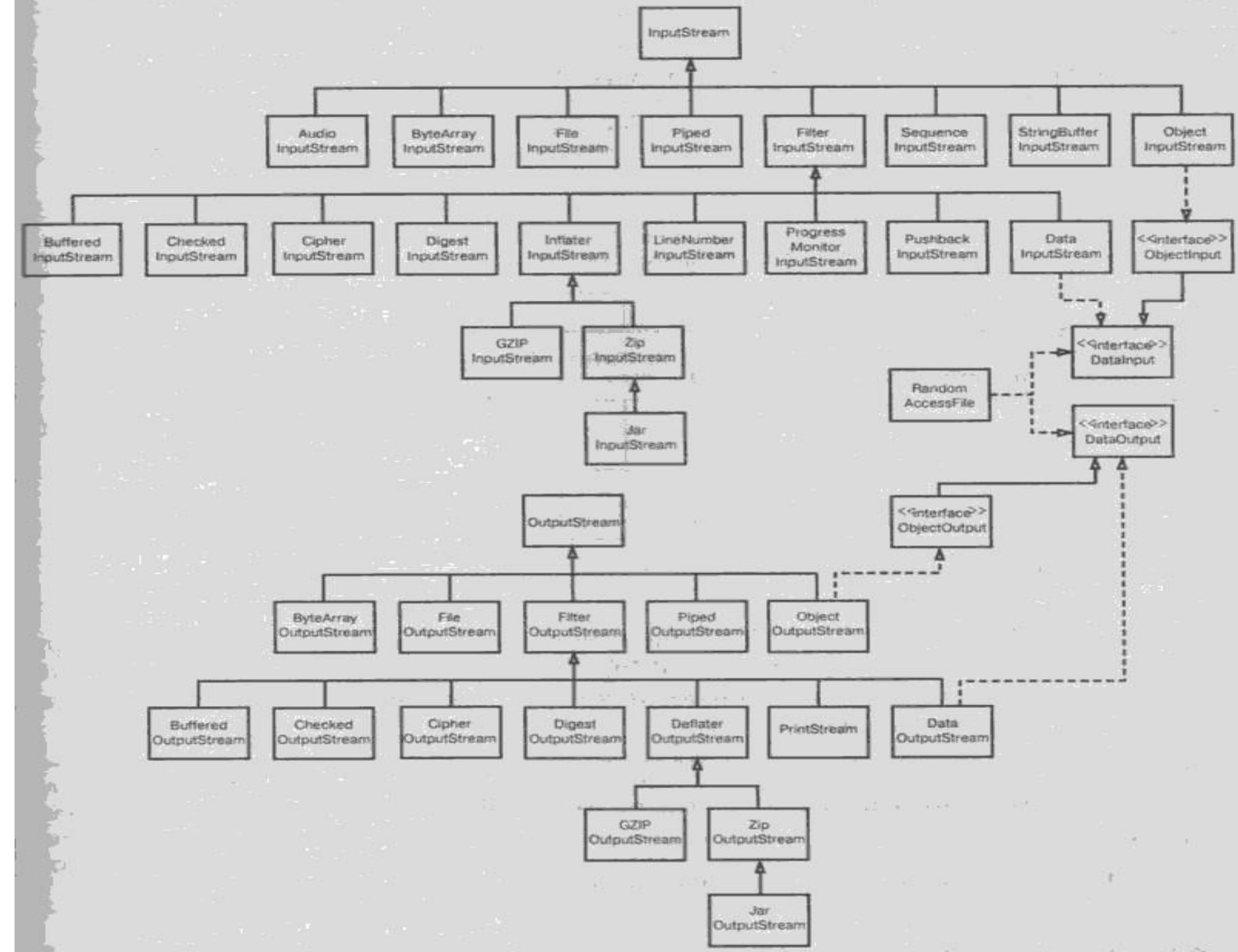
```
void write(int c)
void write(char[] cbuf)
abstract void write(char[] cbuf, int off, int len)
void write(String str)
void write(String str, int off, int len)
```

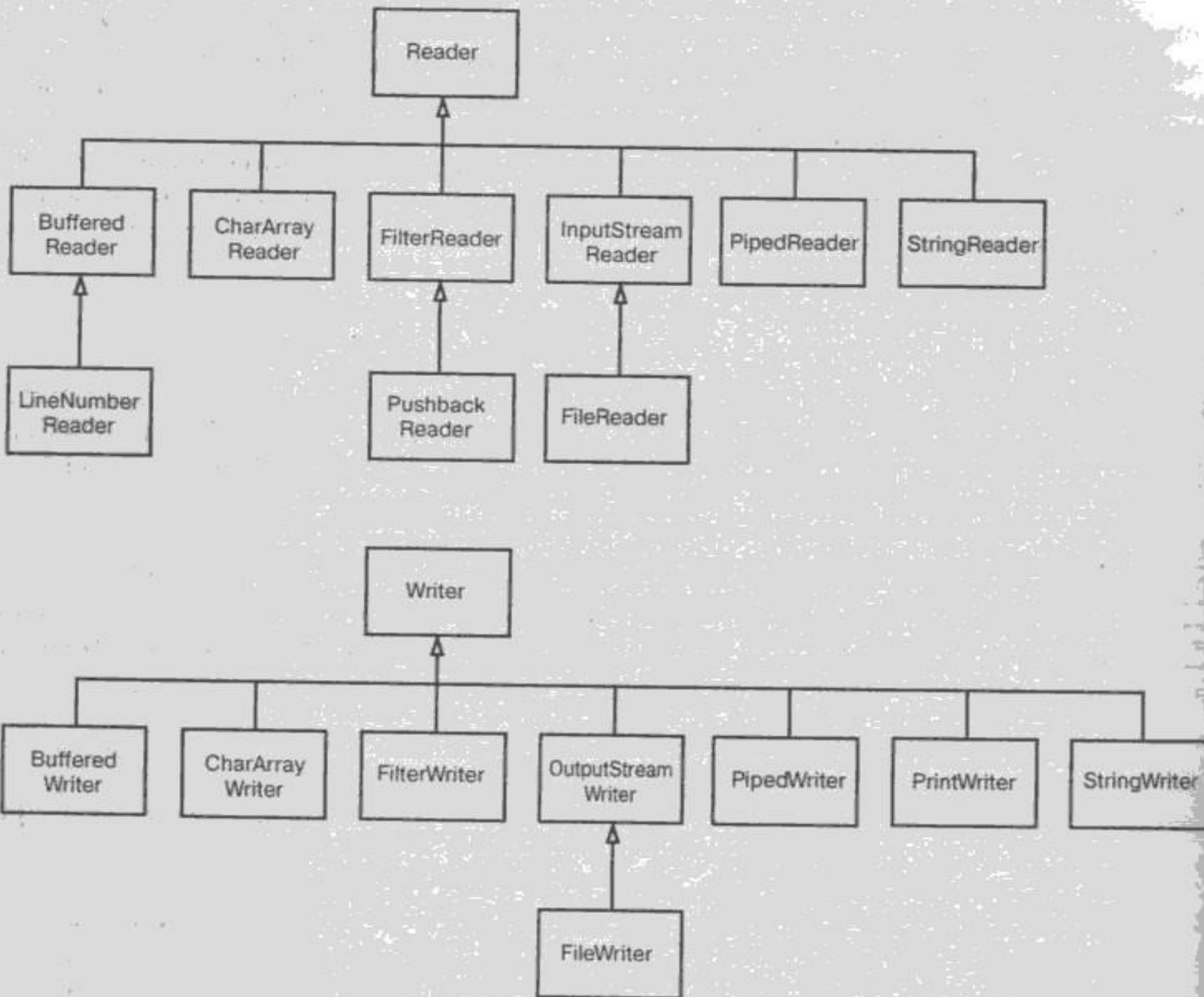
```
Writer append(char c)
Writer append(CharSequence csq)
Writer append(CharSequence csq, int start, int end)
```

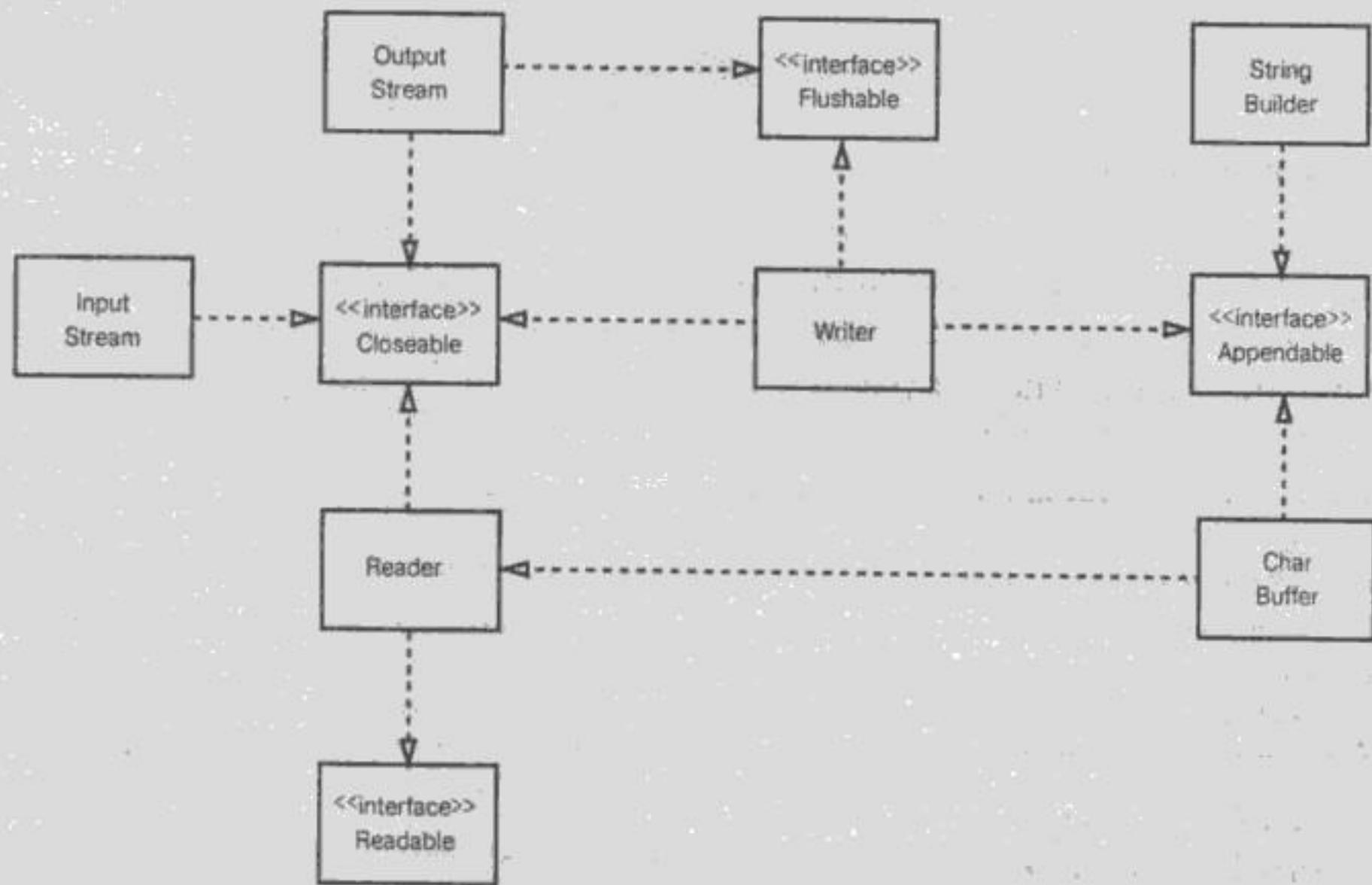
```
abstract void close()
abstract void flush()
```

Writer append() Example

```
public class AppendExample {  
    public static void main(String[] args) throws IOException {  
  
        try (Writer writer = new PrintWriter(System.out)) {  
            writer.append("Hello, this is a string #1\n")  
                .append("and string #2\n")  
                .append("and #3 too\n")  
                .append("#4 \n");  
        }  
    }  
}
```



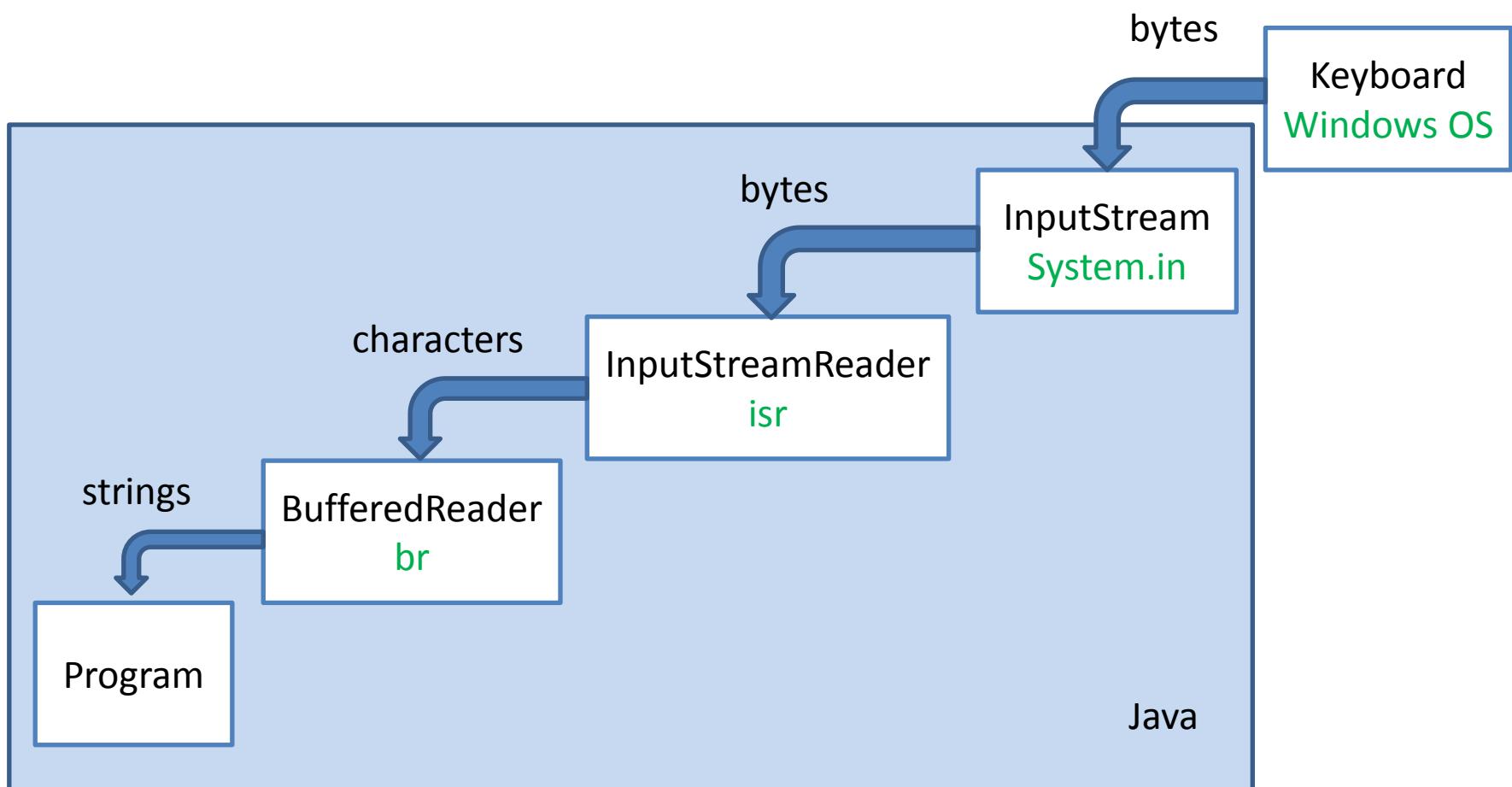




Input chain Demo

```
import java.io.*;  
  
public class KeyboardInput {  
  
    public static void main(String args[]) throws IOException {  
        try (InputStreamReader isr = new InputStreamReader(System.in);  
             BufferedReader br = new BufferedReader(isr)) {  
  
            System.out.println("Unix: Type ctrl-d to exit."  
                               + "\nWindows: Type ctrl-z to exit");  
  
            String str;  
            while ((str = br.readLine()) != null) {  
                System.out.println("Readed: " + str);  
            }  
        }  
    }  
}
```

Input chain Demo



Text File Reader Demo

```
8  import java.io.*;
9
10 public class TextFileReaderDemo {
11
12     public static void main(String[] args) throws IOException {
13         try (FileInputStream fis = new FileInputStream("f:\\\\MyFile.txt");
14              InputStreamReader isr = new InputStreamReader(fis);
15              BufferedReader br = new BufferedReader(isr)) {
16
17             String str;
18             int lineNumber = 0;
19             while ((str = br.readLine()) != null) {
20                 System.out.println(++lineNumber + ": " + str);
21             }
22         }
23     }
24 }
```

FileReader Demo

```
8  import java.io.*;
9
10 public class FileReaderDemo {
11
12     public static void main(String[] args) throws IOException {
13         try (FileReader fr = new FileReader("f:\\MyFile.txt");
14              BufferedReader br = new BufferedReader(fr)) {
15
16             String str;
17             int lineNumber = 0;
18             while ((str = br.readLine()) != null) {
19                 System.out.println(++lineNumber + ": " + str);
20             }
21         }
22     }
23 }
```

Серіалізація

(ObjectInputStream / ObjectOutputStream)

- Дозволяє зберігати та завантажувати об'єкти
- Клас об'єкту має **implements Serializable**
- Якщо об'єкт посилається на інші об'єкти – вони також зберігаються/завантажуються
- **transient**-поля не зберігаються
 - при завантаженні вони ініціалізуються значеннями за замовчуванням
- При завантаженні об'єкта його клас має бути тієї ж версії що і при збереженні
 - **static final long serialVersionUID = <class_version>;**
 - якщо версія не задана явно, то компілятор генерує номер версії на основі вихідного коду цього класу

Серіалізація – приклад (1)

```
import java.util.*;  
import java.io.*;  
  
abstract class Department implements Serializable {  
  
    String title;  
  
    Department(String title) {  
        this.title = title;  
    }  
}
```

Серіалізація – приклад (2)

```
class Kafedra extends Department {  
    /*transient*/ int students;  
  
    Kafedra(String title, int students) {  
        super(title);  
        this.students = students;  
    }  
  
    public String toString() {  
        return title + ": " + students + " students";  
    }  
}
```

Серіалізація – приклад (3)

```
class Facultet extends Department {  
  
    ArrayList<Kafedra> kafedras = new ArrayList<Kafedra>();  
  
    Facultet(String title) {  
        super(title);  
    }  
  
    public String toString() {  
        return title + ":" + kafedras;  
    }  
}
```

Серіалізація – приклад (4)

```
public class Main {  
    public static void main(String[] args) throws  
        IOException, ClassNotFoundException {  
        Facultet fiot = new Facultet("FIOT");  
        fiot.kafedras.add(new Kafedra("AUTS", 45));  
        fiot.kafedras.add(new Kafedra("ASOIU", 45));  
        fiot.kafedras.add(new Kafedra("OT", 50));  
        fiot.kafedras.add(new Kafedra("TK", 40));  
        System.out.println(fiot);  
  
        try (ObjectOutputStream out = new ObjectOutputStream(  
            new FileOutputStream("fiot.bin"))) {  
            out.writeObject(fiot);  
        }  
  
        try (ObjectInputStream in = new ObjectInputStream(  
            new FileInputStream("fiot.bin"))) {  
            Facultet facultet = (Facultet) in.readObject();  
            System.out.println(facultet );  
        }  
    }  
}
```