Chương 3
LUỒNG DỮ LIỆU

Nội dung

- Xử lý biệt lệ
- Luồng dữ liệu
- Thao tác trên tập tin

Exception Handling

Xử lý mỗi sử dụng cơ chế biệt lệ trong Java

Các cách xử lý lỗi

- Sử dụng các mệnh đề điều kiện kết hợp với các giá trị cờ.
- Sử dụng cơ chế xử lý biệt lệ.

Ví dụ: Lớp Inventory

```
public class Inventory
{
      public final int MIN = 0;
      public            final int MAX = 100;
      public final int CRITICAL = 10;
      public boolean addToInventory (int amount)
   {
         int temp;
       temp = stockLevel + amount;
       if (temp > MAX)
       {
              System.out.print("Adding " + amount + " item will cause stock ");
          System.out.println("to become greater than " + MAX + " units
                    (overstock)");
              return false;
       }
```

Ví dụ: Lớp Inventory (2)

```
    else
    {
        stockLevel = stockLevel + amount;
            return true;
    }
} // End of method addToInventory
:
```

Các vấn đề đối với cách tiếp cận điều kiện/cờ

```
reference1.method1 ()
    if (reference2.method2() == false)
        return false;
```

```
reference2.method2 ()
    if (store.addToInventory(amt) == false)
        return false;
```

```
store.addToInventory (int amt)
    if (temp > MAX)
        return false;
```

Các vấn đề đối với cách tiếp cận điều kiện/cờ

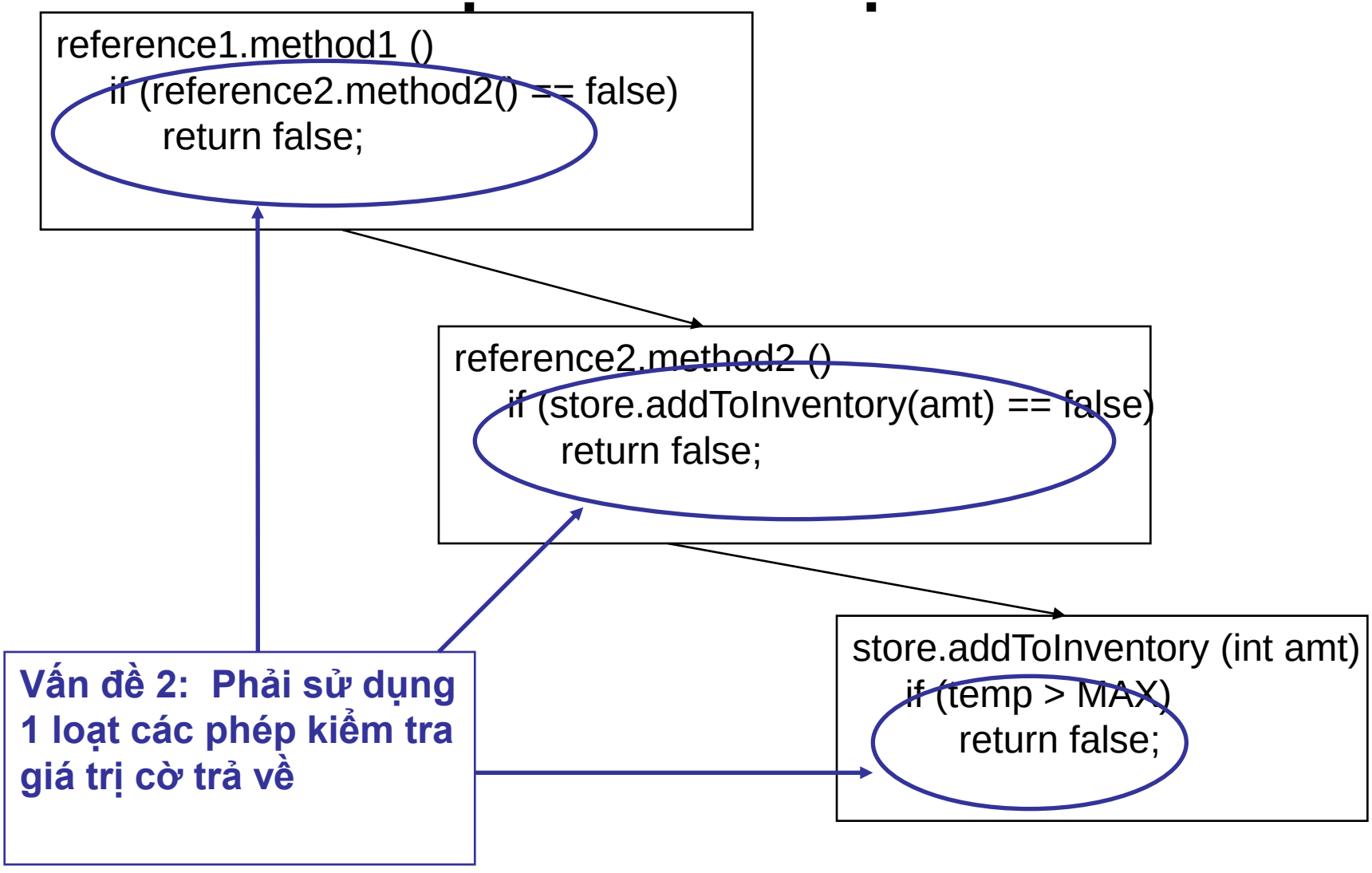
```
reference1.method1 ()
    if (reference2.method2() == false)
        return false;
```

Vấn đề 1: Phương thức chủ có thể quên kiểm tra điều kiện trả về

```
reference2.method2 ()
    if (store.addToInventory(amt) == false)
        return false;
```

```
store.addToInventory (int amt)
    if (temp > MAX)
        return false;
```

Các vấn đề đối với cách tiếp cận điều kiện/cờ



Các vấn đề đối với cách tiếp cận điều kiện/cờ

```
reference1.method1 ()
    if (reference2.method2() == false)
        return false;
```

```
reference.method2 ()
    if (store.addToInventory(amt) == false)
        return false;
```

??

??

```
store.addToInventory (int amt)
    if (temp > MAX)
        return false;
```

Vấn đề 3: Phương thức chủ có thể không biết cách xử lý khi lỗi xảy ra

Các cách xử lý lỗi

- Sử dụng các mệnh đề điều kiện kết hợp với các giá trị cờ.
- Sử dụng cơ chế xử lý biệt lệ.

Xử lý biệt lệ

- Cú pháp:

```
try
{
        // Code that may cause an error/exception to occur
}
catch (ExceptionType identifier)
{
        // Code to handle the exception
}
```

Xử lý biệt lệ: đọc dữ liệu từ bàn phím

```
import java.io.*;

class Driver
{
    public static void main (String [] args)
    {
          BufferedReader stringInput;
        InputStreamReader characterInput;
        String s;
        int num;
          characterInput = new InputStreamReader(System.in);
        stringInput = new BufferedReader(characterInput);
```

Xử lý biệt lệ: đọc dữ liệu từ bàn phím

```
try
  {
     System.out.print("Type an integer: ");
          s = stringInput.readLine();
          System.out.println("You typed in..." + s);
          num = Integer.parseInt (s);
     System.out.println("Converted to an integer..." + num);
    }
   catch (IOException e)
  {
     System.out.println(e);
  }
  catch (NumberFormatException e)
  {
                  :        :        :
       }
  }
}
```

Xử lý biệt lệ:
Biệt lệ xảy ra khi nào

```
try
  {
     System.out.print("Type an integer: ");
          s = stringInput.readLine();
          System.out.println("You typed in..." + s);
          num = Integer.parseInt (s);
     System.out.println("Converted to an integer..." + num);
    }
```

Kết quả của phương thức readLine()

```
try
  {
     System.out.print("Type an integer: ");
          s = stringInput.readLine();
          System.out.println("You typed in..." + s);
          num = Integer.parseInt (s);
     System.out.println("Converted to an integer..." + num);
    }
```

Biệt lệ có thể xảy ra ở đây

Lớp BufferedReader

http://java.sun.com/j2se/1.4.1/docs/api/java/io/BufferedReader.html

```
public class BufferedReader
{
        public BufferedReader (Reader in);
        public BufferedReader (Reader in, int sz);
        public String readLine () throws IOException;
                :
}
```

Kết quả của phương thức parseInt ()

```
try
  {
     System.out.print("Type an integer: ");
          s = stringInput.readLine();
          System.out.println("You typed in..." + s);
          num = Integer.parseInt (s);
     System.out.println("Converted to an integer..." + num);
    }
```

Biệt lệ có thể xảy ra ở đây

Lớp Integer

- http://java.sun.com/j2se/1.4.1/docs/api/java/lang/Integer.html

```
public class Integer
{
        public Integer (int value);
        public Integer (String s) throws NumberFormatException;
                :                       :
        public static int parseInt (String s) throws NumberFormatException;
                :                       :
}
```

Cơ chế xử lý biệt lệ

```
try
  {
     System.out.print("Type an integer: ");
          s = stringInput.readLine();
          System.out.println("You typed in..." + s);
          num = Integer.parseInt (s);
     System.out.println("Converted to an integer..." + num);
    }
   catch (IOException e)
  {
     System.out.println(e);
  }
  catch (NumberFormatException e)
  {
                  :        :        :
       }
  }
}
```

Cơ chế xử lý biệt lệ

```
Integer.parseInt (String s)
{
        :
        :
}
```

```
Driver.main ()
try
{
    num = Integer.parseInt (s);
}
     :
catch (NumberFormatException e)
{
    :
}
```

Cơ chế xử lý biệt lệ

```
Driver.main ()
try
{
    num = Integer.parseInt (s);
}
     :
catch (NumberFormatException e)
{
    :
}
```

```
Integer.parseInt (String s)
{
    Người sử dụng không nhập
    chuỗi số
}
```

Cơ chế xử lý biệt lệ

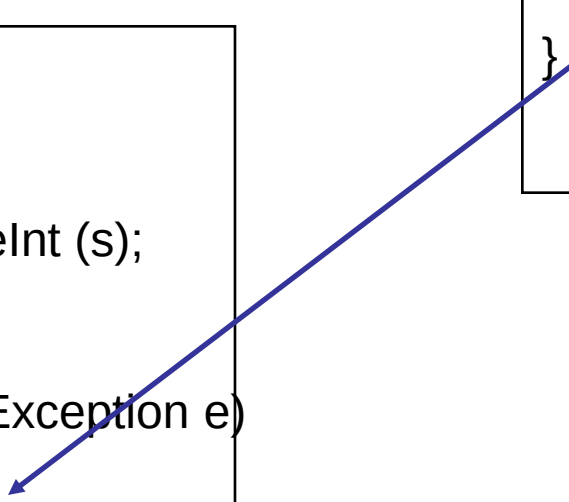
```
Driver.main ()
try
{
    num = Integer.parseInt (s);
}
     :
catch (NumberFormatException e)
{
    :
}
```

```
Integer.parseInt (String s)
{
    NumberFormatException e =
        new
    NumberFormatException ();
}
```

Cơ chế xử lý biệt lệ

```
Integer.parseInt (String s)
{
    NumberFormatException e =
        new
}   NumberFormatException ();
```

```
Driver.main ()
try
{
    num = Integer.parseInt (s);
}
     :
catch (NumberFormatException e)
{
    :
}
```



Cơ chế xử lý biệt lệ

```
Integer.parseInt (String s)
{


}
```

```
Driver.main ()
try
{
    num = Integer.parseInt (s);
}
     :
catch (NumberFormatException e)
{

}
```

Biệt lệ sẻ được xử lý ở đây

Bắt biệt lệ

```
    catch (NumberFormatException e)
    {
                :     :     :
    }
     }
}
```

Bắt biệt lệ

```
    catch (NumberFormatException e)
        {
            System.out.println(e.getMessage());
            System.out.println(e);
            e.printStackTrace();
        }
    }
}
```

Bắt biệt lệ

```
                catch (NumberFormatException e)
        {
            System.out.println(e.getMessage());
            System.out.println(e);
            e.printStackTrace();
        }
    }
}
```

Nhập vào: ”exception"

java.lang.NumberFormatException: For input string: “exception"

```
java.lang.NumberFormatException: For input string: “exception"
        at
java.lang.NumberFormatException.forInputString(NumberFormatException.java:
48)
        at java.lang.Integer.parseInt(Integer.java:426)
        at java.lang.Integer.parseInt(Integer.java:476)
        at Driver.main(Driver.java:39)
```

Các loại biệt lệ

- Biệt lệ không cần kiểm tra
- Biệt lệ phải kiểm tra

Đặc điểm của biệt lệ không cần kiểm tra

- Trình biên dịch không yêu cầu phải bắt các biệt lệ khi nó xảy ra.
  - *Không cần khối try-catch*
- Các biệt lệ này có thể xảy ra bất cứ thời điểm nào khi thi hành chương trình.
- Thông thường là những lỗi nghiêm trọng mà chương trình không thể kiểm soát
  - Xử dụng các mệnh đề điều kiện để xử lý sẽ tốt hơn.
- Ví dụ:
  - NullPointerException,IndexOutOfBoundsException, ArithmeticException…

Biệt lệ không cần kiểm tra: NullPointerException

```
int [] arr = null;
arr[0] = 1;
```


```
arr = new int [4];
int i;
for (i = 0; i <= 4; i++)
    arr[i] = i;

arr[i-1] = arr[i-1] / 0;
```

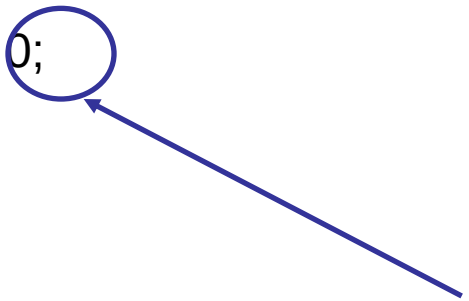
Biệt lệ không cần kiểm tra: : ArrayIndexOutOfBoundsException

```
  int [] arr = null;
arr[0] = 1;

arr = new int [4];
int i;
for (i = 0; i <= 4; i++)
   arr[i] = i;

arr[i-1] = arr[i-1] / 0;
```

ArrayIndexOutOfBoundsException
(when i = 4)

Biệt lệ không cần kiểm tra: : ArithmeticExceptions

```
  int [] arr = null;
arr[0] = 1;

arr = new int [4];
int i;
for (i = 0; i <= 4; i++)
    arr[i] = i;

arr[i-1] = arr[i-1] / 0;
```



ArithmeticException
(Division by zero)

Biệt lệ cần phải kiểm tra

- Phải xử lý khi biệt lệ có khả năng xảy ra
  - Phải sử dụng khối try-catch
- Liên quan đến 1 vấn đề cụ thể
  - Khi một phương thức được gọi thi hành
- Ví dụ:
  - IOException

Tránh bỏ qua việc xử lý biệt lệ

```
     try
{
         s = stringInput.readLine();
         num = Integer.parseInt (s);
  }
 catch (IOException e)
{
    //System.out.println(e);
}
```

Tránh bỏ qua việc xử lý biệt lệ

```
try
{
    s = stringInput.readLine();
    num = Integer.parseInt (s);
}
 catch (IOException e)
{
   System.out.println(e);
}
catch (NumberFormatException e)
{
                // Do nothing here but set up the try-catch block to bypass the
                // annoying compiler error
}
```

NO!

Mệnh đề finally

- Là 1 mệnh đề không bắt buộc trong khối try-catch-*finally.*
- Dùng để đặt khối lệnh sẽ được thi hành bất kể biệt lệ có xay ra hay không.

Mệnh đề finally: có biệt lệ

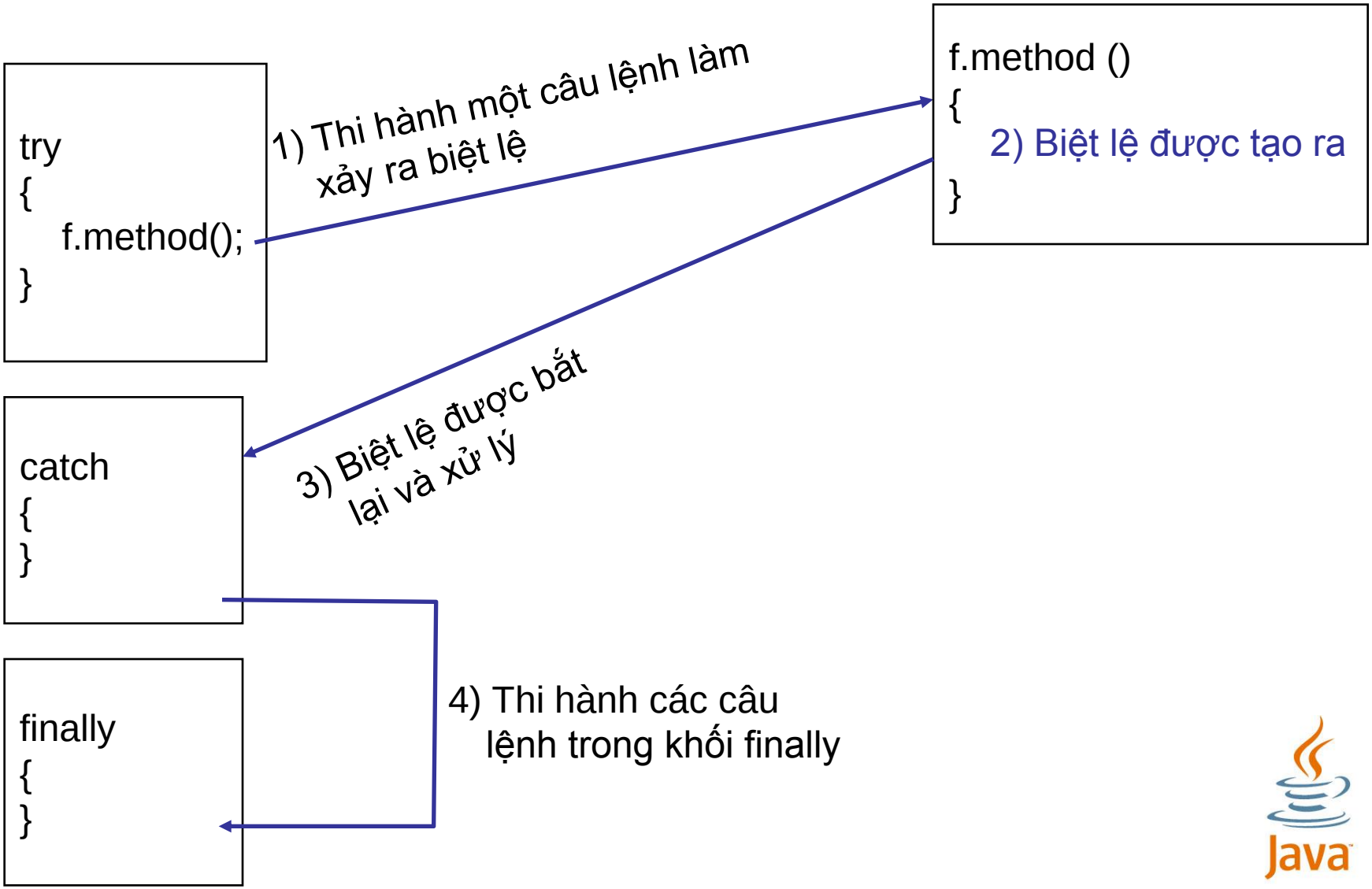
```
try
{
    f.method();
}
```

```
catch
{
}
```

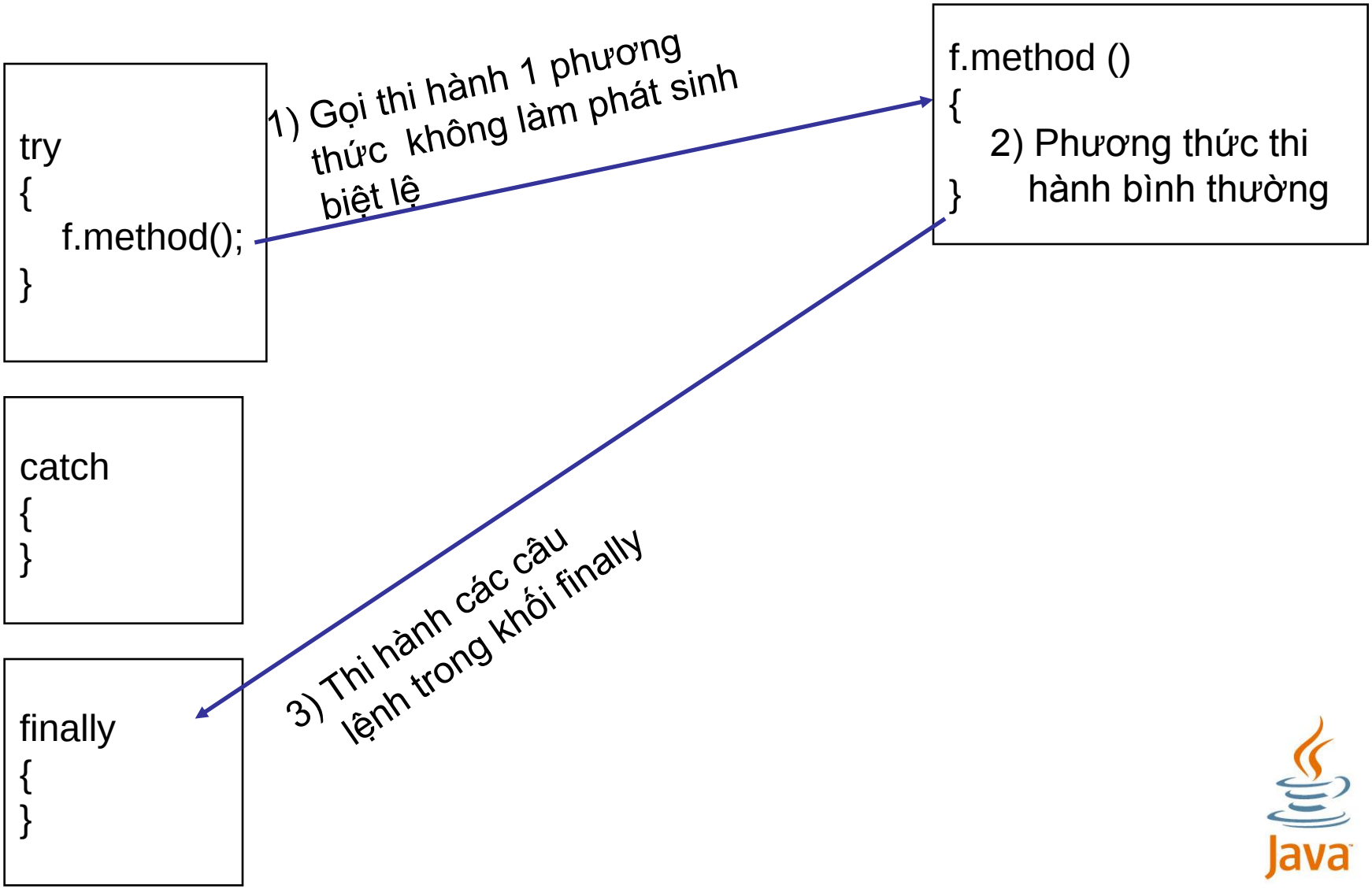
```
finally
{
}
```

```
Foo.method ()
{

}
```

Mệnh đề finally: có biệt lệ



Mệnh đề finally: không có biệt lệ



Try-Catch-Finally: Ví dụ

```
class Driver
{
    public static void main (String [] args)
    {
        TCFExample eg = new TCFExample ();
        eg.method();
    }
}
```

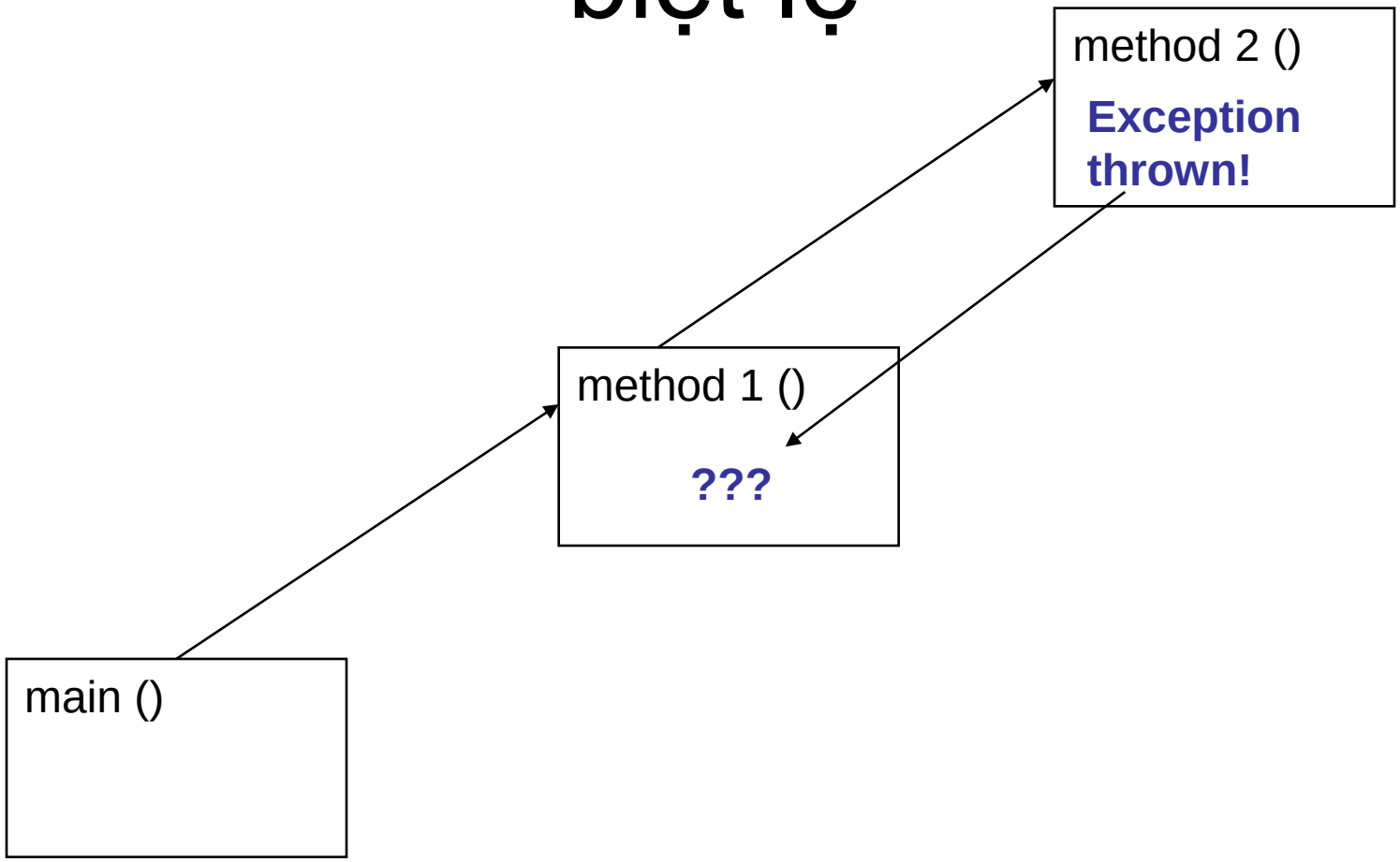
Try-Catch-Finally: Ví dụ

```
public class TCFExample
{
     public void method ()
    {
        BufferedReader br;
        String s;
        int num;
             try
        {
            System.out.print("Type in an integer: ");
            br = new BufferedReader(new InputStreamReader(System.in));
            s = br.readLine();
            num = Integer.parseInt(s);
            return;
        }
```

Try-Catch-Finally: Ví dụ

```
        catch (IOException e)
    {
        e.printStackTrace();
        return;
    }
    catch (NumberFormatException e)
    {
        e.printStackTrace ();
        return;
    }
        finally
    {
        System.out.println("<<<This code will always execute>>>");
        return;
    }
  }
}
```

Hàm được gọi không thể xử lý biệt lệ



Hàm được gọi không thể xử lý biệt lệ

```
import java.io.*;

public class TCExample
{

        public void method () throws IOException,
NumberFormatException
    {
        BufferedReader br;
        String s;
        int num;

        System.out.print("Type in an integer: ");
        br = new BufferedReader(new InputStreamReader(System.in));
        s = br.readLine();
        num = Integer.parseInt(s);
    }
}
```

Hàm được gọi không thể xử lý biệt lệ

```
class Driver
{
    public static void main (String [] args)
    {
        TCExample eg = new TCExample ();
        boolean inputOkay = true;
```

Hàm được gọi không thể xử lý biệt lệ

```
        do
        {
            try
            {
                eg.method();
                inputOkay = true;
            }
            catch (IOException e)
            {
                e.printStackTrace();
            }
            catch (NumberFormatException e)
            {
                inputOkay = false;
                System.out.println("Please enter a whole number.");
            }
        } while (inputOkay == false);
    }   // End of main
}       // End of Driver class
```

Hàm được gọi không thể xử lý biệt lệ

```
class Driver
{
    public static void main (String [] args)
    {
        TCExample eg = new TCExample ();
        boolean inputOkay = true;
```

Hàm được gọi không thể xử lý biệt lệ

```
        do
        {
            try
            {
                eg.method();
                inputOkay = true;
            }
            catch (IOException e)
            {
                e.printStackTrace();
            }
            catch (NumberFormatException e)
            {
                inputOkay = false;
                System.out.println("Please enter a whole number.");
            }
        } while (inputOkay == false);
    }// End of main
}    // End of Driver class
```

Hàm được gọi không thể xử lý biệt lệ

```
class Driver
{
    public static void main (String [] args)
    {
        TCExample eg = new TCExample ();
        boolean inputOkay = true;
```

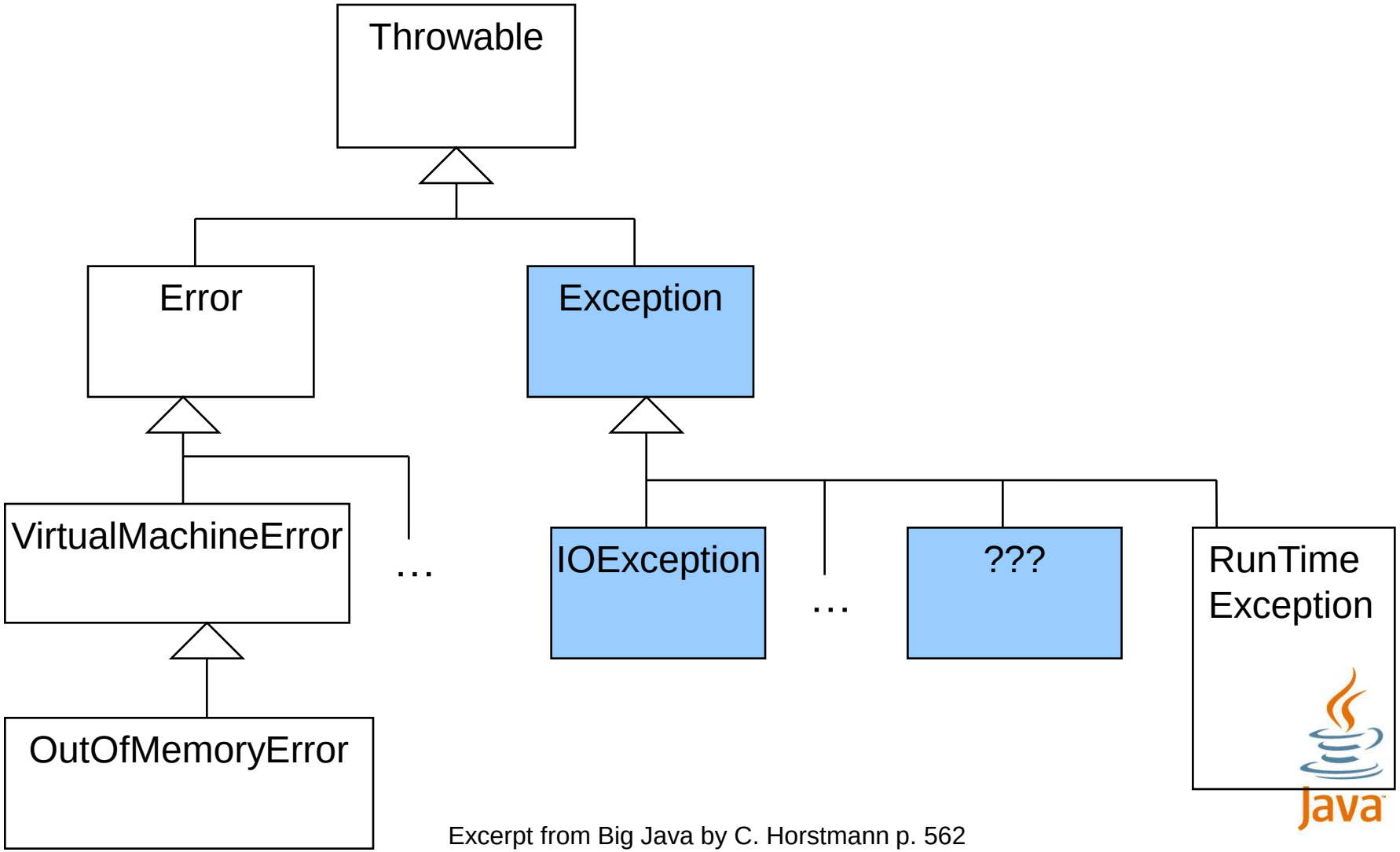
Hàm được gọi không thể xử lý biệt lệ

```
        do
        {
            try
            {
                eg.method();
                inputOkay = true;
            }
            catch (IOException e)
            {
                e.printStackTrace();
            }
            catch (NumberFormatException e)
            {
                inputOkay = false;
                System.out.println("Please enter a whole number.");
            }
        } while (inputOkay == false);
    }   // End of main
}       // End of Driver class
```

Hàm main() không xử lý biệt lệ

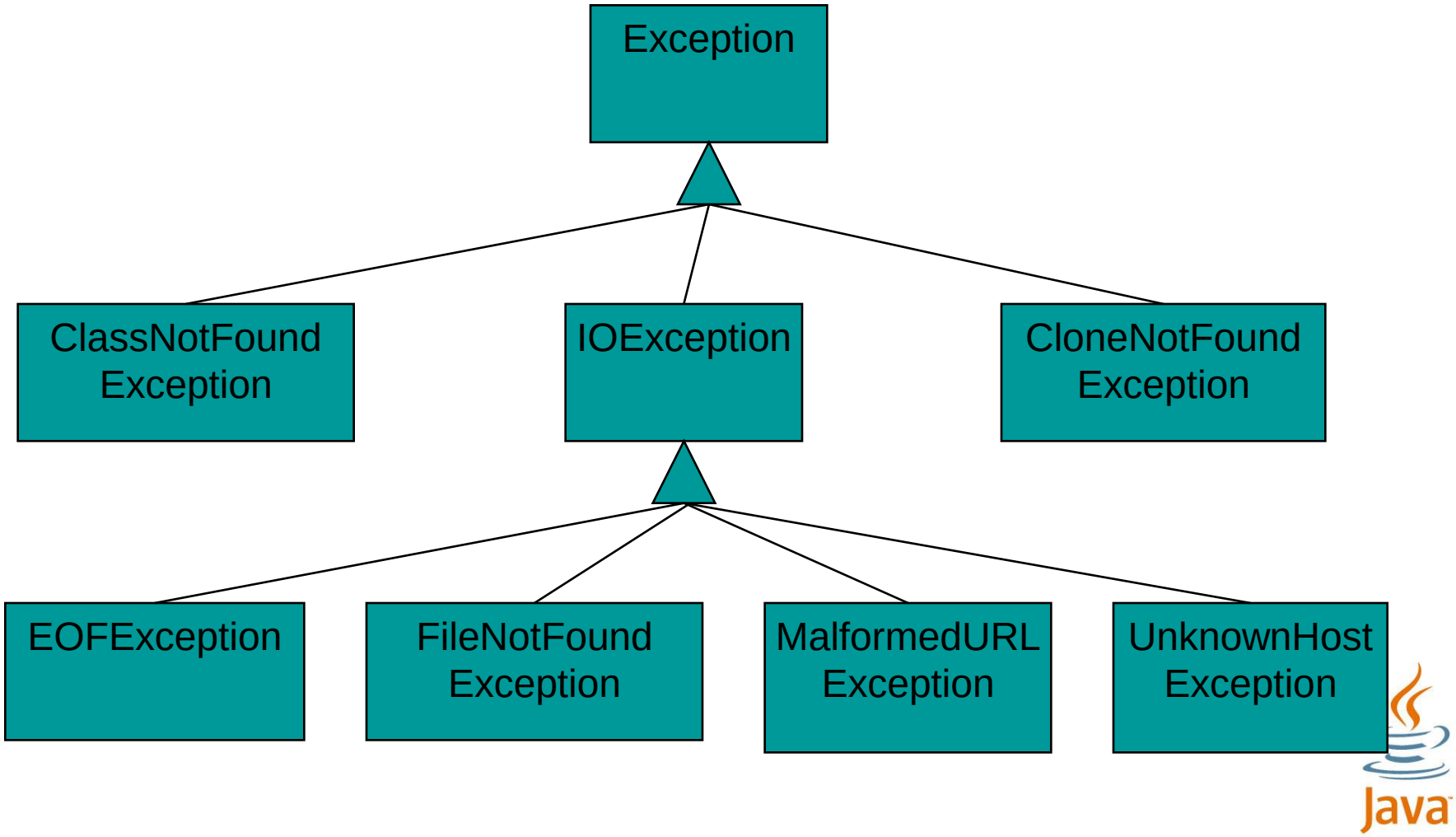
```
class Driver
{
   public static void main (String [] args) throws IOException,
            NumberFormatException
    {
        TCExample eg = new TCExample ();
        eg.method();
    }
}
```

Tạo ra kiểu biệt lệ mới



Excerpt from Big Java by C. Horstmann p. 562

Lớp Exception



Tạo biệt lệ mới

```
class Driver
{
    public static void main (String [] argv)
    {
        Inventory chinookInventory = new Inventory ();
        CommandProcessor userInterface = new
    CommandProcessor
                    (chinookInventory);
        userInterface.startProcessingInput ();
    }
}
```

Tạo biệt lệ mới

```
public class CommandProcessor
{
    private char menuOption;
    private Inventory storeInventory;

    public CommandProcessor (Inventory storeToTrack)
    {
        menuOption = 'q';
        storeInventory = storeToTrack;
    }
```

Tạo biệt lệ mới

```
 public void startProcessingInput ()
{
    do
    {
        displayMenu();
        readMenuOption();
        switch (menuOption)
        {
            case 'a':
            case 'A':
                storeInventory.getAmountToAdd();
                break;

            case 'r':
            case 'R':
                storeInventory.getAmountToRemove();
                break;
```

Tạo biệt lệ mới

```
case 'd':
case 'D':
    storeInventory.displayInventoryLevel();
    break;

case 'c':
case 'C':
    if (storeInventory.inventoryTooLow())
        System.out.println("Stock levels critical!");
     else
         System.out.println("Stock levels okay");
    storeInventory.displayInventoryLevel();
    break;

case 'q':
case 'Q':
    System.out.println("Quitting program");
    break;
```

Tạo biệt lệ mới

```
            default:
                System.out.println("Enter one of A, R, D, C or Q");
            }
    } while ((menuOption != 'Q') && (menuOption != 'q'));
}  // End of method startProcessingInput
```

Tạo biệt lệ mới

```
 protected void displayMenu ()
{
    System.out.println("\n\nINVENTORY PROGRAM: OPTIONS");
    System.out.println("\t(A)dd new stock to inventory");
    System.out.println("\t(R)emove stock from inventory");
    System.out.println("\t(D)isplay stock level");
    System.out.println("\t(C)heck if stock level is critical");
    System.out.print("\t(Q)uit program");
    System.out.println();
    System.out.print("Selection: ");
}
protected void readMenuOption ()
{
    menuOption = (char) Console.in.readChar();
    Console.in.readLine();
    System.out.println();
}
}   // End of class CommandProcesor
```

Lớp Inventory

```
public class Inventory
{
    public final static int CRITICAL = 10;
    public final static int MIN = 0;
    public final static int MAX = 100;

    private int stockLevel;
    private boolean amountInvalid;
```

Lớp Inventory

```
 public void getAmountToAdd ()
{
    int amount;
    do
    {
        System.out.print("No. items to add: ");
        amount = Console.in.readInt();
        Console.in.readLine();
        try
        {
            addToInventory(amount);
            amountInvalid = false;
        }
```

Lớp Inventory

```
        catch (InventoryOverMaxException e)
        {
            System.out.println(e);
            System.out.println("Enter another value.");
            System.out.println();
            amountInvalid = true;
        }
        finally
        {
            displayInventoryLevel();
        }
    } while (amountInvalid == true);
}     // End of method getAmountToAdd
```

Lớp Inventory

```
 public void getAmountToRemove ()
{
    int amount;
    do
    {
        System.out.print("No. items to remove: ");
        amount = Console.in.readInt();
        Console.in.readLine();
        try
        {
            removeFromInventory(amount);
            amountInvalid = false;
        }
```

Lớp Inventory

```
        catch (InventoryBelowMinException e)
        {
            System.out.println(e);
            System.out.println("Enter another value.");
            System.out.println();
            amountInvalid = true;
        }
        finally
        {
            displayInventoryLevel();
        }
    } while (amountInvalid == true);
}     // End of method getAmountToRemove
```

Lớp Inventory

```
 private void addToInventory (int amount) throws
  InventoryOverMaxException
{
    int temp;
    temp = stockLevel + amount;
    if (temp > MAX)
    {
        throw new InventoryOverMaxException ("Adding " + amount + " item
                    will cause stock to become greater than " + MAX + " units");
    }
    else
    {
        stockLevel = stockLevel + amount;
    }
}
```

Lớp Inventory

```
 private void removeFromInventory (int amount) throws
         InventoryBelowMinException
{
    int temp;
    temp = stockLevel - amount;
    if (temp < MIN)
    {
        throw new InventoryBelowMinException ("Removing " + amount
 + " item will cause stock to become less than " + MIN + " units");
    }
    else
    {
        stockLevel = temp;
    }
}
```

Lớp Inventory

```
 public boolean inventoryTooLow ()
{
    if (stockLevel < CRITICAL)
        return true;
    else
        return false;
}

public void displayInventoryLevel ()
{
    System.out.println("No. items in stock: " + stockLevel);
}

}    // End of class Inventory
```

Lớp InventoryOverMaxException

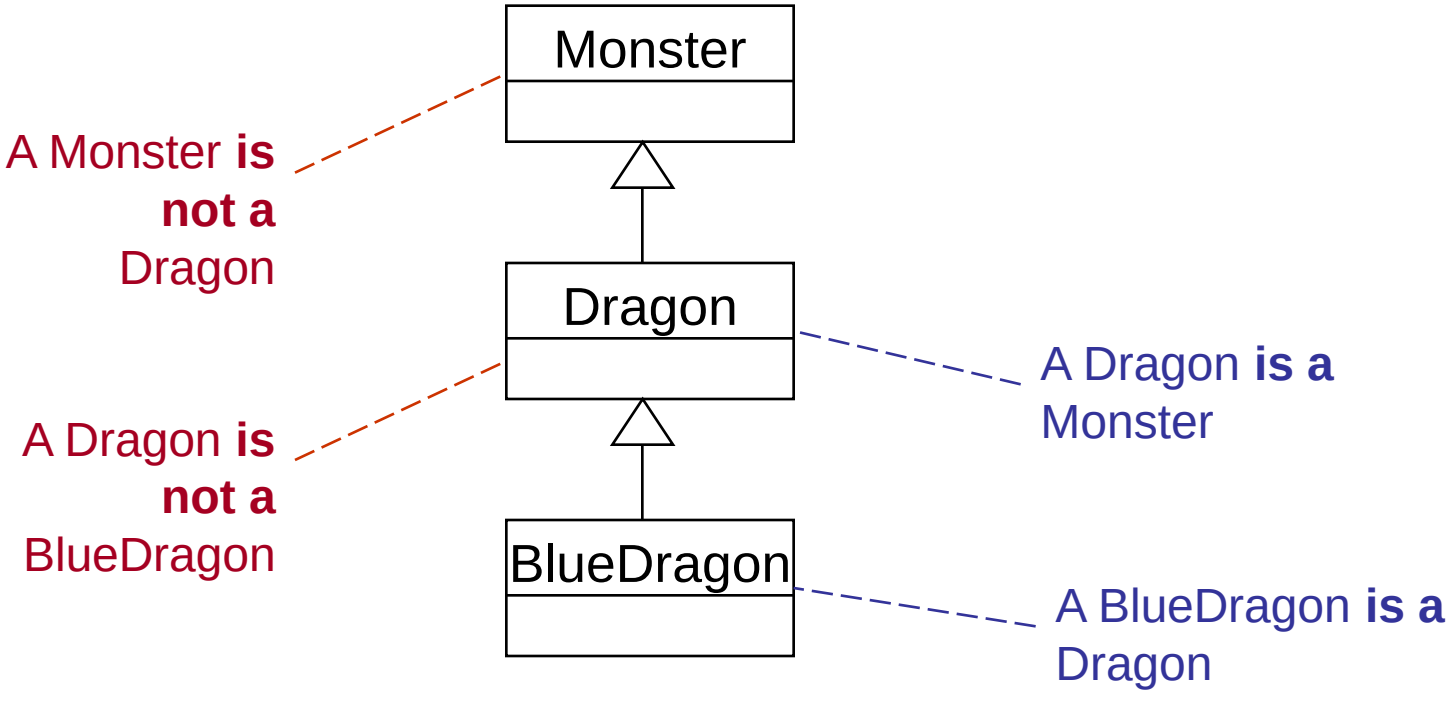
```
public class InventoryOverMaxException extends Exception
{
    public InventoryOverMaxException ()
    {
        super ();
    }

    public InventoryOverMaxException (String s)
    {
        super (s);
    }
}
```

Lớp InventoryBelowMinException

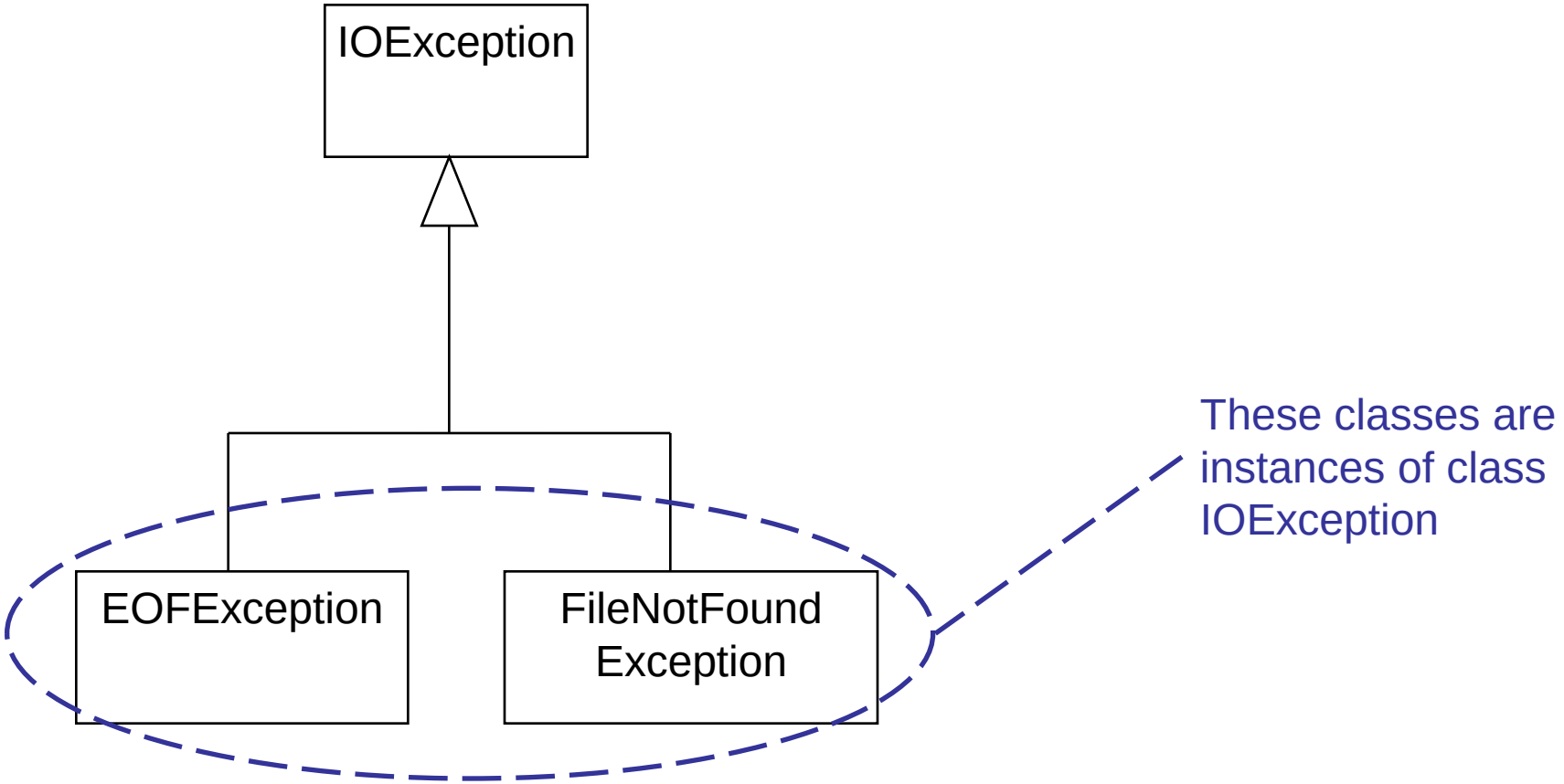
```
public class InventoryBelowMinException extends Exception
{
    public InventoryBelowMinException ()
    {
        super();
    }

    public InventoryBelowMinException (String s)
    {
        super(s);
    }
}
```

Nhắc lại thừa kế

- Có thể thay thế một đối tượng của lớp con cho 1 đối tượng của lớp cha (ngược lại không đúng).



Cây thừa kế của lớp IOExceptions



Thừa kế và vấn đề bắt biệt lệ

- Khi xử lý một chuỗi các biệt lệ cần phải đảm bảo rằng các biệt lệ lớp con được xử lý trước các biệt lệ của lớp cha.
- Xử lý các trường hợp cụ thể trước khi xử lý các trường hợp tổng quát

Thừa kế và vấn đề bắt biệt lệ

Đúng

```
try
{

}
catch (EOFException e)
{

}
catch (IOException e)
{

}
```

Sai

```
try
{

}
catch (IOException e)
{

}
catch (EOFException e)
{

}
```

Quản Lý Tập Tin & Thư Mục

- java.lang.Object
  - +--**java.io.File**
- Lớp File không phục vụ cho việc nhập/xuất dữ liệu trên luồng. Lớp File thường được dùng để biết được các thông tin chi tiết về tập tin cũng như thư mục (tên, ngày giờ tạo, kích thước, …)

Xử lý thư mục – Lớp File

- **Các Constructor:**
  - Tạo đối tượng File từ đường dẫn tuyệt đối

    *public **File**(String pathname)*

    ví dụ: *File f = new File("C:\\Java\\vd1.java");*
  - Tạo đối tượng File từ tên đường dẫn và tên tập tin tách biệt

    *public **File**(String parent, String child)*

    ví dụ: *File f = new File("C:\\Java", "vd1.java");*
  - Tạo đối tượng File từ một đối tượng File khác

    *public **File**(File parent, String child)*

    ví dụ: *File dir = new File ("C:\\Java");*

    File f = new File(dir, "vd1.java");

Xử lý thư mục – Lớp File

- Một số phương thức thường gặp của lớp File

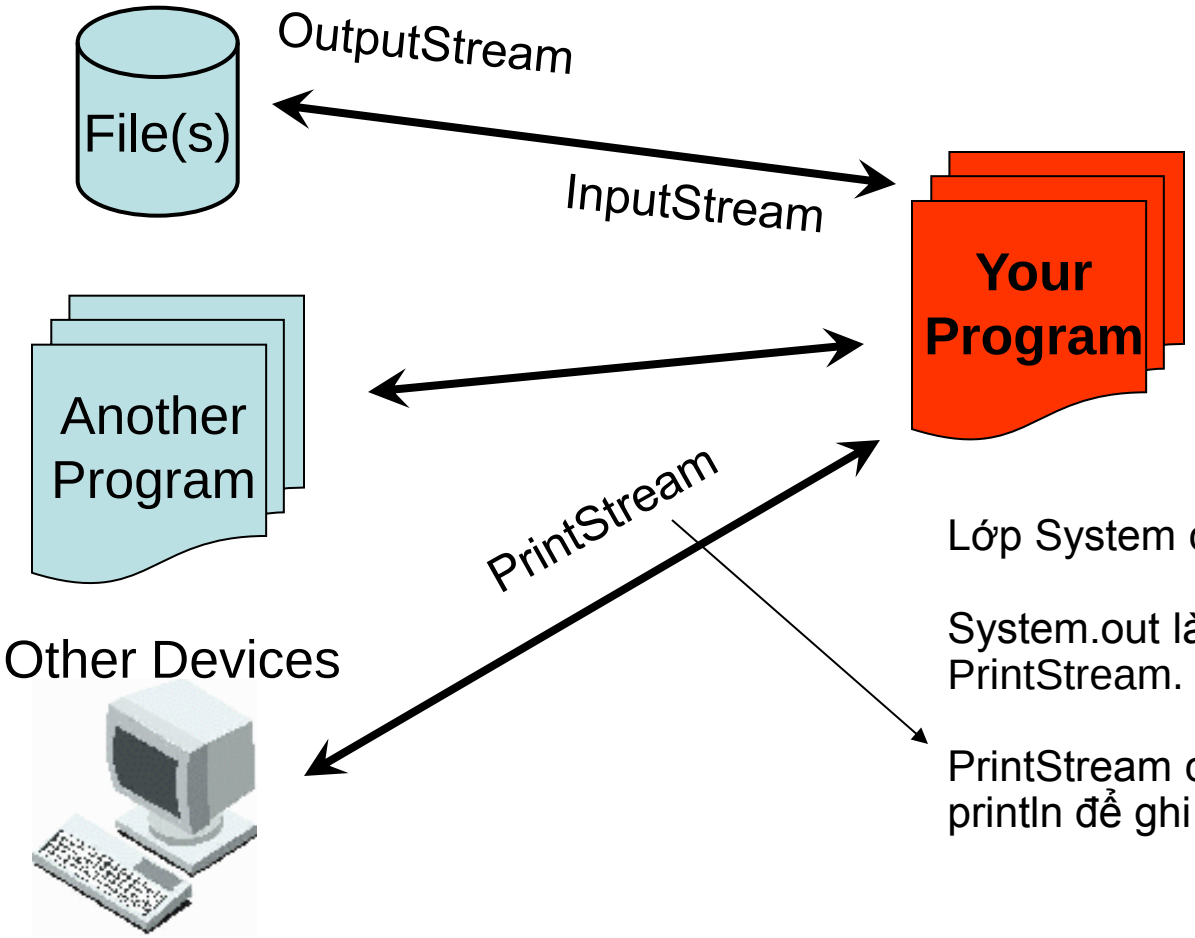
public String ***getName()***	Lấy tên của đối tượng File
public String ***getPath()***	Lấy đường dẫn của tập tin
public boolean ***isDirectory()***	Kiểm tra xem tập tin có phải là thư mục không?
public boolean ***isFile()***	Kiểm tra xem tập tn có phải là một file không?
…	
public String[] ***list()***	Lấy danh sách tên các tập tin và thư mục con của đối tượng File đang xét và trả về trong một mảng.

Xử lý trên thư mục, tập tin

```
public void copyDirectory(File srcDir, File dstDir) throws IOException {
    if (srcDir.isDirectory()) {
            if (!dstDir.exists()) { dstDir.mkdir(); }
            String[] children = srcDir.list();
            for (int i=0; i<children.length; i++) {
                    copyDirectory(new File(srcDir, children[i]), new
                                    File(dstDir, children[i]));
            }
     }
    else {
            copyFile(srcDir, dstDir);
    }
}
```

Nhập/xuất dữ liệu

Nhập xuất dữ liệu trong Java dựa trên mô hình luồng dữ liệu



Lớp System có: in, out

System.out là 1 thể hiện của lớp PrintStream.

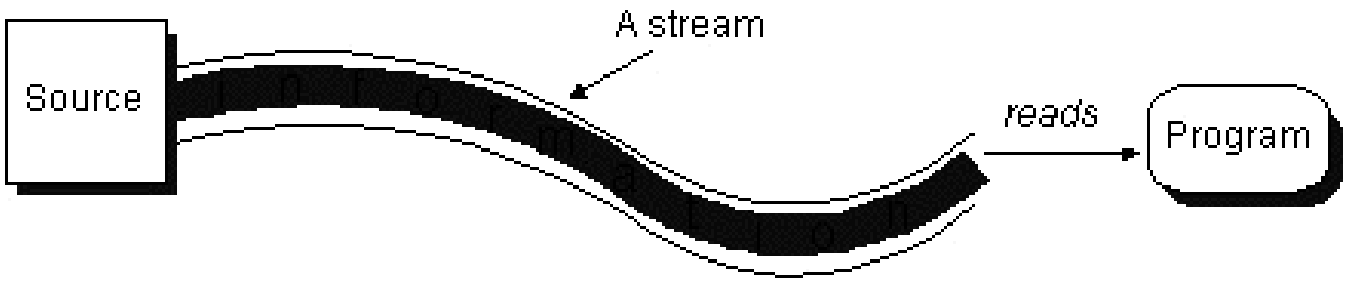
PrintStream có phương thức print, println để ghi dữ liệu xuống luồng.

Nhập xuất dữ liệu

- Dữ liệu có thể đến từ bất kỳ nguồn nào và xuất ra bất kỳ nơi nào
  - Memory
  - Disk
  - Network

- Bất kể nguồn/đích loại nào đều có thể sử dụng luồng để đọc/ghi dữ liệu.

Nhập xuất dữ liệu



Đọc dữ liệu

Open a Stream

While more Information

Read

Close the Stream

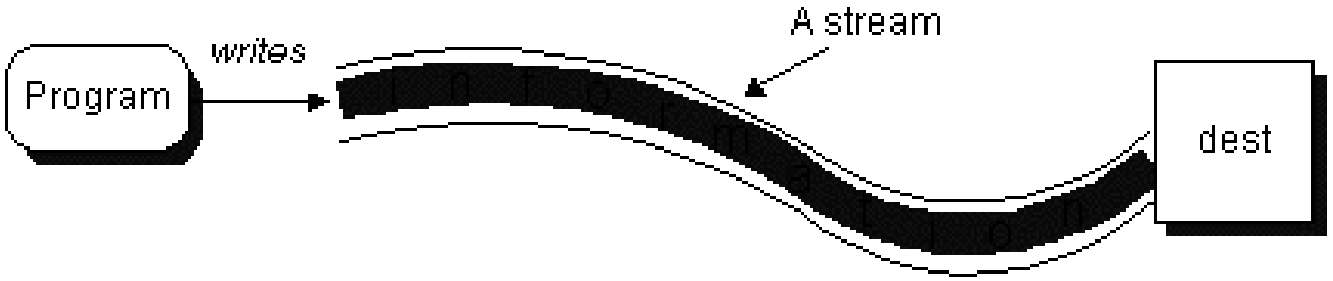
Ghi dữ liệu

Open a Stream

While more Information

Write

Close the Stream

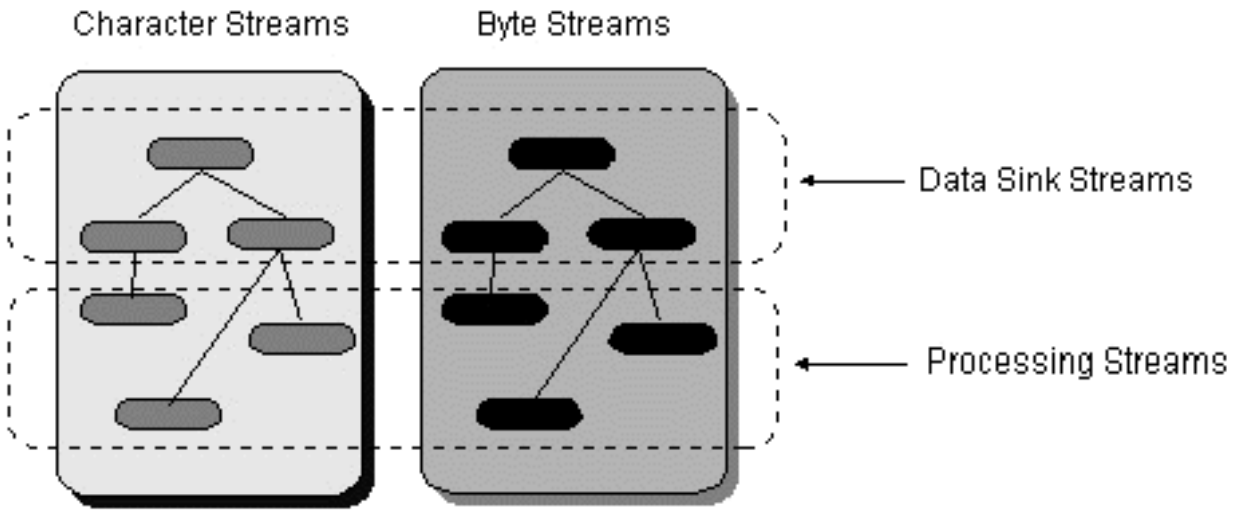


Luồng dữ liệu

- Các luồng là những đường ống dẫn để gửi và nhận thông tin trong các chương trình java.
- Khi một luồng đọc hoặc ghi , các luồng khác bị khoá.
- Nếu lỗi xẩy ra trong khi đọc hoặc ghi luồng, một ngoại lệ sẽ kích hoạt.

Luồng dữ liệu

- Gói java.io cung cấp các lớp cài đặt luồng dữ liệu.
- Phân loại luồng



Luồng dữ liệu

- *Luồng Character* được dùng khi thao tác trên ký tự (16 bits) – Sử dụng lớp *Reader* & *Writer*
- *Byte Streams* are được dùng khi thao tác dữ liệu nhị phân (8 bits) – Sử dụng *InputStream* & *OutputStream* Classes
- Data Sinks
  - Files
  - Memory
  - Pipes
- Processing
  - Buffering
  - Filtering

Các lớp luồng dữ liệu

- Lớp System.out.
- Lớp System.in.
- Lớp System.err.

Lớp InputStream

- Là lớp trừu tượng
- Định nghĩa cách nhận dữ liêu
- Cung cấp số phương thức dùng để đọc và các luồng dữ liệu làm đầu vào.
- Các phương thức:
  - **int read()**
  - **int read(byte[] buffer)**
  - **int read(byte[] buffer, int offset, int length)**
  - **int available( )**
  - **void close ( )**
  - **void reset( )**
  - **long skip( )**

Lớp InputStream

- int read()

```
byte[] b = new byte[10];
for (int i = 0; i < b.length; i++) {
   b[i] = (byte) System.in.read();
}
```

Lớp InputStream

- int read()

```
public class StreamPrinter {
    public static void main(String[] args) {
        try {
           while (true) {
              int datum =  System.in.read( );
              if (datum == -1) break;
              System.out.println(datum);
            }
        } catch (IOException ex) {
            System.err.println("Couldn't read from System.in!");
        }
    }
}
```

Lớp InputStream

- int read(byte[] buffer, int offset, int length)

```
try {
    byte[] b = new byte[100];
    int offset = 0;
    while (offset < b.length) {
        int bytesRead = System.in.read(b, offset, b.length - offset);
        if (bytesRead == -1) break; // end of stream
        offset += bytesRead;
    }
}
catch (IOException ex) {
    System.err.println("Couldn't read from System.in!");
}
```

Lớp InputStream

- int available()

```
try {
    byte[] b = new byte[System.in.available( )];
    System.in.read(b);
}
 catch (IOException ex) {
    System.err.println("Couldn't read from System.in!");
}
```

Lớp InputStream

- long skip()

```
try {
    long bytesSkipped = 0;
    long bytesToSkip = 80;
    while (bytesSkipped < bytesToSkip) {
            long n = in.skip(bytesToSkip - bytesSkipped);
            if (n == -1) break;
            bytesSkipped += n;
    }
 }
catch (IOException ex) { System.err.println(ex); }
```

Lớp OutputStream

- Là lớp trừu tượng.
- Định nghĩa cách ghi dữ liệu vào luồng.
- Cung cấp tập các phương thức trợ giúp. trong việc tạo, ghi và xử lý các luồng xuất.
- Các phương thức:
  - **void write(int)**
  - **void write(byte[ ])**
  - **write(byte[ ], int, int)**
  - **void flush( )**
  - **void close( )**

Lớp OutputStream

- void write(int i)

```
public class AsciiChart {
    public static void main(String[] args) {
        for (int i = 32; i < 127; i++) {
            System.out.write(i);
            // break line after every eight characters.
            if (i % 8 == 7) System.out.write('\n');
            else System.out.write('\t');
        }
        System.out.write('\n');
    }
}
```

Lớp OutputStream

- void write(byte[] buff)
- void write(byte[] buff, int offset, int length)

```
public class WriteBytes
    public static void main(String[] args){
        try{
            String message = “Hello World”;
            byte[] data = message.getBytes();
            System.out.write(data);
        catch(IOException e)
        {
            System.out.println(“IO errors”);
        }
    }
}
```

Lớp OutputStream

```
public class StreamCopier {
    public static void main(String[] args) {
        try {
            copy(System.in, System.out); }
        catch (IOException ex) {
            System.err.println(ex);
        }
     }
    public static void copy(InputStream in, OutputStream out) throws
                                                        IOException {
        byte[] buffer = new byte[1024];
        while (true) {
            int bytesRead = in.read(buffer);
            if (bytesRead == -1) break;
            out.write(buffer, 0, bytesRead);
        }
    }
}
```

Lớp FileOutputStream

- Cho phép kết xuất để ghi ra một luồng tập tin
- Các đối tượng cũng tạo ra sử dụng một chuỗi tên tập tin, tạp tin, hay đối tượng FileDescriptor như một tham số.
- Lớp này nạp chồng các phương thức của lớp OutputStream và cung cấp phương thức ‘finalize( )’ và ‘getFD( )’

Sử Dụng FileOutputStream

```
byte[] originalData = new byte[10];
for (int i=0; i<originalData.length; i++)
  {
  originalData[i]=(byte)(Math.random()*128.0);
  }
FileOutputStream fw=null;
try { fw = new FileOutputStream("io1.dat"); }
catch (IOException fe )
  { System.out.println(fe); }
for (int i=0; i<originalData.length; i++)
  {
  try { fw.write(originalData[i]); }
  catch (IOException ioe)
  {System.out.println(ioe); }
  }
try { fw.close(); }
catch (IOException ioe)
  {System.out.println(ioe); }
```

Lớp FileInputStream

- Cho phép đầu vào đọc từ một tập tin trong một mẫu của một dòng
- Các đối tượng được tạo ra sử dụng chuỗi tên tập tin, tập tin, đối tượng FileDescriptor như một tham số.
- Các phương thức nạp chồng của lớp InputStream. nó cung cấp phương thức ‘finalize( )’ và ‘getFD( )’

Sử Dụng FileInputStream

```
byte data=0;
int bytesInFile=0;
FileInputStream fr = null;
try
  {
  fr = new FileInputStream("io1.dat");
  bytesInFile = fr.available();
  for (int i=0; i<bytesInFile; i++)
      {
      data=(byte)fr.read();
      System.out.println("Read "+data);
      }
  fr.close();
  }
catch (IOException ioe)
  {
  System.out.println("Problem with file");
  }
```

ByteArrayInput

- Sử dụng các đệm bộ nhớ
- Lớp **ByteArrayInputStream**
  - **ByteArrayInputStream(byte[] buf)**
- Tạo ra một luồng nhập từ đệm bộ nhớ mảng các byte.
  - Không hỗ trợ các phương thức mới
  - Các phương thức nộp chồng của lớp InputStream, giống như ‘read()’, ‘skip()’, ‘available()’ và ‘reset()’.

Byte Array Output

- sử dụng các vùng đệm bộ nhớ
- Lớp **ByteArrayOutputStream**
  - Tạo ra một luồng kết xuất trên mảng byte
  - Cung cấp các khả năng bổ sung cho mảng kết xuất tăng trưởng nhằm chừa chổ cho dữ liệu mới ghi vào.
  - Cũng cung cấp các phương thức để chuyển đổi luồng tới mảng byte, hay đối tượng String.

- Phương thức của lớp **ByteArrayOutputStream** :
  - **ByteArrayOutputStream()**
  - **void reset( )**
  - **int size( )**
  - **byte[] toByteArray()**
  - **String toString()**

Bộ lọc

- Lọc:
  - Là kiểu luồng sửa đổi cách điều quản một luồng hiện có.
  - về cơ bản được sử dụng để thích ứng các luồng theo các nhu cầu của chương trình cụ thể.
  - Bộ lọc nằm giữa luồng cơ sở và CT.
  - Thực hiện một số tiến trình đặt biệt trên các byte được chuyển giao từ đầu vào đến kết xuất.
  - Có thể phối hợp để thực hiện một dãy các tuỳ chọn lọc.

Lớp FilterInputStream

- Là lớp trừu tượng.
- Là cha của tất cả các lớp luồng nhập đã lọc.
- Cung cấp khả năng tạo ra một luồng từ luồng khác.
- Một luồng có thể đọc và cung cấp cung cấp dưới dạng kết xuất cho luồng khác.
- duy trì một dãy các đối tượng của lớp ‘InputStream’
- Cho phép tạo ra nhiều bộ lọc kết xích (chained filters
-).

Lớp FilterOutputStream

- Là dạng bổ trợ cho lớp 'FilterInputStream'.
- Là cha của tất cả các lớp luồng kết xuất.
- Duy trì đối tượng của lớp 'OutputStream' như là một biến 'out'.
- Dữ liệu ghi ra lớp này có thể sửa đổi để thực hiện các thao tác lọc, và sau đó phản hồi đến đối tượng 'OutputStream'.

Vùng đệm nhập/xuất

- Vùng đệm:
  - Là kho lưu trữ dữ liệu.
  - Có thể cung cấp dữ liệu thay vì quay trợ lại nguồn dữ liệu gốc ban đầu.
  - Java sử dụng vùng đệm nhập và kết xuất để tạm thời lập cache dữ liệu được đọc hoặc ghi vào một luồng.
- Trong khi thực hiện vùng đệm nhập:
  - Số lượng byte lớn được đọc cùng thời điểm, và lưu trữ trong một vùng đệm nhập.
  - Khi chương trình đọc luồng nhập, các byte nhập được đọc vào vùng đệm nhập.

Vùng đệm nhập/xuất (tt…)

- Trong trường hợp vùng đệm kết xuất, một chương trình ghi ra một luồng.
- Dữ liệu kết xuất đựơc lưu trữ trong một vùng đệm kết xuất.
- Dữ liệu được lưu trữ cho đến khi vùng đệm trợ nên đầy, hay luồng kết xuất được xả trống.
- Kết thúc, vùng đệm kết xuất được chuyển gửi đến đích của luồng xuất.

Lớp BufferedInputStream

- Tự động tạo ra và duy trì vùng đệm để hổ trợ vùng đệm nhập.
- bởi lớp ‘BufferedInputStream’ là một bộ đệm, nó có thể áp đụng cho một số các đối tượng nhất định của lớp ‘InputStream’.
- Cũng có thể phối hợp các tập tin đầu vào khác.
- Sử dụng vài biến để triển khai vùng đệm nhập.

Lớp BufferedInputStream (Contd…)

- Định nghĩa hai phương thức thiết lập:
  - Một chó phép chỉ định kích thước của vùng đệm nhấp.
  - phương thức kia thì không.
- Cả hai phương thức thiết lập đều tiếp nhận một đối tượng của lớp ‘InputStream’ như một tham số.
- Nạp chồng các phương thức truy cập mà InputStream cung cấp, và không đưa vào bất kỳ phương thức mới nào.

Lớp BufferedOutputStream

- Thực hiện vùng đệm kết xuất theo cách tương ứng với lớp ‘BufferedInputStream’.
- Định nghĩa hai phương thức thiết lập. Nó cho phép chúng ta ấn định kích thước của vùng đệm xuất trong một phương thức thiết lập, cũng giống như cung cấp kích thước vùng đệm mặc định.
- Nạp chồng tất cả phương thức của lớp ‘OutputStream’ và không đưa vào bất kỳ phương thức nào.

Giao diện DataInput

- Được sử dụng để đọc các byte từ luồng nhị phân, và xây dựng lại dữ liệu trong một số kiểu dữ liệu nguyên thuỷ.
- Cho phép chúng ta chuyển đổi dữ liệu từ từ khuôn dạng UTF-8 được sửa đổi Java đến dạng chuỗi
- Định nghiã số phương thức, bao gồm các phương thức để đọc các kiểu dữ liệu nguyên thuỷ.

Những phương thức giao diện DataInput

- **boolean readBoolean( )**
- **byte readByte( )**
- **char readChar( )**
- **short readShort( )**
- **long readLong( )**
- **float readFloat( )**
- **int readInt( )**
- **double readDouble**
- **String readUTF( )**
- **String readLine( )**

Giao diện DataOutput

- Được sử dụng để xây dựng lại dữ liệu một số kiểu dữ liệu nguyên thuỷ vào trong dãy các byte
- Ghi các byte dữ liệu vào luồng nhị phân
- Cho phép chúng ta chuyển đổi một chuỗi vào khuôn dạng UTF-8 được sửa đổi Java và viết nó vào trong một dãy.
- Định nghĩa một số phương thức và tất cả phương thức kích hoạt IOException trong trường hợp lỗi.

Các phương thức giao diện DataOutput

- **void writeBoolean(boolean b)**
- **void writeByte(int value)**
- **void writeChar(int value)**
- **void writeShort(int value)**
- **void writeLong(long value)**
- **void writeFloat(float value)**
- **void writeInt(int value)**
- **void writeDouble(double value)**
- **void writeUTF(String value)**

Mảng byte sang int

```
public class ArrayCopy {

    public static int[] byte2int(byte[]src) {
        int dstLength = src.length >>> 2;
        int[]dst = new int[dstLength];

        for (int i=0; i<dstLength; i++) {
            int j = i << 2;
            int x = 0;
            x += (src[j++] & 0xff) << 0;
            x += (src[j++] & 0xff) << 8;
            x += (src[j++] & 0xff) << 16;
            x += (src[j++] & 0xff) << 24;
            dst[i] = x;
        }
        return dst;
    }
}
```

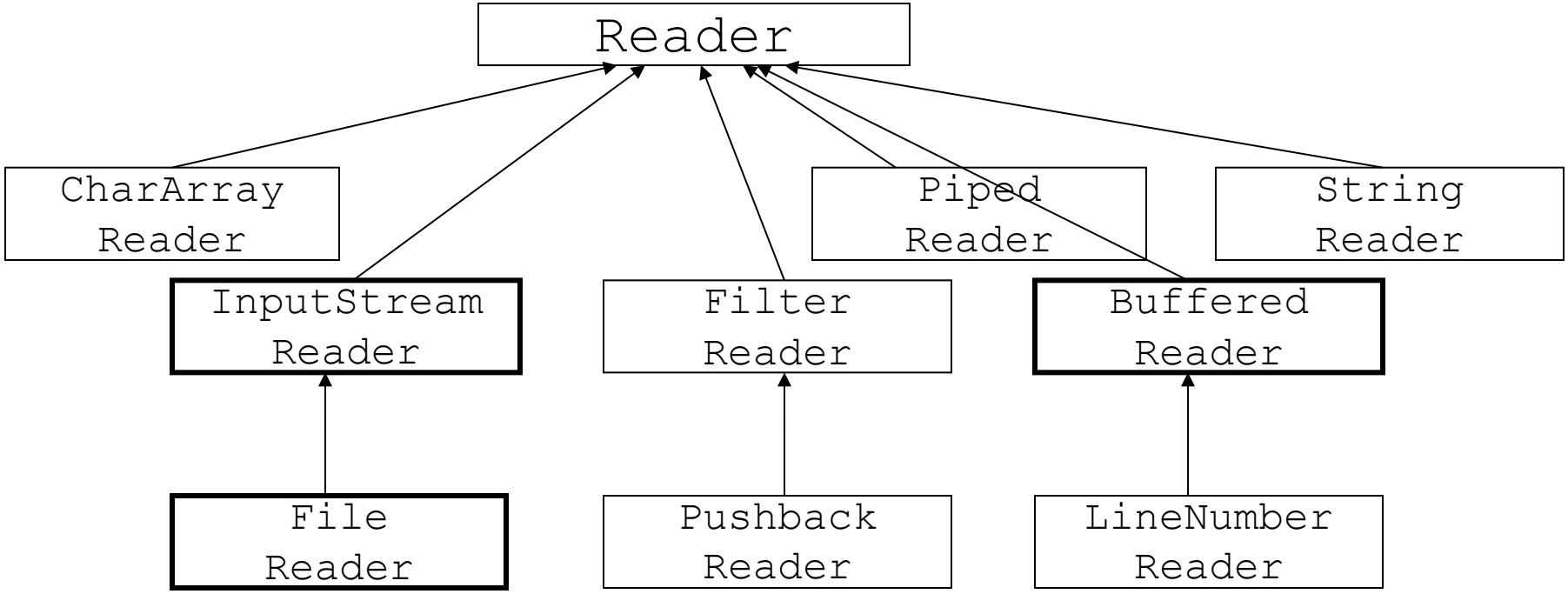
Sử Dụng DataOutputStream

```
int[] originalData = new int[10];
for (int i=0; i<originalData.length; i++)
  originalData[i]=(int)(Math.random()*1000.0);
FileOutputStream fos=null;
try { fos = new FileOutputStream("io2.dat"); }
catch (IOException fe )
  { System.out.println("Cant make new file"); }
DataOutputStream dos = new DataOutputStream(fos);
for (int i=0; i<originalData.length; i++)
  {
  try { dos.writeInt(originalData[i]); }
  catch (IOException ioe)
   {System.out.println("Cant write to file"); }
  }
```

Lớp Reader và Writer

- Là các lớp trừu tượng.
- Chúng nằm tại đỉnh của hệ phân cấp lớp, hỗ trợ việc đọc và ghi các luồng ký tự unicode.

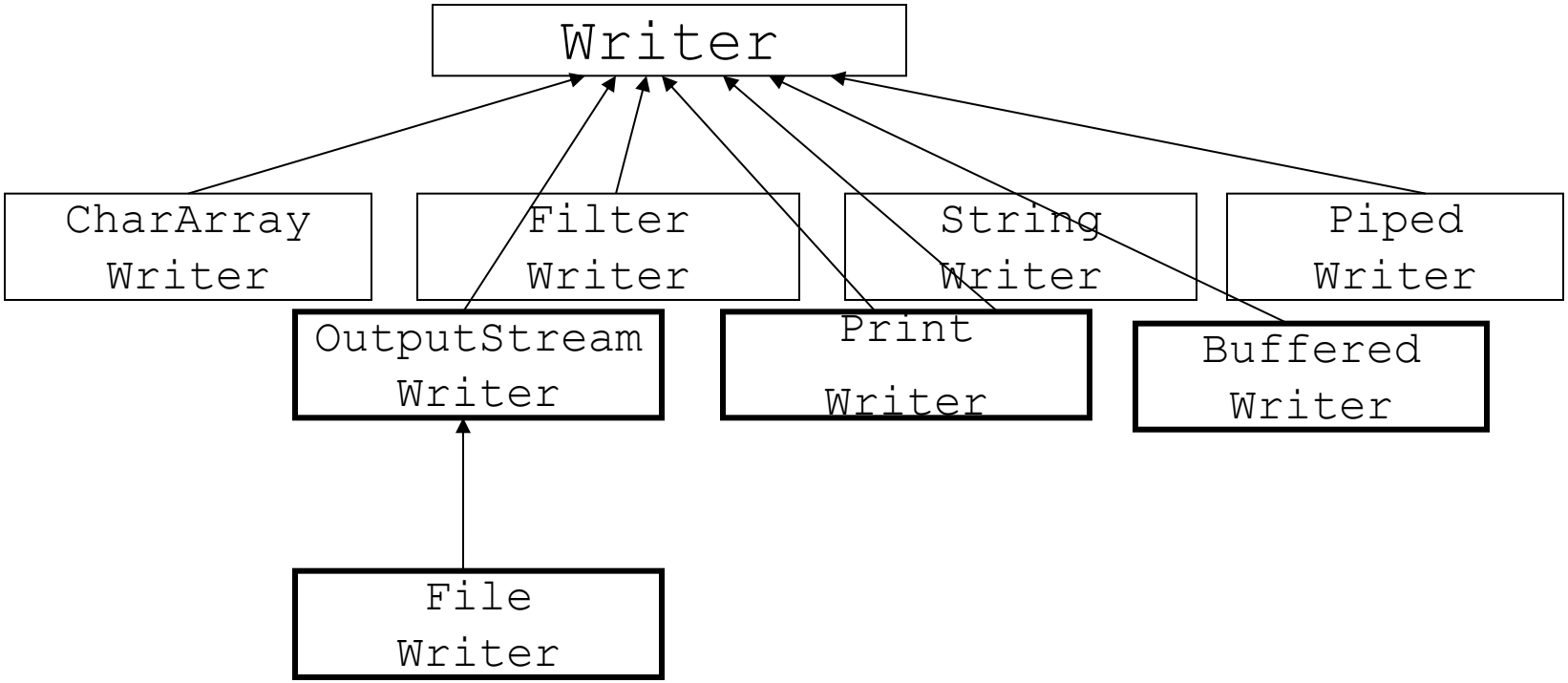
Lớp Reader


Reader
CharArray Reader
InputStream Reader
Filter Reader
Piped Reader
Buffered Reader
String Reader
File Reader
Pushback Reader
LineNumber Reader

Lớp Reader

- Hỗ trợ các phương thức sau:
  - **int read( )**
  - **int read(char[] data)**
  - **int read(char[] data, int offset, int len)**
  - **void reset( )**
  - **long skip( )**
  - **void close( )**
  - **boolean ready( )**

Lớp Writer


Writer
CharArray Writer
Filter Writer
String Writer
Piped Writer
OutputStream Writer
Print Writer
Buffered Writer
File Writer

Lớp Writer

- Hỗ trợ các phương thức sau :
  - **void write(int ch)**
  - **void write(char[] text)**
  - **void write(String str)**
  - **void write(String str, int offset, int len)**
  - **void flush( )**
  - **void close( )**

Đọc Tập Tin Văn Bản

```
import java.io.*;
public class FileRead
{
   public static void main(String[] args)  throws IOException
   {
        //all the following, up to the definition of the reader, are in
        //the class java.io.File, which contains a number of methods
        //related to the file attributes and the directory it resides in.

        String fileName = args[0];     // args[0] for file name

       //the next statement creates a reader for the file
        BufferedReader dat = new BufferedReader(new FileReader(fileName));
        System.out.println("DATA FROM THE FILE: " + fileName);

        String line = dat.readLine();
        while (line != null)
        {
            System.out.println(line);
            line = dat.readLine();
        }
        System.out.println("END OF FILE REACHED");

        dat.close();
   }
}
```

//If there is no more data in the file, readLine returns **null**

Đọc Tập Tin Nhị Phân

```
import java.io.*;
public class IntFileRead
{
  public static void main(String[] args)  throws IOException
  {
    File dataf = new File ("data.txt");
    int number;

    //Create a reader for the file
    FileReader fdat = new FileReader(dataf);
    BufferedReader dat = new BufferedReader(fdat);
    System.out.println("DATA FROM THE FILE:");

    String line = dat.readLine();
    while (line != null)
    {
        number=Integer.parseInt(line);
        System.out.println(number);
        line = dat.readLine();
    }

    System.out.println("END OF FILE REACHED");
    dat.close();
  }//end main
}//end IntFileRead
```

Lớp PrinterWriter

- Thực hiện một kết xuất.
- Lớp này có phương thức bổ sung , trợ giúp in các kiểu dữ liệu cơ bản .
- Lớp PrintWriter thay thế lớp ‘PrintStream’
- Thực tế cải thiện lớp ‘PrintStream’; lớp này dùng một dấu tách dòng phụ thuộc nền tảng điểm các dòng thay vì ký tự ‘\n’.
- Cung cấp phần hỗ trợ cho các ký tự unicode so với PrintStream.
- Các phương thức:
  - **checkError( )**
  - **setError( )**

Ghi Xuống Tập Tin

```
import java.io.*;
public class TextFileWrite
{
   public static void main(String[] args)  throws IOException
   {
        //for this example, set up data we want to write as a four element array of strings
        String [] song = new String [4];
        song[0]="Mary had a little lamb";
        song[1]="Its fleece was white as snow";
        song[2]="And everywhere that Mary went";
        song[3]="The lamb was sure to go";

        //set up the output file to be written to
       String outFileName = args[0];
        //using PrintWriter allows us to use the print and println commands for files
       File outData = new File(outFileName);
       PrintWriter outDat = new PrintWriter(new FileWriter(outData));

       //Now write the data .....
       for (int line=0; line<song.length; line++)
            outDat.println(song[line]);
       System.out.println("DATA WRITTEN ON OUTPUT FILE: "+ outFileName);
       outDat.close();
   }
}
```

Lớp RandomAccessFile

- Cung cấp khả năng thực hiện I/O theo các vị trí cụ thể bên trong một tập tin.
- dữ liệu có thể đọc hoặc ghi ngẫu nhiên ở những vị trí bên trong tập tin thay vi một kho lưu trữ thông tin liên tục.
- phương thức ‘seek( )’ hỗ trợ truy cập ngẫu nhiên.
- Thực hiện cả đầu vào và đầu ra dữ liệu.
- Hỗ trợ các cấp phép đọc và ghi tập tin cơ bản.
- Kế thừa các phương thức từ các lớp ‘DataInput’ và ‘DataOutput’

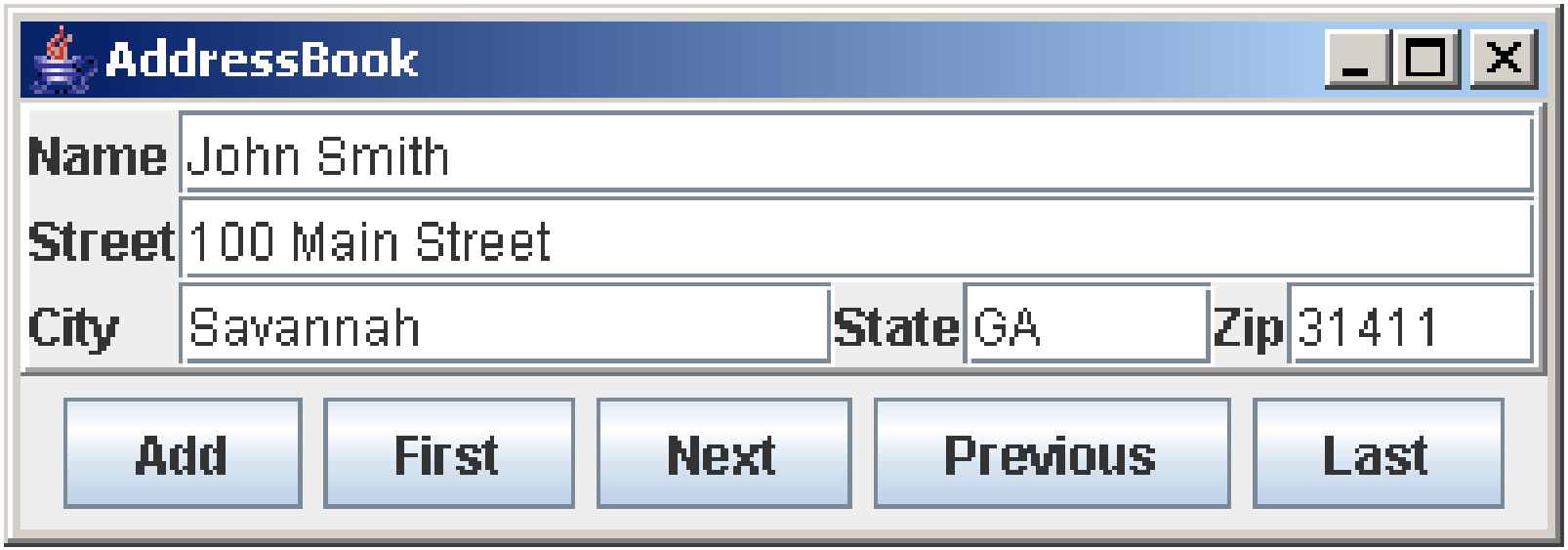
Các phương thức của lớp RandomAccessFile

- **RandomAccessFile(String fn,String mode)**
  “r”, “rw”, …..
- **seek( )**
- **getFilePointer( )**
- **length( )**
- **readBoolean()**
- **….**
- **writeBoolean()**
- **…..**

RandomAccessFile example

```
RandomAccessFile rf = new
    RandomAccessFile(“doubles.dat”,”rw”);

// read third double: go to 2 * 8 (size of one)
rf.seek(8*2);
double x = rf.readDouble();

// overwrite first double value
rf.seek(0);
rf.writeDouble(x);
rf.close();
```

Bài tập


AddressBook
Name
John Smith
Street
100 Main Street
City
Savannah
State
GA
Zip
31411
Add
First
Next
Previous
Last

Serialization

- Thao tác đọc và ghi các đối tượng
- Serialization cũng được hỗ trợ trong các ngôn ngữ khác nhưng rất khó thực hiện
- Java giúp việc thực hiện serializtion rất dễ dàng

ĐK để có thể serializability

- Đối tượng được serialized nếu:
  - Lớp là public
  - Lớp phải implement Serializable
  - Lớp phải có no-argument constructor

Writing objects to a file

```
ObjectOutputStream objectOut =
   new ObjectOutputStream(
      new BufferedOutputStream(
         new
FileOutputStream(fileName)));

objectOut.writeObject(serializableObje
ct);

objectOut.close( );
```

Reading objects from a file

```
ObjectInputStream objectIn =
   new ObjectInputStream(
      new BufferedInputStream(
         new FileInputStream(fileName)));

myObject = (itsType)objectIn.readObject(
);

objectIn.close( );
```