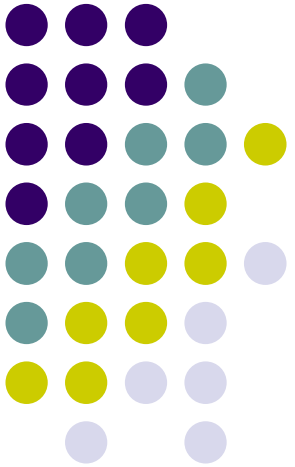
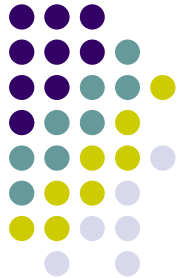
Java IO Stream

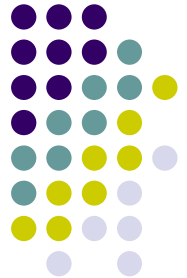


Nội dung



- Luồng nhập xuất là gì?
- Các loại luồng
- Phân cấp lớp luồng
- Dùng Stream để điều khiển luồng nhập xuất.
- Byte streams
- Character streams
- Buffered streams
- Standard I/O streams
- Data streams
- Object streams
- Lớp File

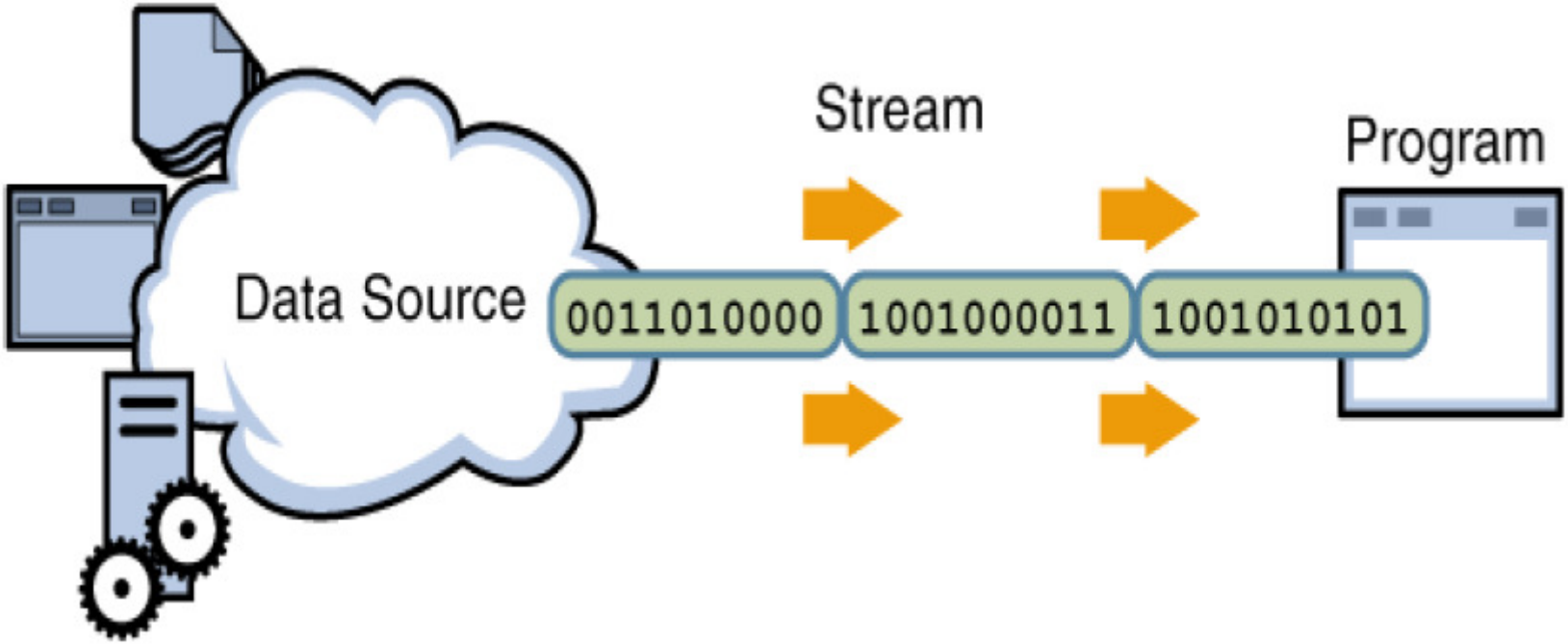
Khái niệm luồng?



- Luồng là một “dòng chảy” của dữ liệu được gắn với các thiết bị vào ra.
- Hai loại luồng:
  - Luồng nhập: Gắn với các thiết bị nhập như bàn phím, máy scan, file...
  - Luồng xuất: Gắn với các thiết bị xuất như màn hình, máy in, file...
- Việc xử lý vào ra thông qua luồng giúp cho lập trình viên không phải quan tâm đến bản chất của thiết bị vào ra.

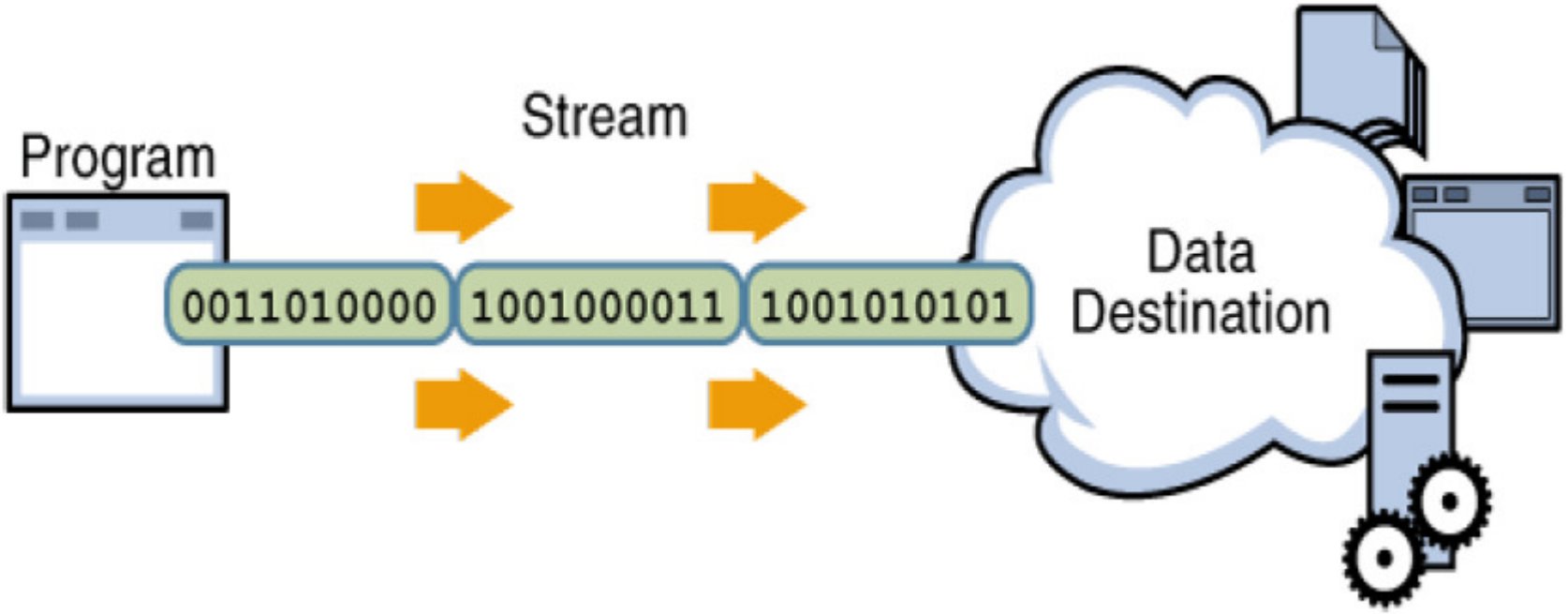
Input Stream

- Chương trình sử dụng input stream để đọc dữ liệu từ nguồn.

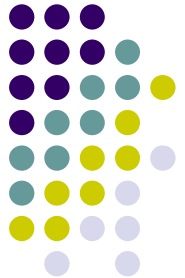


Output Stream

- Chương trình sử dụng output stream để ghi dữ liệu xuống đích.

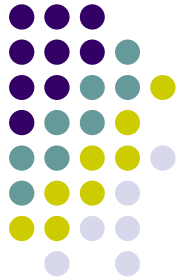


Các loại luồng



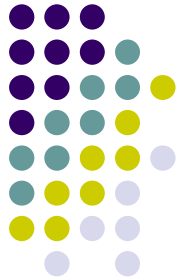
- Character and Byte Streams
  - Character vs. Byte
- Input and Output Streams
  - Dựa trên nguồn và đích
- Node and Filter Streams
  - Khi dữ liệu trong luồng được thao tác hoặc chuyển đổi.

Character and Byte Streams



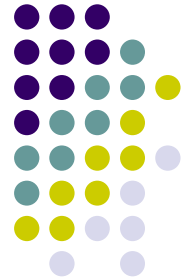
- Byte streams
  - Cho dữ liệu dạng nhị phân
  - Những lớp gốc cho các byte stream:
    - Lớp *InputStream*
    - Lớp *OutputStream*
    - Cả 2 lớp là trừu tượng (Abstract)
- Character streams
  - Cho các ký tự Unicode
  - Những lớp gốc cho character stream:
    - Lớp *Reader*
    - Lớp *Writer*
    - Cả 2 lớp là trừu tượng (Abstract)

Input and Output Streams

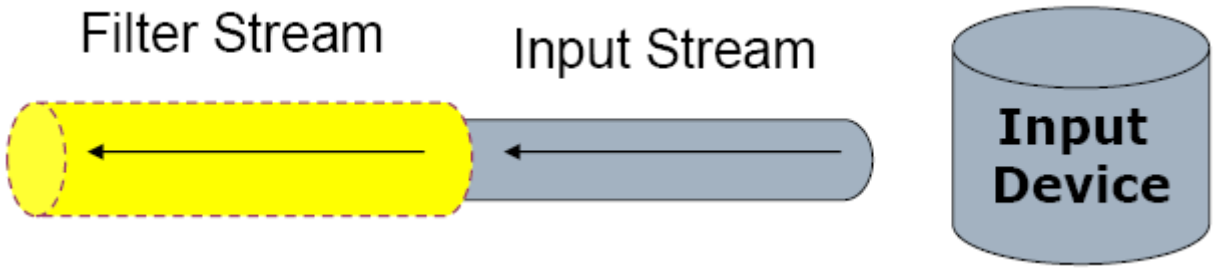


- Input or source streams
  - Có thể đọc từ những nguồn này.
  - Những lớp gốc của tất cả các input stream:
    - Lớp *InputStream*
    - Lớp *Reader*
- Output or destination streams
  - Có thể ghi xuống những luồng này
  - Những lớp gốc của tất cả các output stream:
    - Lớp *OutputStream*
    - Lớp *Writer*

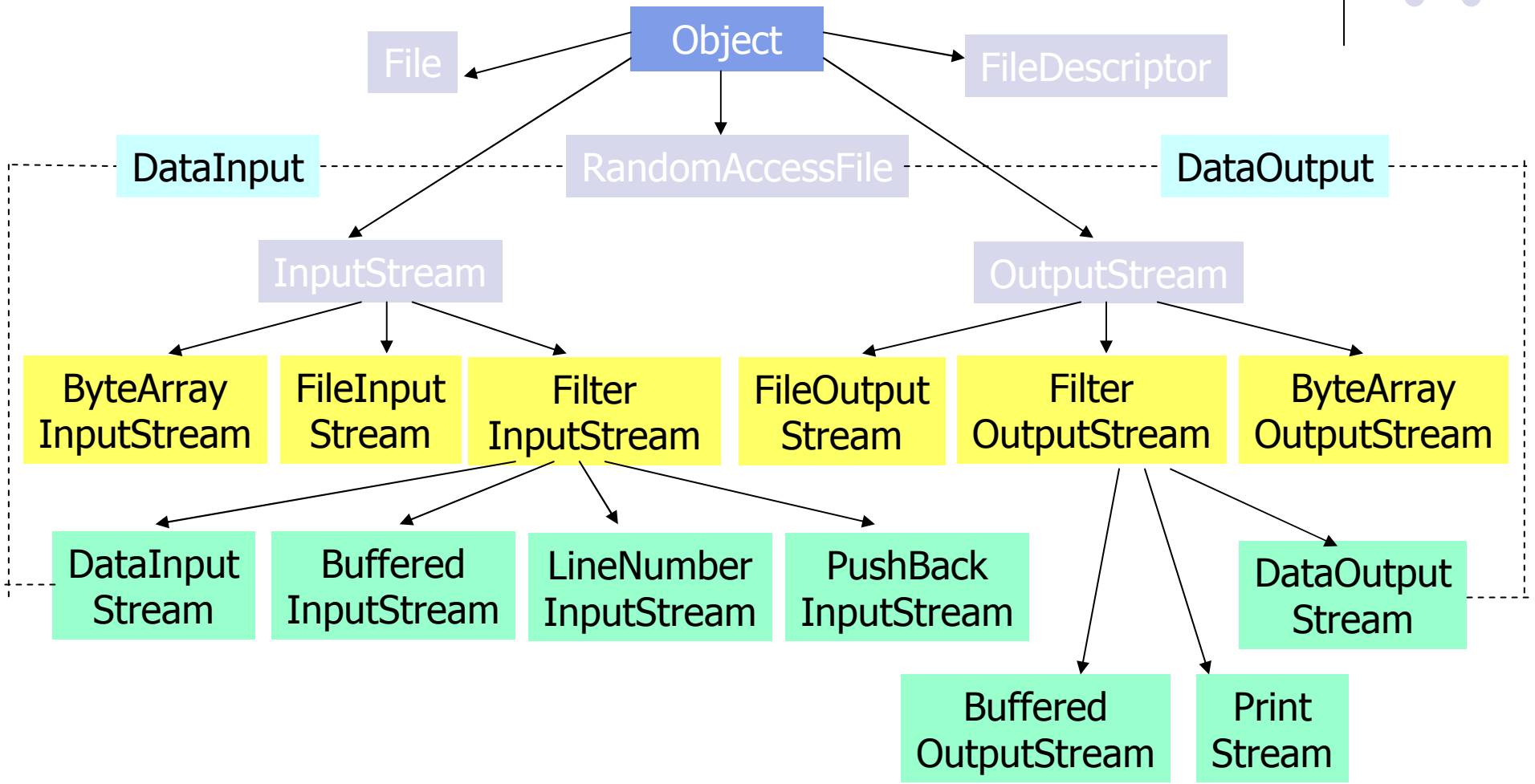
Node and Filter Streams



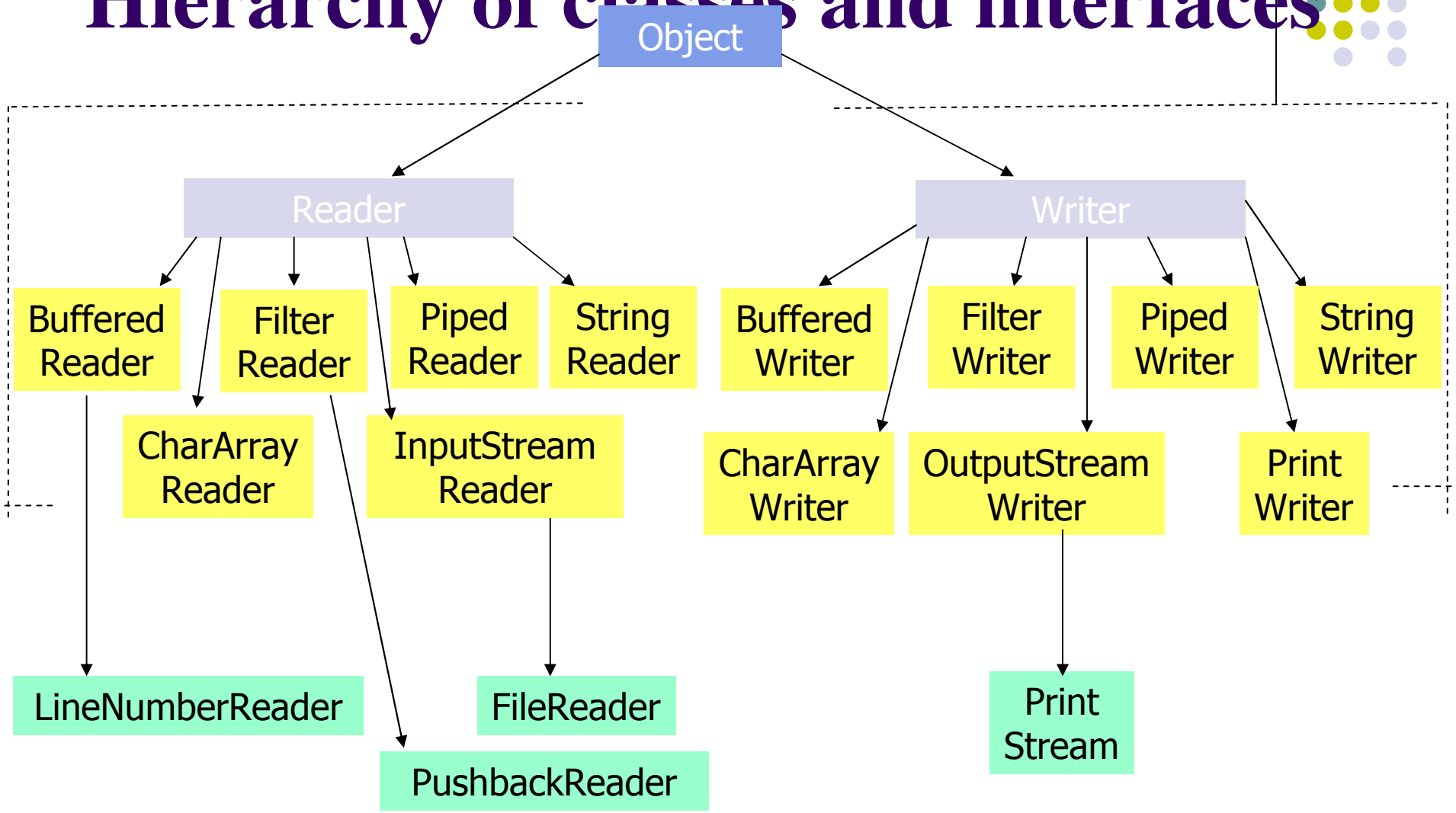
- Node streams (Data sink stream)
  - Chứa những chức năng cơ bản cho việc đọc và ghi từ một vị trí xác định.
  - Các loại node stream bao gồm file, bộ nhớ và pipe
- Filter streams (Processing stream)
  - Luồng lọc có khả năng kết nối với các luồng khác và xử lý dữ liệu "theo cách riêng" của nó.
  - FilterInputStream và FilterOutputStream là 2 lớp luồng lọc cơ bản.



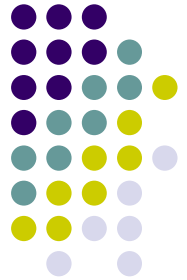
Hierarchy of classes and interfaces



Hierarchy of classes and interfaces



Abstract Classes



- InputStream & OutputStream
- Reader & Writer

InputStream Abstract Class

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

  Đọc một byte kế tiếp của dữ liệu từ luồng.

- `public int` **`read`**`(byte[] bBuf) throws IOException`

  Đọc một số byte dữ liệu từ luồng và lưu vào mảng byte bBuf.

- `public int` **`read`**`(byte[] cBuf, int offset, int length) throws IOException`

  Đọc `length` byte dữ liệu từ luồng và lưu vào mảng byte cBuf bắt đầu tại vị trí offset.

- `public void` **`close`**`() throws IOException`

  Đóng nguồn. Gọi những phương thức khác sau khi đó nguồn sẽ gây ra lỗi IOException

- `public int` **`mark`**`(int readAheadLimit) throws IOException`

  Đánh dấu vị trí hiện hành của stream. Sau khi đánh dấu, gọi reset() sẽ định lại vị trí của luồng đến điểm này. Không phải tất cả luồng byte –input hỗ trợ cho thao tác này.

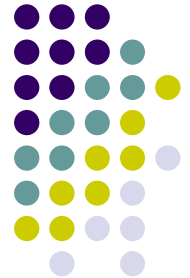
- `public int` **`markSupported`**`()`

  Chỉ ra luồng có hỗ trợ thao tác mark và reset hay không

Node *InputStream* Classes

- `FileInputStream`
  - Đọc các byte từ file
- `ByteArrayInputStream`
  - Thực thi một buffer mà nó chứa các byte, mà nó có thể được đọc từ nguồn.
- `PipedInputStream`
  - Nên được liên kết với `PipedOutputStream`. Những luồng này được sử dụng bởi 2 luồng. Trong đó một cái là đọc dữ liệu từ nguồn trong khi những cái khác thì ghi xuống `PipedOutputStream` tương ứng.

Filter *InputStream* Classes



- `BufferedInputStream`
  - Một class con của FilterInputStream cho phép đặt vùng đệm cho input để đọc các byte dữ liệu một cách hiệu quả.
- `FilterInputStream`
  - For reading filtered byte streams, which may transform the basic source data along the way and provide additional functionalities.
- `ObjectInputStream`
  - Used for object serialization. Deserializes objects and previuosly written using an ObjectOutputStream.
- `DataInputStream`
  - A subclass of FilterInputStream that lets an application read Java primitive data from underlying inputstream in a machine-independent way.
- `LineNumberInputStream`
  - A subclass of FilterInputStream that allows tracking of the current line number.
- `PushbackInputStream`
  - A subclass of FilterInputStream class that allows bytes to be pushed back or unread into the stream

Lớp trừu tượng *OutputStream*

- `public void` **`write`**`(int b) throws IOException`
  Ghi giá trị b xác định theo dạng byte xuống output stream

- `public void` **`write`**`(byte[] b) throws IOException`
  Lưu nội dung của mảng byte b xuống luồng

- `public void` **`write`**`(byte[] b, int off, int len) throws IOException`
  Lưu len byte của mảng byte b xuống luồng, bắt đầu từ vị trí off của mảng

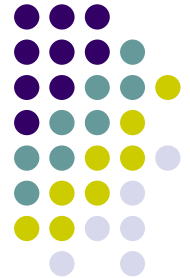
- `public void` **`close`**`() throws IOException`
  Đóng nguồn. Gọi những phương thức khác liên quan đến nguồn này sau khi gọi close se gây ra lỗi IOException.

- `public void` **`flush`**`() throws IOException`
  flushes the stream.(ví dụ: Những byte được lưu trong buffer ngay lập tức được ghi xuống đích)

Node *OutputStream* Classes

- `FileOuputStream`
  - For writing bytes to a file
- `ByteArrayOutputStream`
  - Implements a buffer that contains bytes, which may be written to the stream
- `PipedOutputStream`
  - Should be connected to a PipedInputStream. These streams are typically used by two threads wherein one of these threads writes data to this stream while the other thread reads from the corresponding PipeInputStream.

Filter *OutputStream* Classes

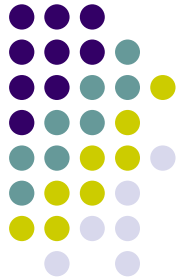


- `BufferedOutputStream`
  - A subclass of FilterOutputStream that allows buffering of output in order to provide for the efficient writing of bytes. Allows writing of bytes to the underlying output stream without necessarily causing a call to underlying system for each byte written.
- `FilterOutputStream`
  - For writing filtered byte streams, which may transform the basic source of data along the way and provide additional functionalities.
- `ObjectOutputStream`
  - Used for object serialization. Serializes object and primitve data to OuputStream.
- `DataOutputStream`
  - A subclass of FilterOutputStream that lets an application write Java primitive data to underlying output stream in machine-independent way.
- `PrintStream`
  - A subclass of FilteOutputStream that provides capability for printing representations of various data values conveniently.

The *Reader* Class: Methods

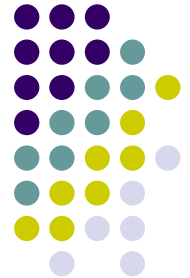
- `public int` **`read`**`() throws IOException`
  - Đọc một ký tự
- `public int` **`read`**`(char[] cbuf) throws IOException`
  - Đọc những ký tự và lưu chúng vào mảng cbuf
- `public abstract int` **`read`**`(char[] cbuf, int off, int len) throws IOException`
  - Đọc len ký tự và lưu chúng vào tron mảng cbuf, bắt đầu tại vị trí off của mảng
- `public abstract void` **`close`**`() throws IOException`
  - Đóng luồng. Gọi những phương thức Reader khác của sau khi gọi close sẽ gây ra lỗi IOException
- `public void` **`mark`**`(int readAheadLimit) throws IOException`
  - Đánh dấu vị trí hiện hành của stream. Sau khi đánh dấu, gọi reset() để thử đặt lại vị trí luồng tới điểm này. Không phải tất cả character-input đều hỗ trợ thao tác này
- `public boolean` **`markSupported`**`()`
  - Chỉ ra luồng có hỗ trợ thao tác này hay không. Mặc định là không hỗ trợ.
- `public void` **`reset`**`() throws IOException`
  - Đặt lại vị trí luồng tới vị trí đánh dấu lần cuối

Node *Reader* Classes



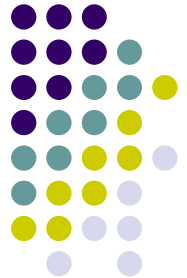
- `FileReader`
  - Cho việc đọc từ file
- `CharArrayReader`
  - Thực thi một vùng đệm ký tự có thể được đọc
- `StringReader`
  - Cho việc đọc từ nguồn string
- `PipedReader`
  - Dùng theo cặp (tương ứng với PipedWriter) bằng 2 luồng mà chúng có thể liên lạc với nhau.Một trong những cái luồng đọc các ký tự.

Filter *Reader* Classes



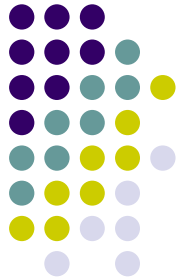
- `BufferedReader`
  - Allows buffering of characters in order to provide for efficient reading of characters, arrays, and lines
- `FilterReader`
  - For reading filtered character streams
- `InputStreamReader`
  - Converts read bytes to characters
- `LineNumberReader`
  - A subclass of BufferedReader class that able to keep track of line numbers.
- `PushbackReader`
  - A subclass of the FilterReader class that allows character to pushed back or unread into the stream.

The *Writer* Class: Methods



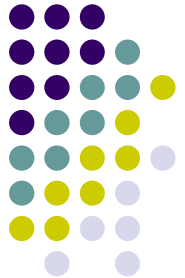
- `public void` **`write`**`(int c) throws IOException`
  Ghi một ký tự đơn được thể hiện bằng số nguyên. Ví dụ: 'A' là được ghi la write(65)
- `public void` **`write`**`(char[] cbuf) throws IOException`
  Ghi nội dung của mảng ký tự cbuf xuống luồng
- `public abstract void` **`write`**`(char[] cbuf, int off, int len) throws IOException`
  Ghi một mảng ký tự cbuf với chiều dài là len, bắt đầu là vị trí off
- `public void` **`write`**`(String str) throws IOException`
  Ghi một chuỗi str
- `public void` **`write`**`(String str, int off, int len) throws IOException`
  Ghi một chuỗi str với chiều dài là len, bắt đầu từ vị trí off
- `public abstract void` **`flush`**`() throws IOException`
  Đẩy dữ liệu xuống đích đến.
- `public abstract void` **`close`**`() throws IOException`
  Đóng luồng.

Node *Writer* Classes



- `FileWriter`
  - For writing to character to file
- `CharArrayWriter`
  - Implements a character buffer that can be written to
- `StringWriter`
  - For writing to a string source
- `PipedWriter`
  - Used in pairs(with corresponding with PipedReader) by two threads that want to communicate. One of these threads writes characters to this stream.

Filter *Writer* Classes

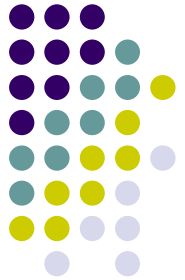


- `BufferedWriter`
  - Allows buffering of characters in order to provide for the efficient writing of characters, arrays, and lines.
  -
- `FilterWriter`
  - For writing filtered character streams.
- `OutputStreamWriter`
  - Encodes characters written to it into bytes.
- `PrintWriter`
  - Prints formatted representations of objects to a text-output stream.

Điều khiển luồng nhập xuất(I/O)

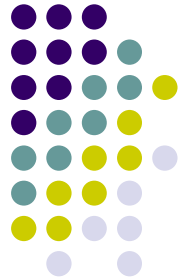
- Tạo đối tượng luồng và liên kết nó với dữ liệu nguồn(data-destination)
- Đưa ra đối tượng luồng với chức năng mong muốn thông qua chuỗi luồng (Give the stream object the desired functionality through stream chaining)
- Đóng luồng

Byte Stream



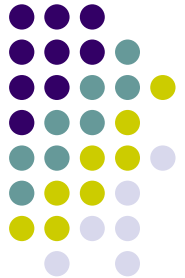
- Chương trình sử dụng luồng byte để thực hiện nhập xuất những byte 8-bit
- Tất cả cá lớp luồng byte được kế thừa từ *`InputStream`* và *`OutputStream`*
- Có nhiều lớp luồng byte
  - `FileInputStream` và `FileOutputStream`
- Chúng được sử dụng trong cùng một cách; chúng khác nhau chủ yếu là cách thức chúng được khởi tạo.

Khi nào không sử dụng Byte Stream



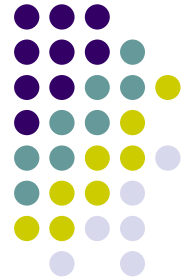
- Byte Stream thể hiện một loại dữ liệu nhập xuất mức thấp do đó chúng ta nên tránh:
  - Nếu dữ liệu chứa những ký tự, phương pháp tốt nhất là sử dụng character streams
  - Có những stream cho những kiểu dữ liệu phức tạp
- Byte Stream chỉ nên sử dụng cho hầu hết những nhập xuất nguyên thủy
- Tất cả các stream khác đều được dựa trên byte stream

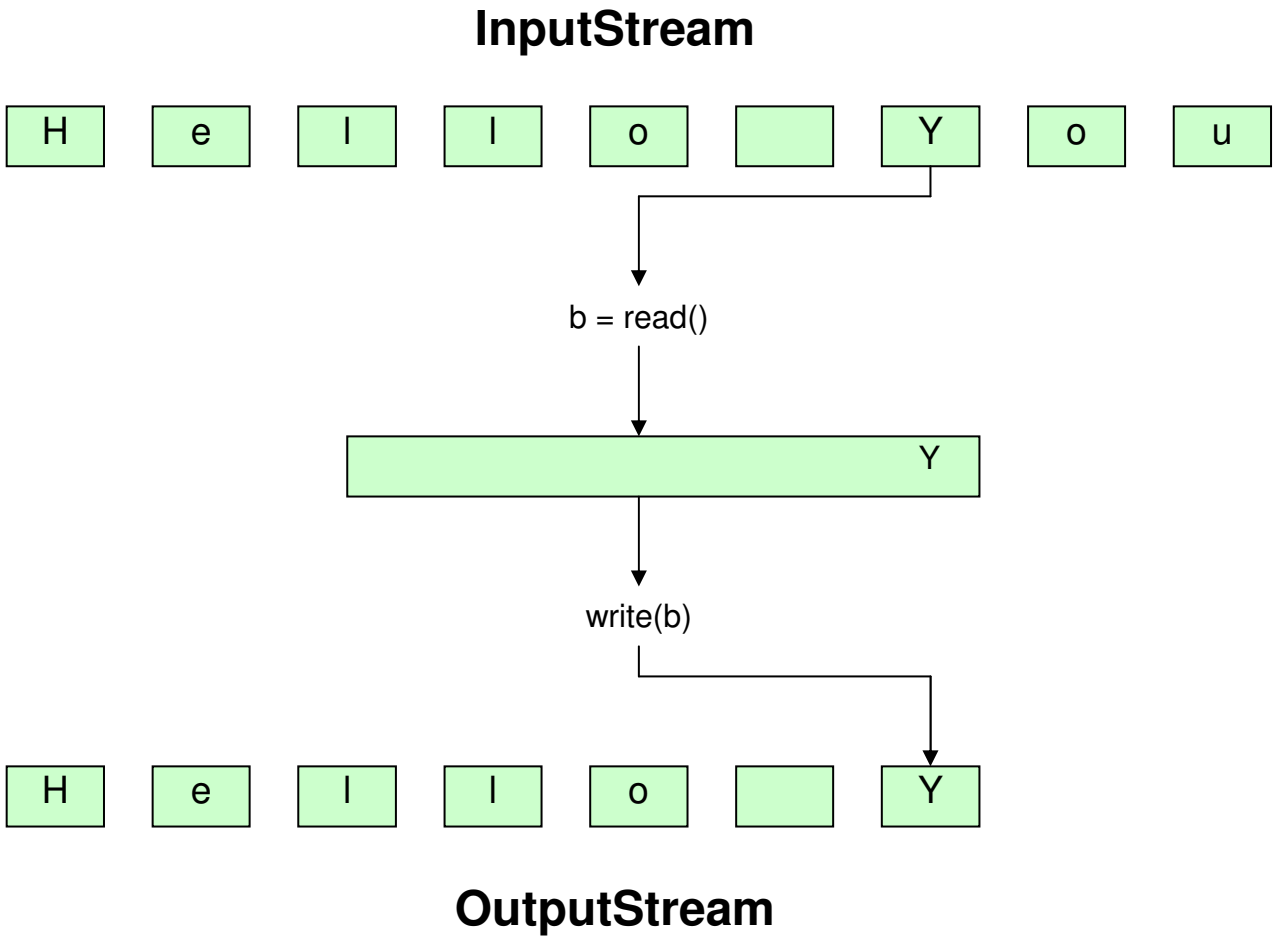
Example: *FileInputStream & FileOutputStream*



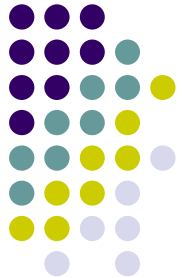
```
import java.io.*;
public class CopyBytes {
    public static void main(String[] args) throws IOException {
        FileInputStream in = null;
        FileOutputStream out = null;
        try {
            in = new FileInputStream(“d:/src.txt");
            out = new FileOutputStream(“d:/dst.txt");
            int c;
            while ((c = in.read()) != -1) { out.write(c); }
        }
        finally {
            if (in != null) { in.close(); }
            if (out != null) { out.close(); }
        }
    }
}
```

Simple Byte Stream input and output



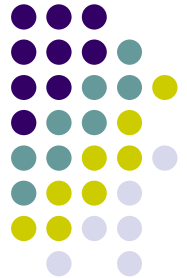


Character Stream



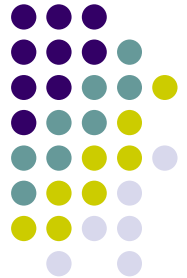
- Java platform lưu trữ những giá trị ký tự theo dạng Unicode
- Tất cả các lớp character stream được kế thừa từ Reader và Writer
- Có các lớp character stream : FileReader và FileWriter.

Example: FileReader & FileWriter



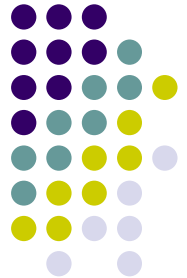
```
public class CopyCharacters {
    public static void main(String[] args) throws IOException {
        FileReader inputStream = null;
        FileWriter outputStream = null;
        try {
            inputStream = new FileReader(“d:/fileIn.txt");
            outputStream = new FileWriter(“d:/fileOut.txt");
            int c;
            while ((c = inputStream.read()) != -1) { outputStream.write(c);}
        }
        finally {
            if (inputStream != null) { inputStream.close(); }
            if (outputStream != null) { outputStream.close(); }
        }
    }
}
```

Character Stream và Byte Stream



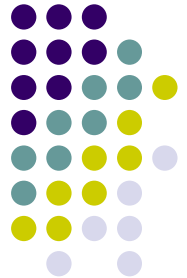
- Character stream thường là "wrappers“ cho byte stream
- Character stream sử dụng byte stream thực hiện nhập xuất vật lý, trong khi character stream xử lý chuyển đổi giữa character và byte
  - Chẳng hạn FileReader dùng như FileInputStream, trong khi FileWriter dùng như FileOutputStream

Buffered Streams



- Một I/O không có bộ đệm có nghĩa là mỗi yêu cầu đọc hoặc ghi được xử lý trực tiếp bởi OS
  - Điều này làm chương trình kém hiệu quả, vì mỗi yêu cầu thường phải truy xuất đĩa, hoạt động mạng, hoặc một và thao tác khá mà nó tương đối tốn thời gian.
- Để giảm công việc loại này, Java platform thực thi luồng I/O có bộ đệm
  - Buffered input stream đọc dữ liệu từ vùng nhớ như là bộ đệm; API input được gọi chỉ khi bộ đệm rỗng.
  - Tương tự, buffered output stream ghi dữ liệu xuống bộ đệm, và API output chỉ được gọi khi buffer đầy.

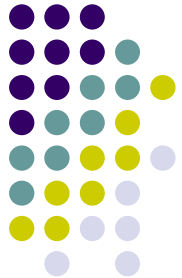
Tạo Buffered Stream?



- Một chương trình có thể chuyển một stream không có bộ đệm thành stream có bộ đệm bằng cách sử dụng vỏ bao.
  - Một unbuffered stream object được chuyển qua hàm dựng cho một lớp stream có bộ đệm
- Ví dụ:

```
inputStream = new BufferedReader(new
   FileReader("characterinput.txt"));
outputStream = new BufferedWriter(new
   FileWriter("characteroutput.txt"));
```

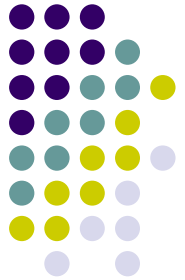
Buffered Stream Classes



- *BufferedInputStream* và *BufferedOutputStream* tạo ra byte stream có bộ đệm

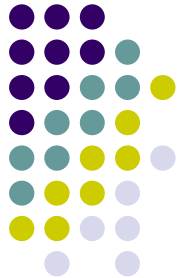
- *BufferedReader* và *BufferedWriter* tạo ra character stream có bộ đệm

Standard Streams on Java Platform



- Có 3 standard stream
  - Input: System.in
  - Output: System.out
  - Error: System.err
- Những đối tượng này được định nghĩa tự động và không cần thiết được mở.
- System.out and System.err được định nghĩa như là những PrintStream object

Data Streams



- Data stream hỗ trợ nhập xuất nhị phân của các loại dữ liệu nguyên thuỷ (boolean, char, byte, short, int, long, float, and double) cũng như là String

- Tất cả data stream thực thi cả `DataInput` *cũng như* `DataOutput` *interface*

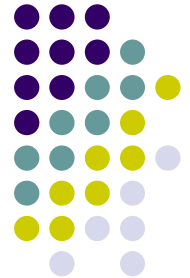
- `DataInputStream` and `DataOutputStream` thực thi những phương thức của những interface này

DataOutputStream

- *DataOutputStream* có thể chỉ được tạo như là mộ vỏ bọc cho đối tượng byte stream

```
out = new DataOutputStream(
    new BufferedOutputStream(
    new FileOutputStream(dataFile)));
    for (int i = 0; i < prices.length; i ++) {
         out.writeDouble(prices[i]);
         out.writeInt(units[i]);
         out.writeUTF(descs[i]);
    }
```

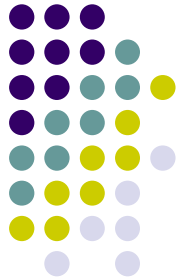
DataInputStream



- Giống như *DataOutputStream*, *DataInputStream* phải được xây dựng như là một vỏ bọc cho đối tượng byte stream
- Điều kiện End-of-file được dò bởi biệt lệ *EOFException*, thay vì kiểm tra để trả về giá trị hết file.

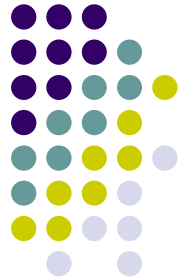
```
in = new DataInputStream(new BufferedInputStream(
                    new FileInputStream(dataFile)));
      try{
            double price = in.readDouble();
            int unit = in.readInt();
            String desc = in.readUTF();
      } catch (EOFException e){
      }
```

Object Streams



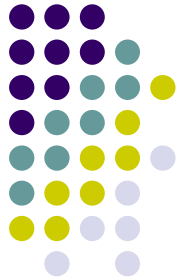
- Object streams support I/O of objects
  - Like Data streams support I/O of primitive data types
  - The object has to be *Serializable* type
- The object stream classes are ObjectInputStream and ObjectOutputStream
  - These classes implement ObjectInput and ObjectOutput, which are subinterfaces of DataInput and DataOutput
  - An object stream can contain a mixture of primitive and object values

Input and Output of Complex Object



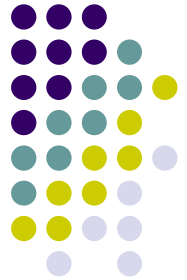
- The writeObject and readObject methods are simple to use, but they contain some very sophisticated object management logic
  - This isn't important for a class like Calendar, which just encapsulates primitive values. But many objects contain references to other objects.
- If readObject is to reconstitute an object from a stream, it has to be able to reconstitute all of the objects the original object referred to.
  - These additional objects might have their own references, and so on.

WriteObject

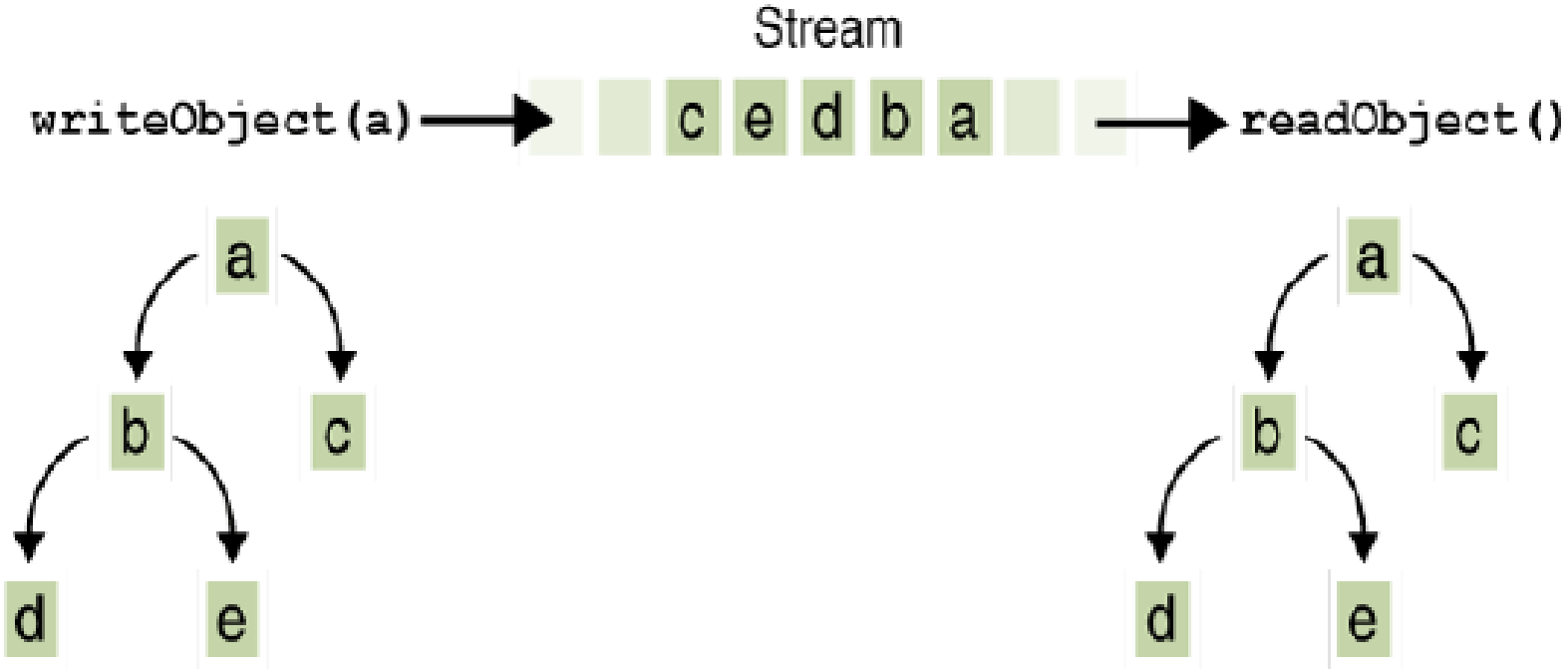


- The writeObject traverses the entire web of object references and writes all objects in that web onto the stream
- A single invocation of writeObject can cause a large number of objects to be written to the stream.

I/O of multiple referred-to objects



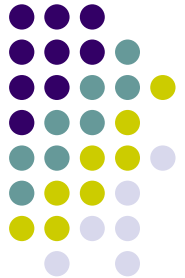
- Object a contains references to objects b and c, while b contains references to d and e



I/O of multiple referred-to objects

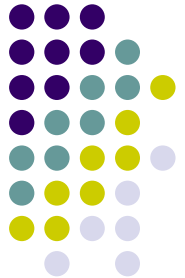
- Invoking writeObject(a) writes not just a, but all the objects necessary to reconstitute a, so the other four objects in this web are written also
- When a is read back by readObject, the other four objects are read back as well, and all the original object references are preserved.

Always Close Streams



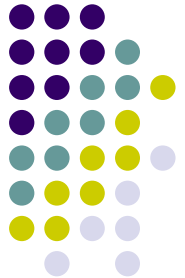
- Closing a stream when it's no longer needed is very important — so important that your program should use a finally block to guarantee that both streams will be closed even if an error occurs
  - This practice helps avoid serious resource leaks.

The *File* Class



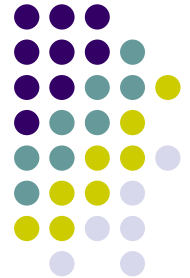
- Not a stream class
- Important since stream classes manipulate *File* objects
- Abstract representation of actual files and directory pathname

The *File* Class: Constructor & Methods



- public **File**(String pathname)
- public String **getName**()
- public boolean **exists**()
- public long **length**()
- public long **lastModified**()
- public boolean **canRead**()
- public boolean **canWrite**()
- public boolean **isFile**()
- public boolean **isDirectory**()
- public String[] **list**()
- public boolean **mkdir**()
- public boolean **delete**()

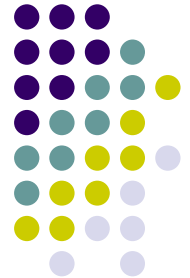
The *File* Class: Example



```
import java.io.*;

public class FileInfoClass {
   public static void main(String args[]) {
        String fileName = args[0];
        File fn = new File(fileName);
        System.out.println("Name: " + fn.getName());
        if (!fn.exists()) {
                System.out.println(fileName + " does not exists.");
                /* Create a temporary directory instead. */
                System.out.println("Creating temp directory...");
                fileName = "temp";
                fn = new File(fileName);
                fn.mkdir();
                System.out.println(fileName + (fn.exists()? "exists":
                        "does not exist"));
                System.out.println("Deleting temp directory...");
                fn.delete();
        }
```

The *File* Class: Example



```
            System.out.println(fileName + " is a " + (fn.isFile()? "file."
                    :"directory."));
             if (fn.isDirectory()) {
                    String content[] = fn.list();
                    System.out.println("The content of this directory:");
                    for (int i = 0; i < content.length; i++)
                            System.out.println(content[i]);
            }
            if (!fn.canRead()) {
                    System.out.println(fileName + " is not readable.");
                    return;
            }
            System.out.println(fileName + " is " + fn.length() + " bytes long.");
            System.out.println(fileName + " is " + fn.lastModified()
                                                    + " bytes long.");
            if (!fn.canWrite()) {
                    System.out.println(fileName + " is not writable.");
            }
    }
}
```