



Các luồng vào / ra




# Nội dung

- Khái niệm về luồng dữ liệu
- Luồng và tệp
- Lớp File
- Chuỗi hóa đối tượng
- Truy cập tệp tuần tự
- Truy cập tệp ngẫu nhiên



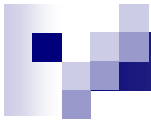
# Tài liệu tham khảo

- *Giáo trình Lập trình HĐT*, chương 12
- *Thinking in Java*, chapter 12
- *Java how to program*, chapter 17



# Luồng dữ liệu (data streams)

- Chương trình Java nhận và gửi dữ liệu thông qua các đối tượng là các thực thể thuộc một kiểu luồng dữ liệu nào đó
- Luồng (stream) là một dòng dữ liệu đến từ một nguồn (source) hoặc đi đến một đích (sink)
- Nguồn và đích có thể là tệp (file), bộ nhớ, một tiến trình (process), hay thiết bị (bàn phím, màn hình, ...)



# Luồng byte và char

- Luồng byte (luồng nhị phân): thao tác theo đơn vị byte
  - ☐ InputStream
  - ☐ OutputStream
- Luồng char: thao tác với ký tự
  - ☐ Reader
  - ☐ Writer



# InputStream

- `int read()`
- `int read(byte buf[])`
- `int read(byte buf[], int offset, int length)`
- `void close()`



# OutputStream

- `int write(int c)`
- `int write(byte buf[])`
- `int write(byte buf[], int offset, int length)`
- `void close()`
- `void flush()`



# Reader

- `int read()`
- `int read(char buf[])`
- `int read(char buf[], int offset, int length)`
- `void close()`

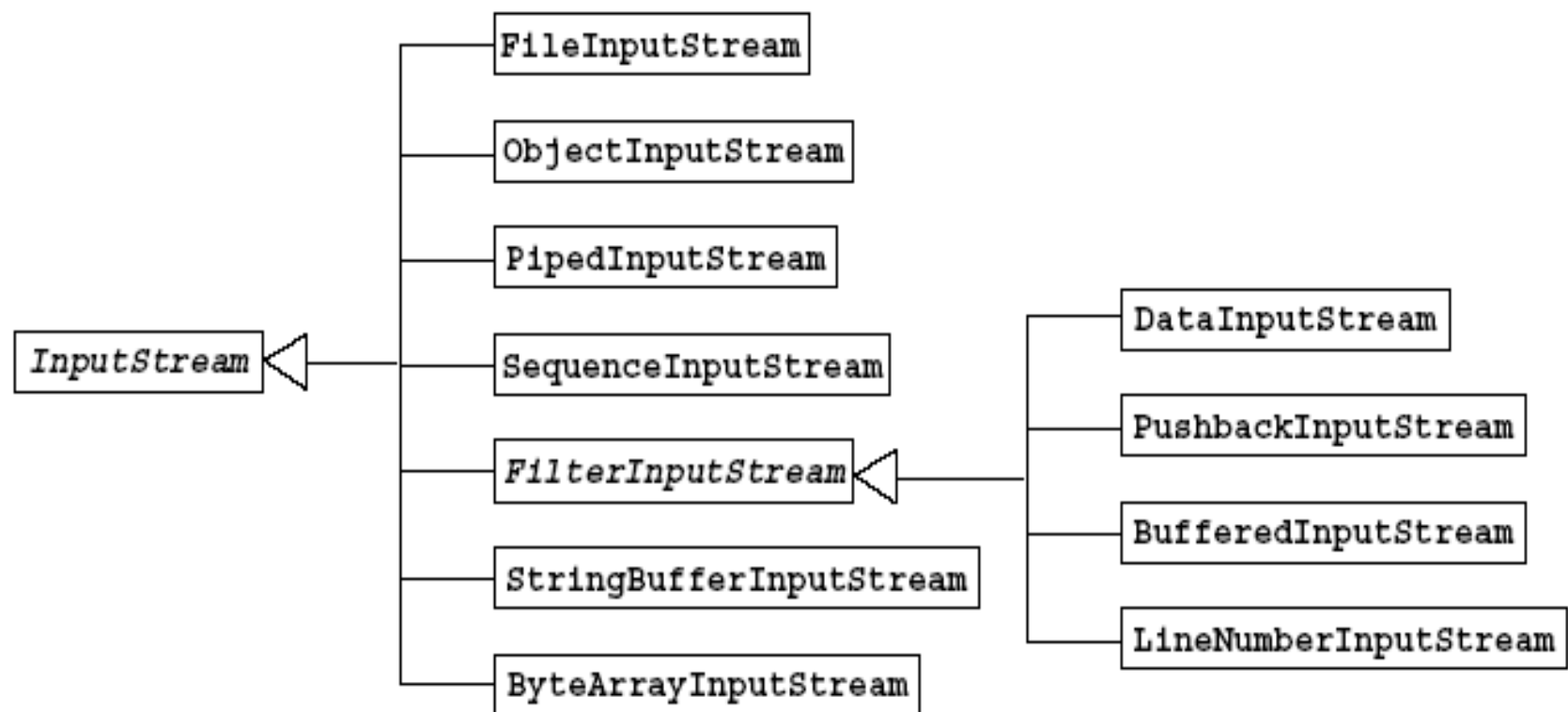




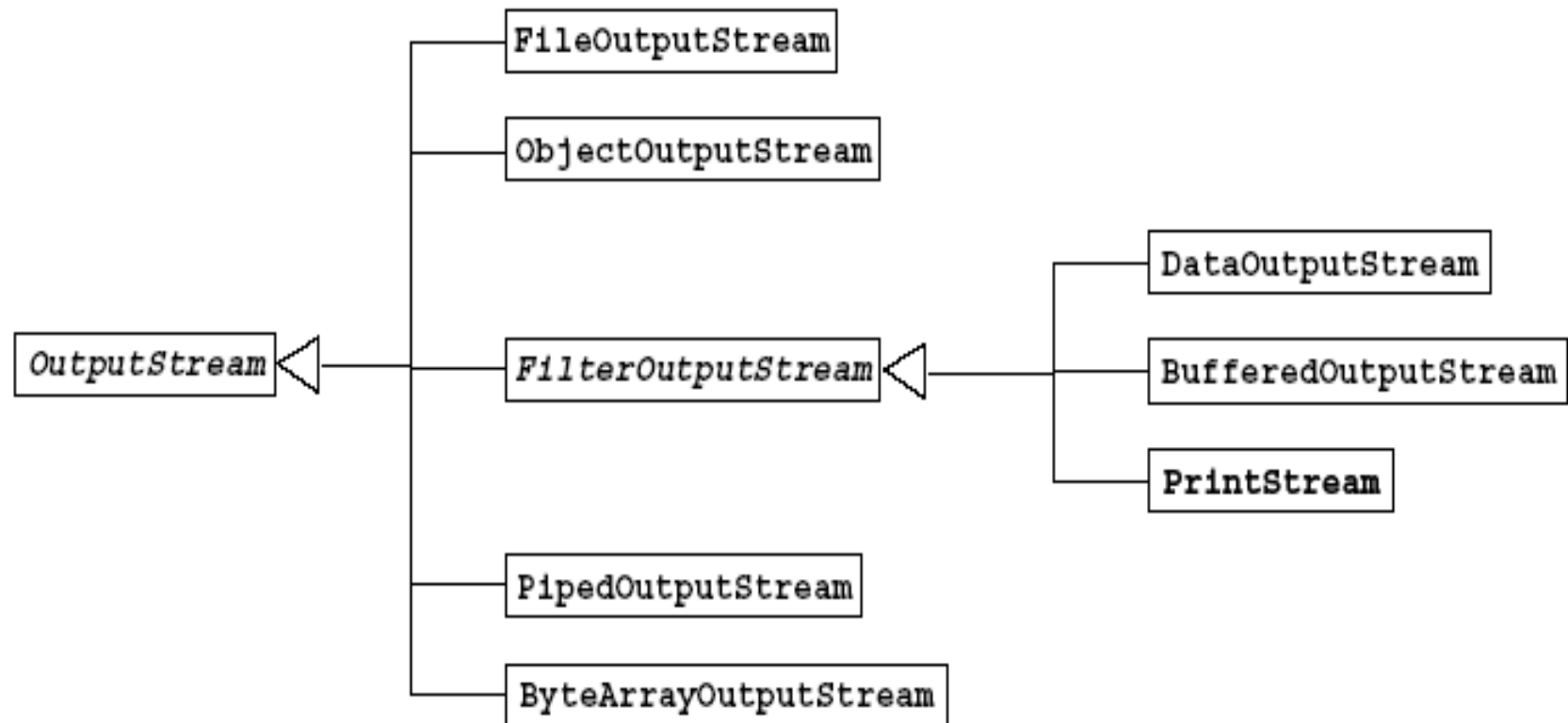
# Writer

- `int write(int c)`
- `int write(char buf[])`
- `int write(char buf[], int offset, int length)`
- `void close()`
- `void flush()`

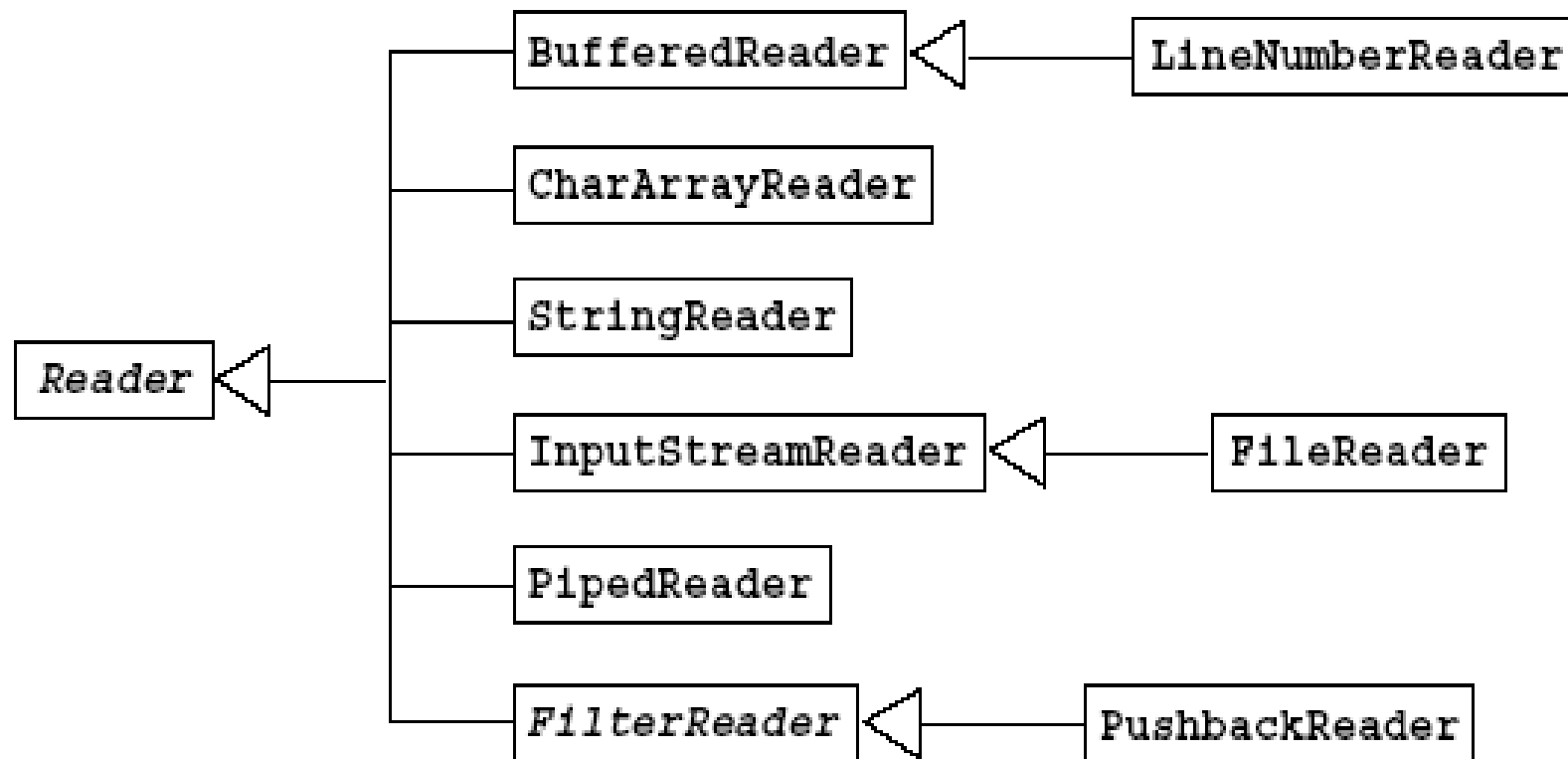
# Phả hệ của InputStream



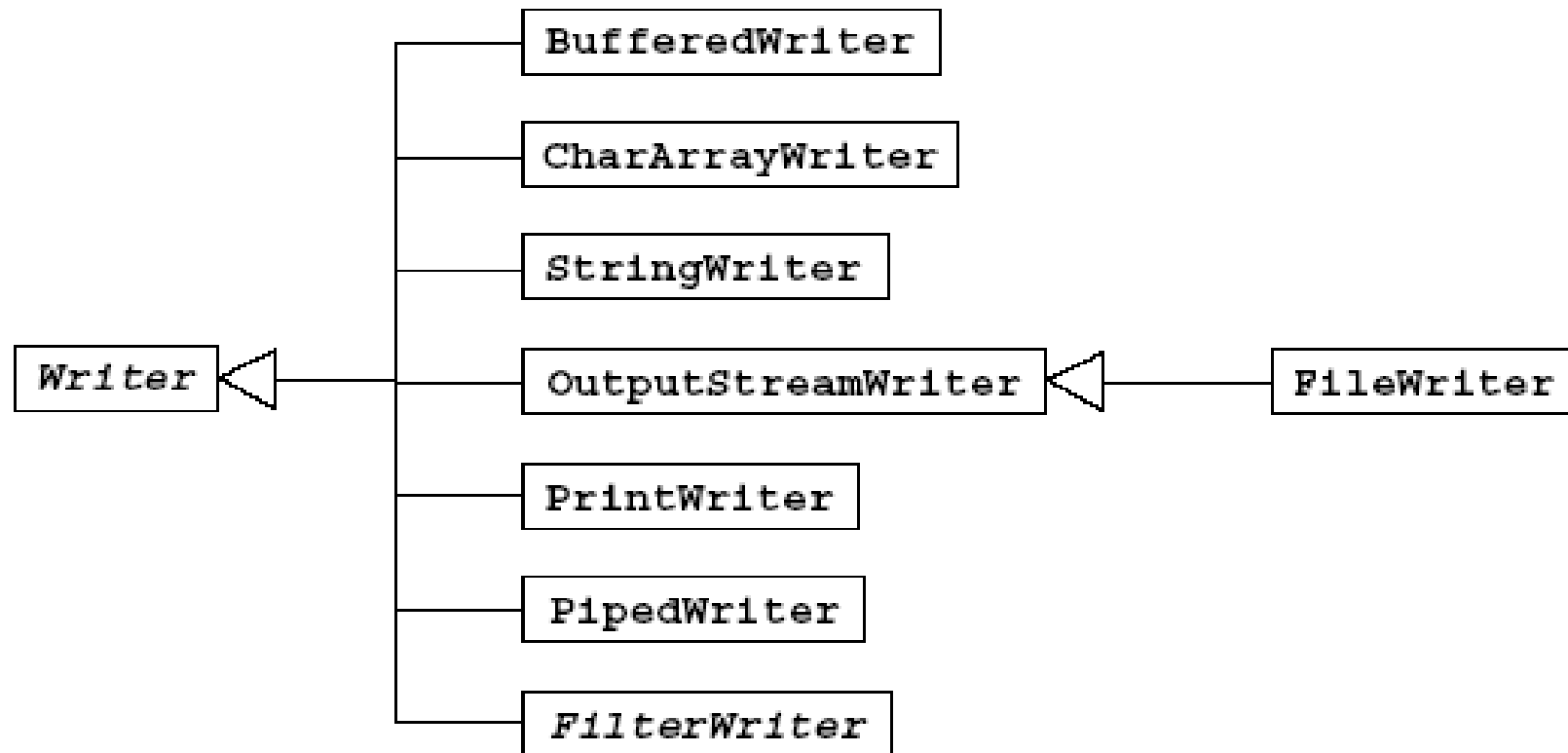
# Phả hệ của OutputStream



# Phả hệ của Reader



# Phả hệ của Writer





# Đối tượng vào / ra

- Để nhập hoặc xuất dữ liệu, chúng ta phải tạo ra đối tượng vào hoặc ra
- Đối tượng vào hoặc ra thuộc kiểu luồng tương ứng và phải được gắn với một nguồn dữ liệu hoặc một đích tiêu thụ dữ liệu



# Sử dụng bộ đệm

- Bộ đệm là một kỹ thuật để tăng tính hiệu quả của thao tác vào / ra
  - đọc và ghi dữ liệu theo khối
  - giảm số lần thao tác với thiết bị
- Thay vì ghi trực tiếp tới thiết bị thì chương trình ghi lên bộ đệm
  - khi bộ đệm đầy thì dữ liệu được ghi ra thiết bị theo khối
  - có thể ghi vào thời điểm bất kỳ bằng phương thức flush()
- Thay vì đọc trực tiếp từ thiết bị thì chương trình đọc từ bộ đệm
  - khi bộ đệm rỗng thì dữ liệu được đọc theo khối từ thiết bị



# Nhập xuất qua thiết bị chuẩn

## Console I/O

- `System.out` cho phép in ra luồng ra chuẩn
  - là đối tượng của lớp `PrintStream`
- `System.err` cho phép in ra luồng thông báo lỗi chuẩn
  - là đối tượng của lớp `PrintStream`
- `System.in` cho phép đọc vào từ thiết bị vào chuẩn
  - là đối tượng của lớp `InputStream`





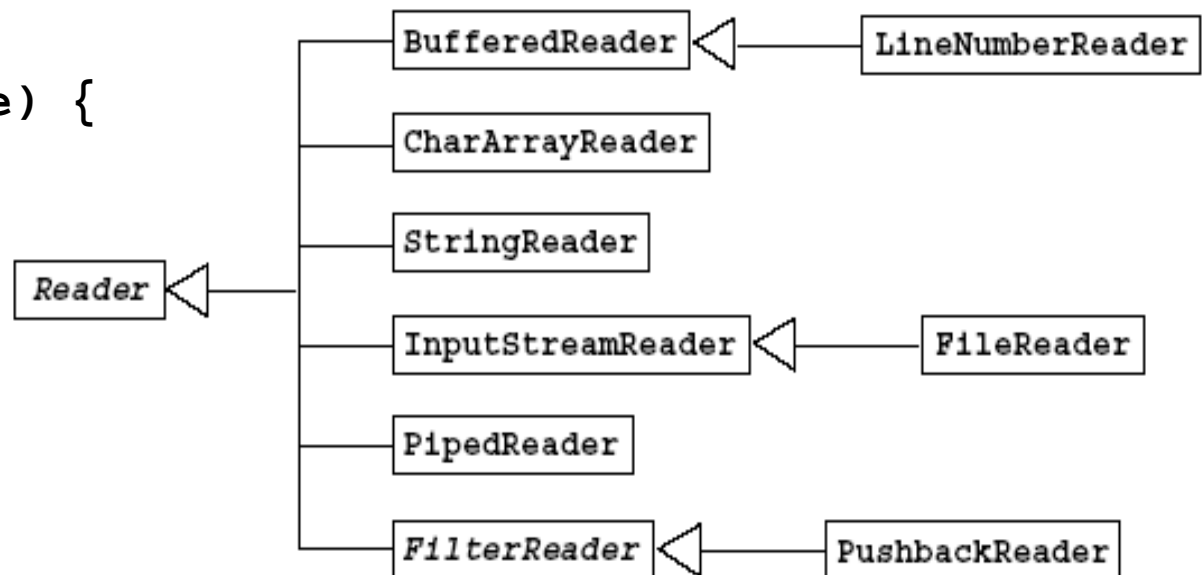
# Đọc dữ liệu từ luồng vào chuẩn

- System.in không sử dụng được trực tiếp
- Chúng ta muốn đọc một dòng ký tự
  1. tạo đối tượng luồng ký tự (InputStreamReader)
  2. tạo đối tượng luồng có bộ đệm (BufferedReader)

# Ví dụ:

```
InputStreamReader reader = new InputStreamReader(System.in);  
BufferedReader in = new BufferedReader(reader);
```

```
String s;  
try {  
    s = in.readLine();  
}  
catch (IOException e) {  
    ...  
}
```





# Lớp File

- Một trong các nguồn và đích dữ liệu thông thường là tệp
- Lớp File cung cấp các chức năng cơ bản để thao tác với tệp
  - Nằm trong gói java.io
  - Tạo tệp, mở tệp, các thông tin về tệp và thư mục
  - Chú ý phân biệt File và Stream



# Tạo đối tượng File

- `File myFile;`
- `myFile = new File("data.txt");`
- `myFile = new File("myDocs",  
"data.txt");`
- Thư mục cũng được coi như là một tệp
  - `File myDir = new File("myDocs");`
  - `File myFile = new File(myDir,  
"data.txt");`
  - có phương thức riêng để thao tác với thư mục



# Các phương thức

## ■ Tên tệp

- ☐ `String getName()`
- ☐ `String getPath()`
- ☐ `String getAbsolutePath()`
- ☐ `String getParent()`
- ☐ `boolean renameTo(File newName)`

## ■ Kiểm tra tệp

- ☐ `boolean exists()`
- ☐ `boolean canWrite()`
- ☐ `boolean canRead()`
- ☐ `boolean isFile()`
- ☐ `boolean isDirectory()`
- ☐ `boolean isAbsolute()`



## Các phương thức (2)

### ■ Nhận thông tin

- ☐ `long lastModified()`

- ☐ `long length()`

- ☐ `boolean delete()`

### ■ Thư mục

- ☐ `boolean mkdir()`

- ☐ `String[] list()`



# Thao tác với tệp ký tự

## ■ Đọc từ tệp

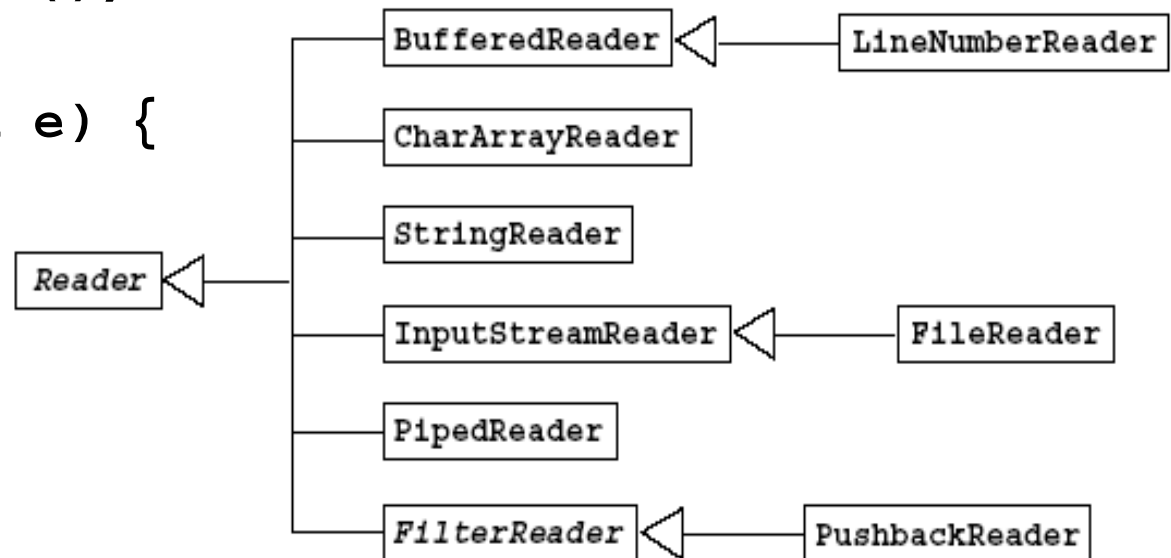
- ☐ FileReader: đọc ký tự từ tệp
- ☐ BufferedReader: đọc có bộ đệm (đọc từng dòng `readLine()`)

## ■ Ghi ra tệp

- ☐ FileWriter: ghi ký tự ra tệp
- ☐ PrintWriter: ghi theo dòng (`print()` và `println()`)

# Ví dụ: Đọc vào từ tệp

```
File file = new File("data.txt");
FileReader reader = new FileReader(file);
BufferedReader in = new BufferedReader(reader);
String s;
try {
    s = in.readLine();
}
catch (IOException e) {
    ...
}
```







## Ví dụ: Đọc vào (cont.)

```
class Abc {  
    public void read(BufferedReader in) {  
        String s;  
        try {  
            s = in.readLine();  
            ...  
        }  
        catch (IOException e) {...}  
    }  
    public void doSomething() {...}  
    ...  
}
```



## Ví dụ: Đọc vào (cont.)

```
File file = new File("data.txt");  
FileReader reader = new FileReader(file);  
BufferedReader in = new BufferedReader(reader);  
Abc abc = new Abc();  
abc.read(in);  
abc.doSomething();
```



# Ví dụ: Ghi ra tệp

```
File file = new File("data.out");
FileWriter writer = new FileWriter(file);
PrintWriter out = new PrintWriter(writer);
String s = "Hello";
try {
    out.println(s);
    out.close();
}
catch (IOException e) {
}
```



## Ví dụ: Ghi ra (cont.)

```
class Abc {  
    ...  
    public void write(PrintWriter out) {  
        ...  
        try {  
            out.println(s);  
            out.close();  
        }  
        catch (IOException e) {...}  
    }  
}
```



## Ví dụ: Ghi ra (cont.)

```
class Abc {  
    ...  
    public String write() {  
        String buf;  
        buf += ...  
        return buf;  
    }  
}
```



# Ví dụ: File copy

```
import java.io.*;

public class CopyFile {
    public static void main(String args[]) {
        try {
            FileReader src = new FileReader(args[0]);
            BufferedReader in = new BufferedReader(src);
            FileWriter des = new FileWriter(args[1]);
            PrintWriter out = new PrintWriter(des);
            String s;

            s = in.readLine();
            while (s != null) {
                out.println(s);
                s = in.readLine();
            }

            in.close();
            out.close();
        }
        catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```



# Ví dụ: File copy (2)

```
import java.io.*;

public class CopyFile2 {
    public static void main(String args[]) {
        try {
            FileReader src = new FileReader(args[0]);
            FileWriter des = new FileWriter(args[1]);
            char buf[] = new char[128];
            int charsRead;
            charsRead = src.read(buf);
            while (charsRead != -1) {
                des.write(buf, 0, charsRead);
                charsRead = src.read(buf);
            }
            src.close();
            des.close();
        }
        catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```



# Thao tác với tệp dữ liệu (tuần tự)

## ■ Đọc dữ liệu

- ☐ FileInputStream: đọc dữ liệu từ tệp
- ☐ DataInputStream: đọc dữ liệu kiểu nguyên thủy
- ☐ ObjectInputStream: đọc đối tượng

## ■ Ghi dữ liệu

- ☐ FileOutputStream: ghi dữ liệu ra tệp
- ☐ DataOutputStream: ghi dữ liệu kiểu nguyên thủy
- ☐ ObjectOutputStream: ghi đối tượng





# DataInputStream/DataOutputStream

- DataInputStream: đọc các dữ liệu nguyên thủy
  - readBoolean, readByte, readChar, readShort, readInt, readLong, readFloat, readDouble
- DataOutputStream: ghi các dữ liệu nguyên thủy
  - writeBoolean, writeByte, writeChar, writeShort, writeInt, writeLong, writeFloat, writeDouble



# Ghi dữ liệu nguyên thủy (tuần tự)

```
import java.io.*;

public class TestDataOutputStream {
    public static void main(String args[]) {
        int a[] = {2, 3, 5, 7, 11};

        try {
            FileOutputStream fout = new FileOutputStream(args[0]);
            DataOutputStream dout = new DataOutputStream(fout);

            for (int i=0; i<a.length; i++)
                dout.writeInt(a[i]);
            dout.close();
        }
        catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```



# Đọc dữ liệu nguyên thủy (tuần tự)

```
import java.io.*;

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

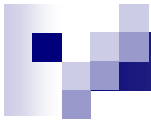
        try {
            FileInputStream fin = new FileInputStream(args[0]);
            DataInputStream din = new DataInputStream(fin);

            while (true) {
                System.out.println(din.readInt());
            }
        }
        catch (EOFException e) {
        }
        catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```



# Đọc ghi đối tượng

- Một đối tượng có thể được lưu trong bộ nhớ tại nhiều vùng nhớ khác nhau
  - các thuộc tính không phải là kiểu nguyên thủy
- Đối tượng muốn ghi / đọc (*chuỗi hóa*) được phải thuộc lớp có cài đặt giao diện Serializable
  - đây là giao diện *nhãn*, không có phương thức
  - Các thuộc tính không nguyên thủy cũng phải *chuỗi hóa* được



```
import java.io.Serializable;

class Record implements Serializable {
    private String name;
    private float score;


    public Record(String s, float sc) {
        name = s;
        score = sc;
    }

    public String toString() {
        return "Name: " + name + ", score: " + score;
    }
}
```



```
import java.io.*;
```

```
public class TestObjectOutputStream {  
    public static void main(String args[]) {  
        Record r[] = { new Record("john", 5.0F),  
                        new Record("mary", 5.5F),  
                        new Record("bob", 4.5F) };  
  
        try {  
            FileOutputStream fout = new FileOutputStream(args[0]);  
            ObjectOutputStream out = new ObjectOutputStream(fout);  
  
            for (int i=0; i<r.length; i++)  
                out.writeObject(r[i]);  
  
            out.close();  
        }  
        catch (IOException e) {  
            e.printStackTrace();  
        }  
    }  
}
```



```
import java.io.*;

public class TestObjectInputStream {
    public static void main(String args[]) {
        Record r;

        try {
            FileInputStream fin = new FileInputStream(args[0]);
            ObjectInputStream in = new ObjectInputStream(fin);

            while (true) {
                r = (Record) in.readObject();
                System.out.println(r);
            }
        } catch (EOFException e) {
            System.out.println("No more records");
        } catch (ClassNotFoundException e) {
            System.out.println("Unable to create object");
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```



# Lớp RandomAccessFile

- Là một lớp độc lập (kế thừa trực tiếp từ Object)
- Đảm nhận việc đọc và ghi dữ liệu ngẫu nhiên
  - cài đặt các giao diện DataInput và DataOutput
- Kích thước bản ghi phải cố định





```
import java.io.*;

public class WriteRandomFile {
    public static void main(String args[]) {
        int a[] = { 2, 3, 5, 7, 11, 13 };

        try {
            File fout = new File(args[0]);
            RandomAccessFile out;
            out = new RandomAccessFile(fout, "rw");

            for (int i=0; i<a.length; i++)
                out.writeInt(a[i]);
            out.close();
        }
        catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```



```
import java.io.*;

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


        try {
            File fin = new File(args[0]);
            RandomAccessFile in = new RandomAccessFile(fin, "r");

            int recordNum = (int) (in.length() / 4);
            for (int i=recordNum-1; i>=0; i--) {
                in.seek(i*4);
                System.out.println(in.readInt());
            }
        }
        catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```



# Lớp Scanner

- Là lớp mới hỗ trợ nhập dữ liệu, kế thừa trực tiếp từ `Object` (từ Java 1.5)
- Khởi tạo với đối số là đối tượng vào (luồng, tệp, chuỗi ký tự)
- Có các phương thức hỗ trợ nhập trực tiếp
  - `nextType`, `hasNextType`



```
Scanner sc = new Scanner(System.in);  
int i = sc.nextInt();
```

---

```
Scanner sc;  
sc = new Scanner(new File("myNumbers"));  
while (sc.hasNextLong()) {  
    System.out.println(sc.nextLong());  
}
```

--

```
String str = "2 3 5 7";  
Scanner sc = new Scanner(str);  
while (sc.hasNextInt()) {  
    System.out.println(sc.nextInt());  
}
```



# Trừu tượng hóa nguồn nhập/xuất

- Không nhất thiết phải chỉ rõ nguồn nhập/xuất
  - Tăng tính tái sử dụng

```
class Abc {  
    public void read(Scanner sc) {  
        ...  
    }  
    public void doSomething() {...}  
    ...  
}
```



# Tổng kết

- Cần nắm rõ khái niệm
  - ☐ luồng (vào/ra), luồng ký tự/nhị phân
  - ☐ đối tượng vào/ra
  - ☐ Nguồn (tệp, bàn phím,...), đích (...)
  - ☐ Chuỗi hóa đối tượng
- Vào/ra gắn với xử lý ngoại lệ
- Cần độc lập việc vào/ra và xử lý của chương trình



# Bài tập

- Tự thực hành tất cả các ví dụ trong slide và trong Giáo trình
- Viết các lớp vào/ra cho bài tập 2 (đọc/ghi tệp)