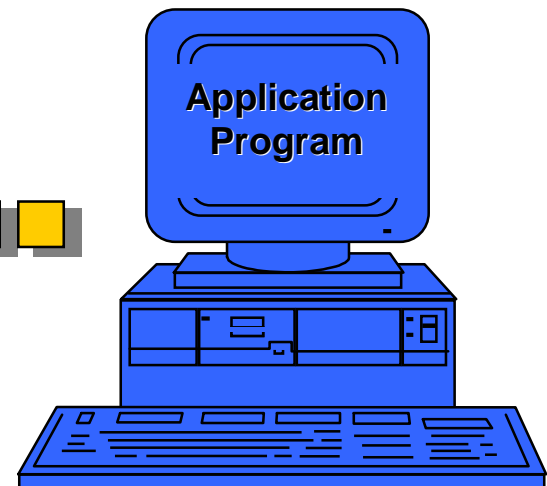
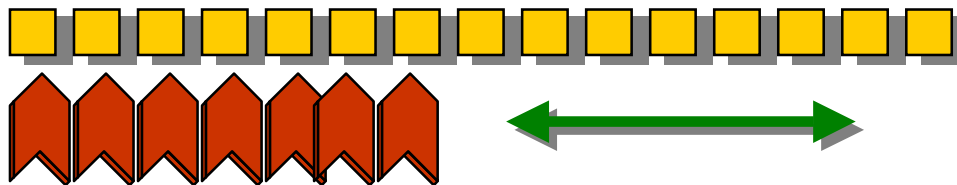
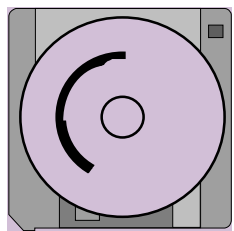


Java 高级编程



- **Java异常处理**
- **Java多线程编程**
- **Java的I/O编程**
- **Java的网络编程**



• Java异常

- 是特殊的运行错误对象，对应着Java语言特定的运行错误处理机制

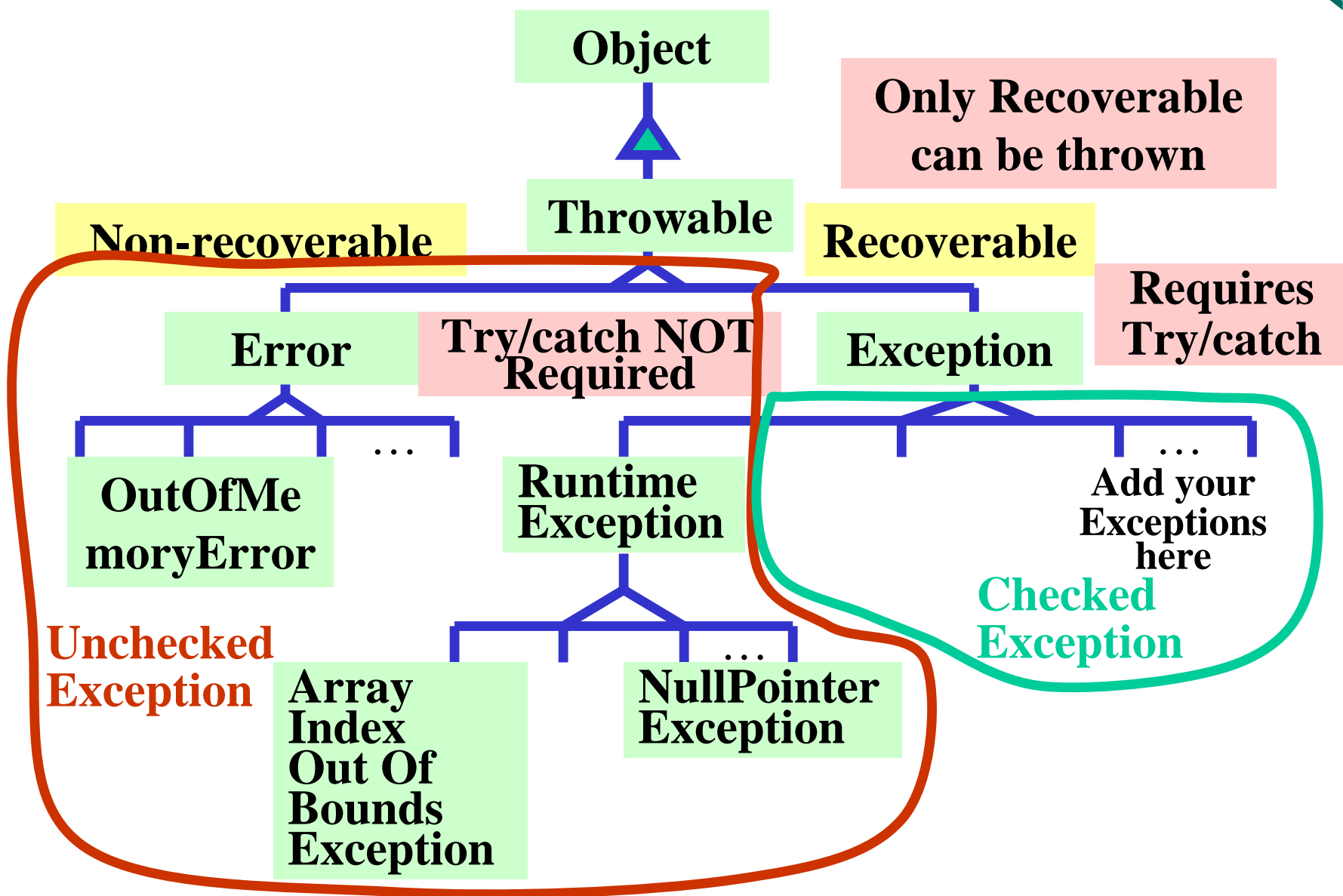
• Java异常类

- 每个异常类都代表了一种运行错误，类中包含了该运行错误的信息和处理错误的方法等内容

• Java异常机制

- 每当Java 程序运行过程中发生一个可识别的运行错误时，系统会产生一个相应的异常类的对象，即产生一个异常。系统中就一定有相应的机制来处理它，从而保证整个程序运行的安全性





- 系统定义的运行异常： P250

- 系统定义的异常主要用来处理系统可预见的较常见的运行错误
- `Exception`类的若干子类，每个子类代表了一种特定的运行错误

- 用户自定义的异常： P251

- 用户定义的异常主要用来处理用户程序中特定的逻辑运行错误
- 声明一个新的异常类，扩展`Exception`类（直接或间接）
- 为新的异常类定义属性或方法，或重载父类的属性和方法，使这些属性或方法能够体现该类所对应的错误的信息



```
class Worker { //定义一个可以抛出Except1, Except2 异常的方法
    public void f1(int val) throws Except1, Except2 {
        switch (val) {
            case1: throw new Except1("Gotcha!");
            case2: throw new Except2("Againnn"); }
        }
    }
}
```

```
class Except1 extends Exception { //定义Except1异常
    private String theMessage;
    public Except1(String aMsg) {
        theMessage = aMsg; }
    public String toString() {
        return "Except1" + theMessage; }
}
```



```
class Except2 extends Exception { //定义Except2异常
    private String theMessage;
    public Except2(String aMsg) {
        theMessage = aMsg; }
    public String toString() {
        return "Except2" + theMessage; }
}
```



```

public class except_example {
    public static void main(String args[]) {
        Worker dobj = new Worker();
        for (int i = 0; i < 3; i++) {
            try { //异常处理模块
                dobj.f1(I);
                System.out.println("No except," + i);    }
            catch (Except1 e) { //捕捉Except1异常
                System.out.println(e.toString() + i);    }
            catch (Except2 e) { //捕捉Except2异常
                System.out.println(e.toString() + i);    }
            finally {
                System.out.println("Finally code");    }
        }
    }
}
    
```



- Java程序在运行时如果引发了一个可识别的错误，就会产生一个与该错误相对应的异常类的对象，这个过程被称为异常的抛出
- 系统自动抛出的异常： P252
 - 系统定义的异常主要用来处理系统可预见的较常见的运行错误

```
Public class TestsystemException {  
    public static void main(String args[])  
    {  
        int a=0, b=5;  
        System.out.println(b/a);  
    }  
} //throw an ArithmeticException object
```



- 语句抛出的异常： P253

修饰符 返回类型 方法名 (参数列表) throws 异常类名列表

{ ... throw 异常类名 ... }

```
int dequeue() throws EmptyQueueException {  
    int data;  
    if (isEmpty())  
        throw (new EmptyQueueException(this));  
    else { data=m_FirstNode.getData();  
        m_FirstNode = m_FirstNode.getNext();  
        return data;}  
}
```



- 捕捉异常 P254

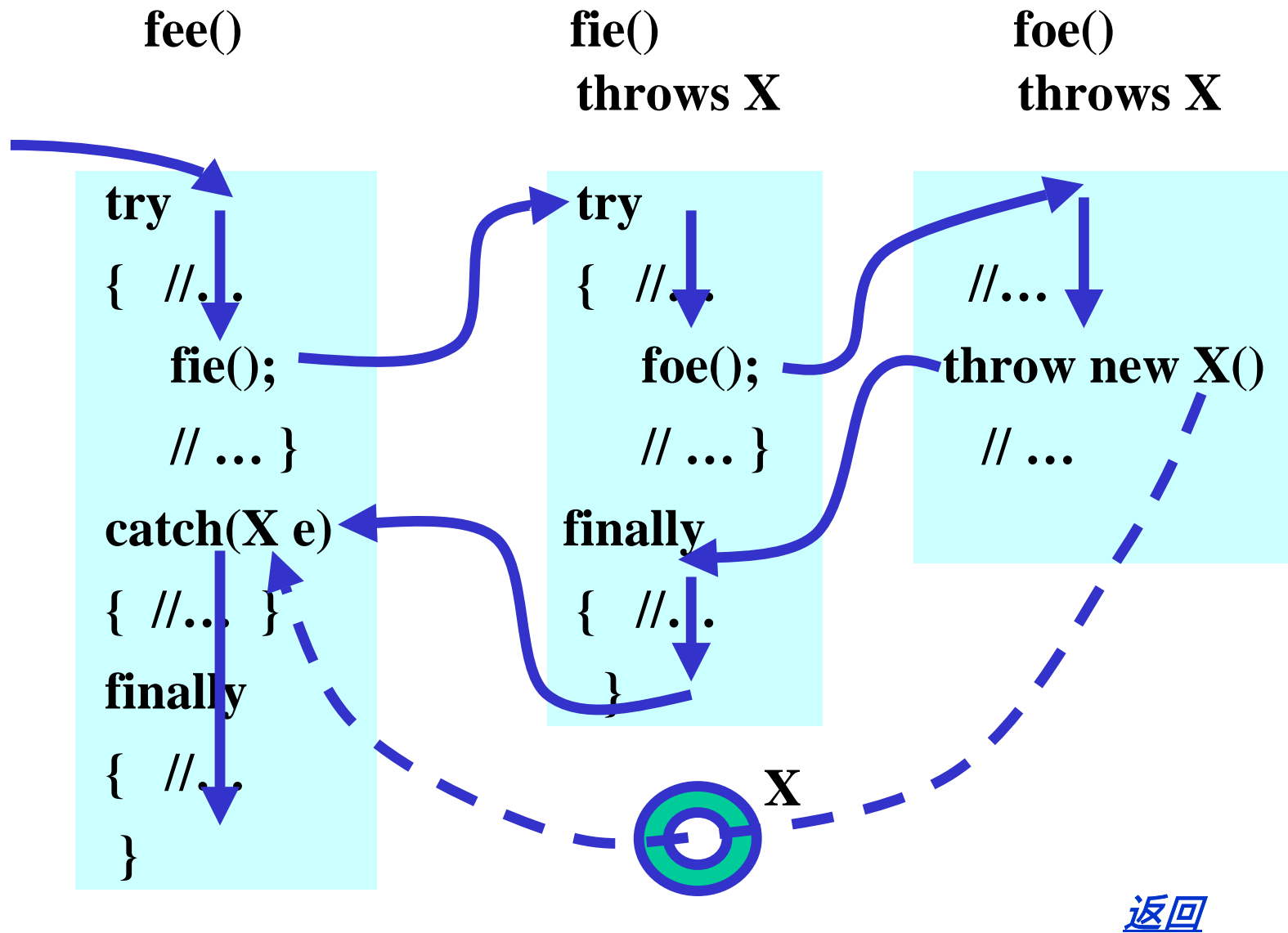
```
try    {    <try-block>  }  
    catch (exception_type id)  { <handler>}  
    catch (exception_type id)  { <handler>}  
    .....  
    finally { <finally-block> }
```

- [示例1](#)

- 多异常的处理 P255-256

- [示例2](#)





```
class SuperException extends Exception { }  
class SubException extends SuperException { }  
class BadCatch {  
    public void goodTry() {  
        try { throw new SubException();  
        } catch (SuperException superRef) { ...  
        } catch (SubException subRef) {...// 永远不会执行  
        } // catch 顺序不合理，通常子类异常放在父类  
        异常前面 }  
    }  
}
```

[返回](#)

异常处理

```
try { ... }
```

```
catch { ... }
```

```
finally { ... }
```

抛出异常

```
public boolean myM(int x)  
    throws AnException {  
    ...  
    throw new AnException();  
}
```

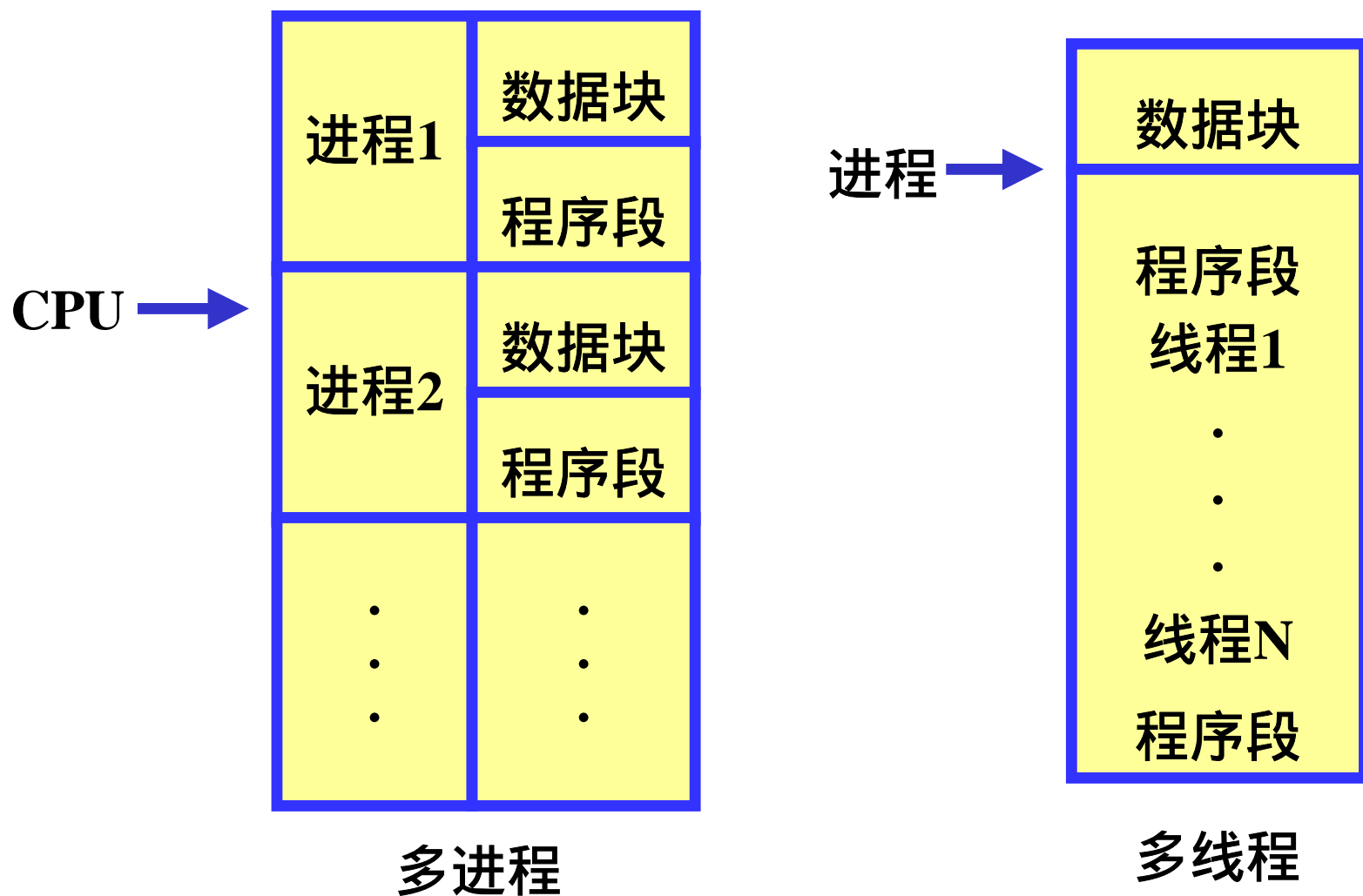
建立异常类

```
public class A extends  
    Exception {  
    ...  
}
```



异常传播

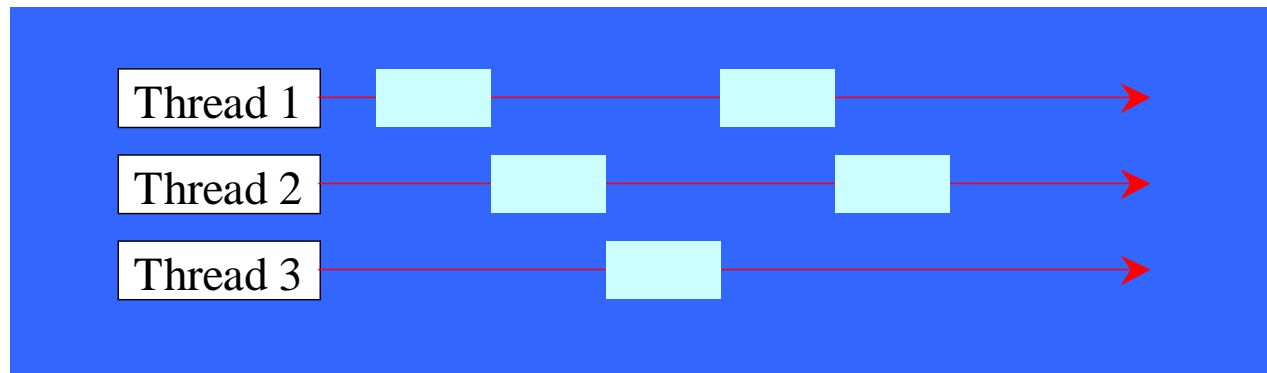


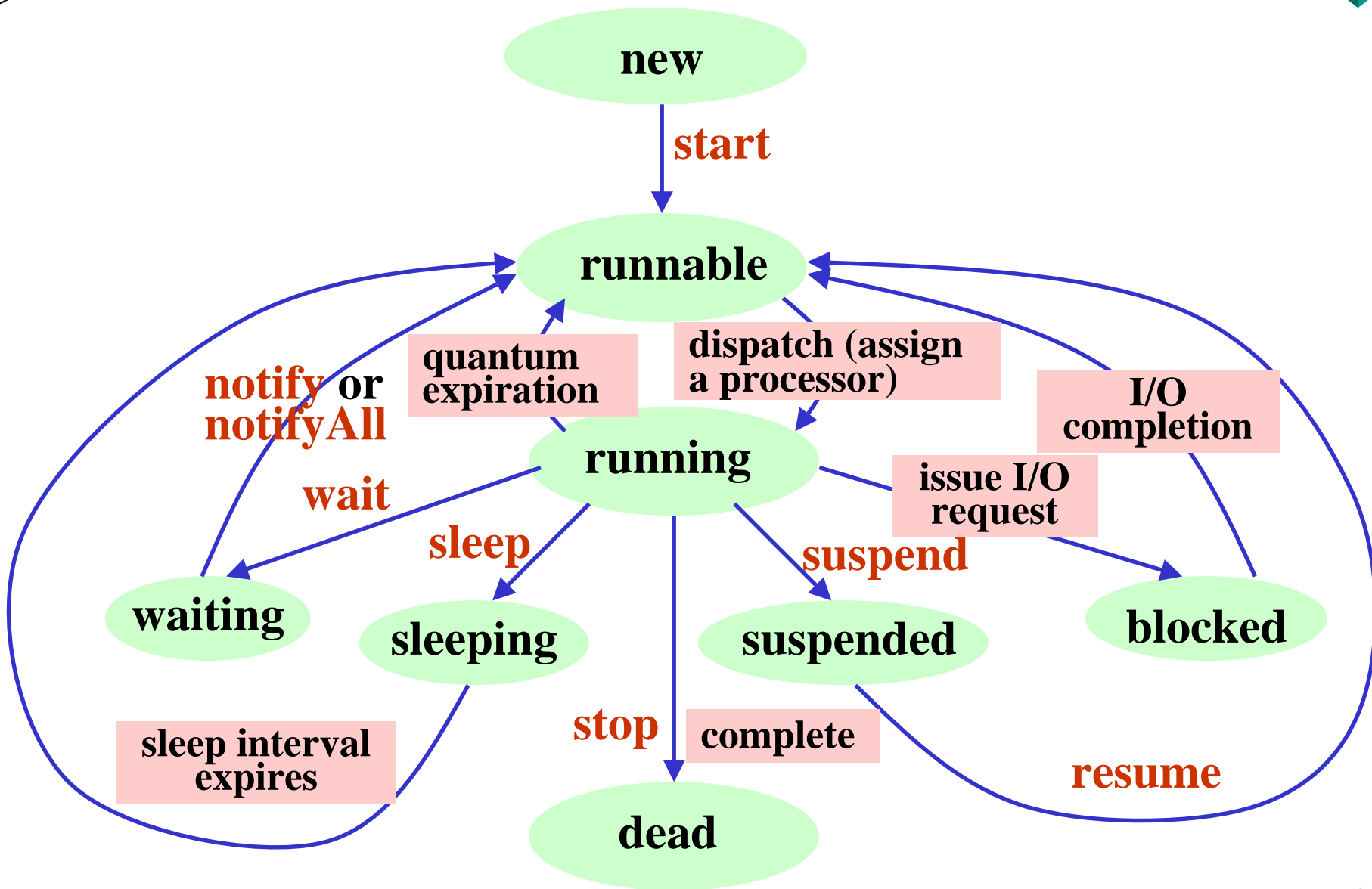


在多
CPUs
上的多
线程



在单
CPU
上的多
线程





(1) Thread类中有关优先级的常量与方法

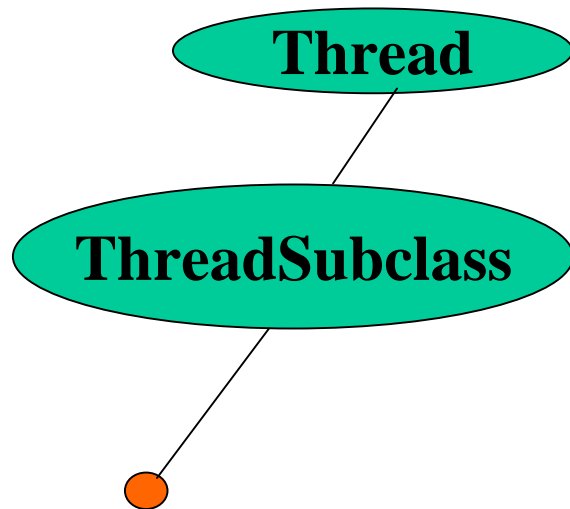
- **MAX- (10), MIN- (1) ,NORM-(5, default)**
- **public final void setPriority(int newPriority),**
newPriority 必须在 MIN- 和 MAX之间
- **public final int getPriority()**

(2) 线程调度

- 在当前可运行线程中选优先级别最高的运行
- 相同优先级问题



- Java中编程实现多线程有两种途径
 - 建立Thread 类的某个子类的对象

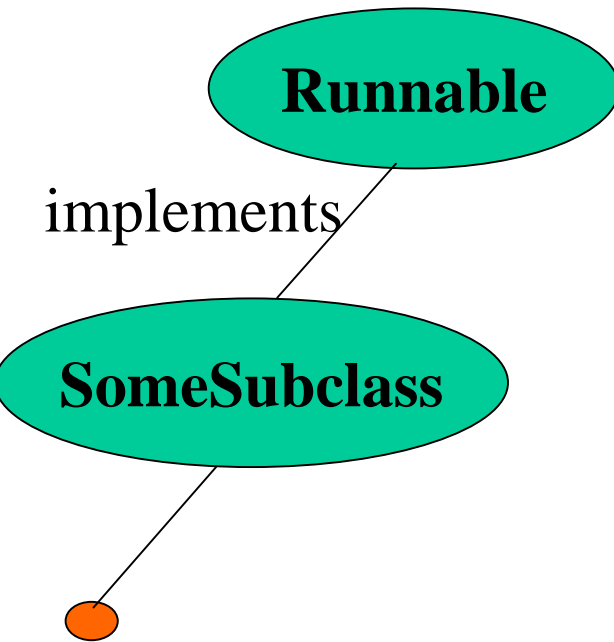


```
class ThreadX extends Thread {  
    public void run() {  
        //logic for the thread  
    }  
}
```

```
ThreadX tx = new ThreadX();  
tx.start();
```



➤ 创建实现 Runnable 接口的某个类的对象



```
class RunnableY implements Runnable {  
    public void run() {  
        //logic for the thread  
    }  
}
```

```
RunnableY ry = new RunnableY();  
Thread ty = new Thread(ry);  
ty.start();
```



•构造函数

➤ Thread()、Thread(String n)、Thread(Runnable target)、Thread(Runnable target , String n)

静态方法:

```
activeCount();  
currentThread();  
sleep();  
yield();
```

实例方法:

```
getPriority();  
setPriority();  
start();  
stop();  
run();  
isActive();  
suspend();  
resume();  
join();
```



(1) 创建 Thread 类的子类方法：P259-261

- (a) 实现 run() 方法;
- (b) 创建该类的一个对象 (线程);
- (c) 启动 (Start) 线程

```
class PingPong extends Thread {
    String word; int delay;
    PingPong(String whatToSay,int delayTime) {
        word=whatToSay; delay=delayTime; }
    public void run() {
        try { for(;;) { System.out.print(word + " ")
            sleep(delay); }
        } catch (InterruptedException e) { return; }}
    public static void main(String[] args) {
        new PingPong("ping",33).start(); // 1/30s
        new PingPong("PONG",100).start(); } // 1/10s
    } // Thread.run cannot throw exceptions, so must catch the exceptions of sleep
```

```
pingPONGping
pingPONGping
pingpingPONG
pingpingPONG
pingpingping
PONGpingping
PONGpingping
pingPONG...
```

(2) 实现 Runnable 接口方法：P262-264

- (a)实现 run()方法;
- (b)建立该类的一个对象;
- (c) 通过该对象建立一个 线程;
- (d) 启动该线程

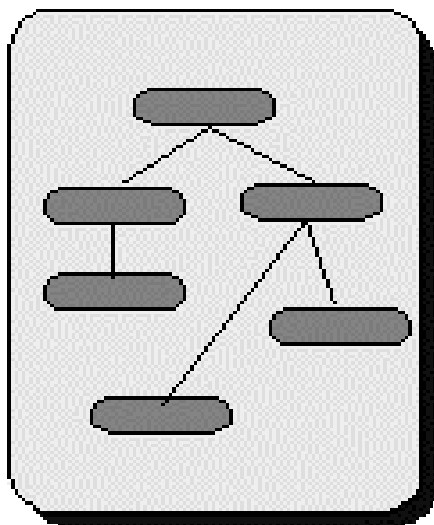
```
class RunPingPong implements Runnable{
    String word;    int delay;
    RunPingPong(String whatToSay,int delayTime){
        word=whatToSay; delay=delayTime;    }
    public void run(){
        try{    for(;;) { System.out.print(word+ " " );
                    Thread.sleep(delay); }
        }catch(InterruptedException e){ return; } }
    public static void main(String[] args){
        Runnable ping=new RunPingPong( " ping " ,33);
        Runnable pong=new RunPingPong("PONG " ,100);
        new Thread(ping).start();
        new Thread(pong).start(); } }
```



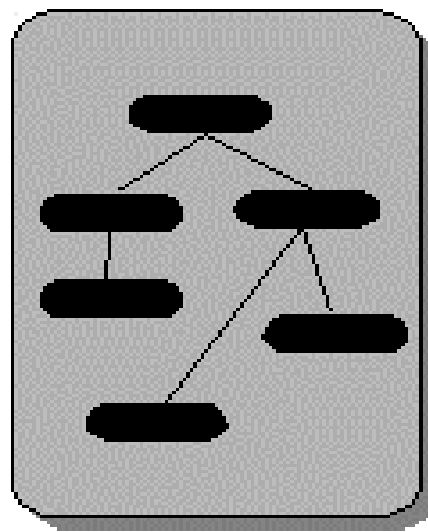
• 标准 Java包

- 为 Input/Output定义类
- 通用性
 - 支持控制台、缓冲区、文件、网络连接 ...
 - 支持原始方式的 IO和格式化的IO...
- 包括字节流系列和字符流系列（Java1.1）

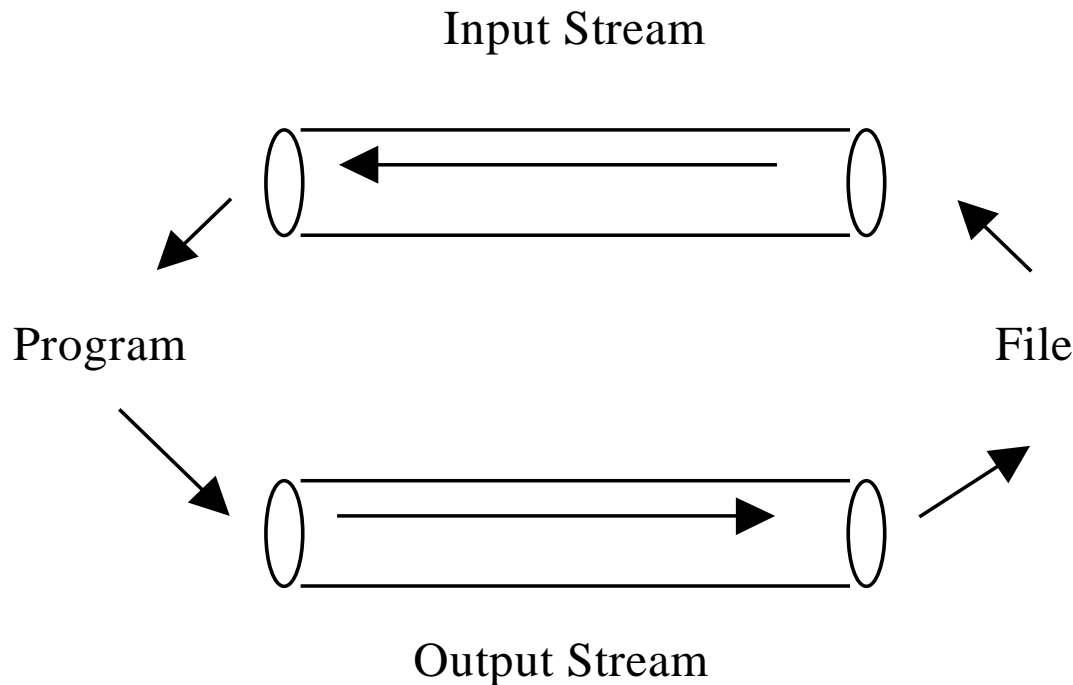
Character Streams



Byte Streams



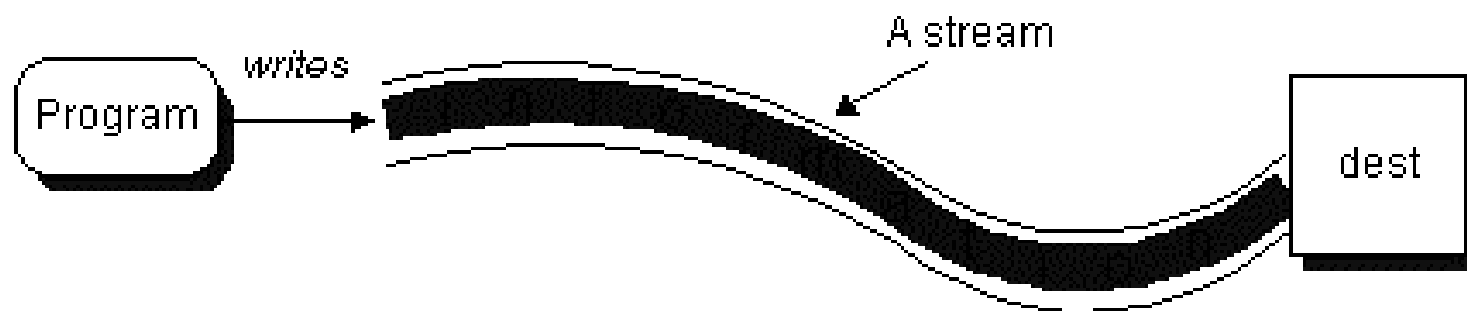
- 流是指在计算机的输入与输出之间运动的数据序列
 - 数据的获取或发送是沿数据序列顺序进行
 - 流中的数据既可以是未加工的原始二进制数据，也可以是经一定编码处理后符合某种格式规定的特定数据



•输入流

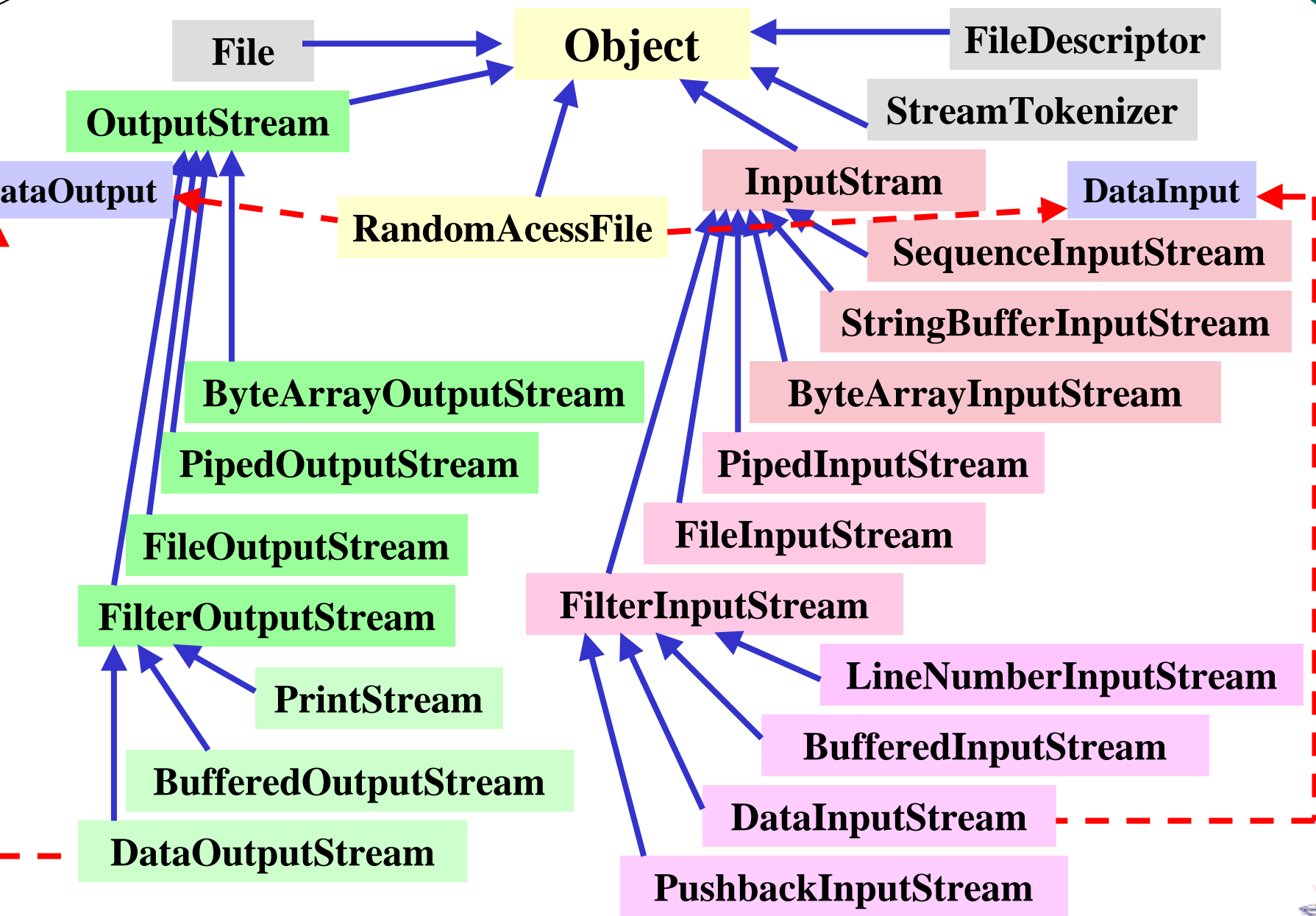


•输出流



- **InputStream 和 OutputStream 是所有字节流（8位字节）的抽象类**
 - **InputStream 中包括一套所有输入流都需要的方法，可以完成最基本的从输入流读入数据的功能**
 - **OutputStream 中包括一套所有输出流都需要的方法，可以通过向输出流中写数据来完成向外设输出数据**





- 所有 java 程序都可以使用 *System* 类中定义的三个流对象
 - *System.in*
 - 是一个 *java.io.BufferedInputStream* 对象
 - 该对象对键盘输入进行封装
 - *System.out*
 - 是一个 *java.io.PrintStream* 对象
 - 该对象对通过命令行输出到屏幕进行封装
 - 可以将输出重定义到一个文件
 - *System.err*
 - 是一个 *java.io.PrintStream* 对象
 - *System.err* 可以和 *System.out* 混合输出



- 标准输入： P277-278
 - 从键盘读入数据： `read()`
 - `System.in.read ()` 必须包含在 `try` 块
 - 读入的是二进制数据（8字节），合成后返回16位的整型量
 - 在键盘缓冲区中没有未被读取数据时，`read`方法将导致系统转入阻塞（`block`）状态
- 标准输出： P278
 - 在屏幕上输出数据： `print()`和`println()`
 - 可输出多种不同类型的变量或对象
- 示例： P278-278



InputStream: P274-275

- **abstract int read() throws IOException**
- **int read(byte b[]) throws IOException**
- **void close() throws IOException**
- **void available() throws IOException**
- **long skip(long n) throws IOException**

OutputStream: P275-276

- **abstract void write(int b) throws IOException**
- **void write(byte[] b) throws IOException**
- **void close() throws IOException**
- **void flush() throws IOException**



```
import java.io.*;    //计算文件中的字符数和空白字符数
class CountSpace{
    public static void main(String[] args) throws IOException
    {    InputStream in;
        if (args.length==0)    in=system.in; //从标准键盘输入
        else in=new FileInputStream(args[0]); //从命令行输入
        int ch;    int total;    int spaces=0;
        for(total=0;(ch=in.read())!=-1;total++){//每次读入一个
                                                    byte , 并合成一个char
            if (Charater.isSpace((char)ch) spaces++;
                }
            System.out.println(total+"chars,"+spaces+"spaces");
        }
    }
```




```
import java.io.*;//将输入的数据拷贝输出，并转化特定字符

class Translate{

    public static void main(String[] args){

        InputStream in=System.in;
        OutputStream out=System.out;
        if (args.length!=2)
            error("from/to needed");//没有配对的from/to

        String from=args[0], to=args[1];
        int ch , i;
        if (from.length()!=to.length())//from/to 长度不相等
            error("same length needed");
```



```
try{ while((ch=in.read())!=-1){  
        if ((i=from.indexOf(ch))!=-1)  
            out.write(to.charAt(i));  
        else out.write(ch);  
    }  
}catch(IOException e) { error("I/O Exception:"+e); }  
  
public static void error(String err){  
    System.err.print("Translate:"+err);  
    System.exit(1); //非零值意指‘不好’  
}  
}
```



- 过滤输入输出流是在输入输出数据的同时能对所传输的数据做指定类型或格式的转换，即可实现对二进制字节数据的理解和编码转换
 - 文件输入输出流：FileInputStream & FileOutputStream
负责完成对本地磁盘文件的顺序读写操作
 - 管道输入输出流：PipedInputStream & PipedOutputStream
负责实现程序内部的线程间通信或不同程序间的通信
 - 字节流：ByteArrayInputStream & ByteArrayOutputStream
可实现与内存缓冲区的同步读写，以及对CPU寄存器的读写操作
 - 顺序输入流：SequenceInputStream可以把两个其他的输入流首尾相接，合并成一个完整的输入流



- 数据输入输出流 `DataInputStream` 和 `DataOutputStream` 分别实现了 `DataInput` 和 `DataOutput` 接口
- 接口 `DataInput` 和 `DataOutput` 定义了独立于具体机器的带格式的读写操作，从而实现了对不同类型数据的读写



- 对于不同类型数据的读写方法：

Read	Write	Type
readBoolean	writeBoolean	boolean
readChar	writeChar	char
readByte	writeByte	byte
readShort	writeShort	short
readInt	writeInt	int
readLong	writeLong	long
readFloat	writeFloat	float
readDouble	writeDouble	double
readUTF	writeUTF	String(UTF format)



```
int account ;
String name;
double balance;
try { input=new DataInputStream(
        new FileInputStream("client.dat"))
} catch (IOException e) {
    System.err.println("Fail to open the file"+e.toString());
    System.exit(1); }
try {  account = input.readInt();
        name = input.readUTF();
        balance = input.readDouble();
} catch (EOFException eof) { ...}
catch (IOException e) {...}
```



- 在Java中，用File类实现磁盘文件和目录的管理
- 每个File类对象表示一个磁盘文件或目录，其对象属性中包含了文件或目录的相关信息，调用它的方法可以完成对文件或目录的常规操作
- 需要从磁盘文件读取数据或将数据写入文件，必须使用文件输入输出流类：FileInputStream 或FileOutputStream
- FileInputStream 或FileOutputStream实现的是对磁盘文件的顺序读写，而且读和写要分别创建不同的对象
- 在Java中可以通过RandomAccessFile类实现对文件的随机读写操作



(1) 创建File类的对象 P279-280

- **public File(String path)**
- **public File(String dirName,String name)**
File(dirName + File.separator +name)
- **public File(File fileDir,String name)**
File(fileDir.getPath(), name)

Example: **File f1=new File("/");**
 File f2=new File("/", "autoexec.bat");
 File f3=new File(f1,"autoexec.bat");

(2) 获取文件或目录属性 P280-281

(3) Example P281-282



- **FileInputStream 和 FileOutputStream**
- **创建文件输入或输出流 :P284**
- **从文件输入输出流中读写数据: P284**
- **Example: P283-284**



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

class FileInputDemo {

    public static void main(String args[]) throws
Exception{

    int size;

    InputStream f1=new FileInputStream("file.htm");

    size=f1.available(); // total size

    // Read the first 1/4 by read()

    for (int i=0; i<size/4; i++)

        System.out.print((char)f1.read());

    System.out.println("Still available: "+f1.available());
```



```
// Read the next 1/8 by read(b[])
```

```
byte b[] = new byte[size/8];
```

```
if (f1.read(b)!=b.length)
```

```
    System.err.println("Something bad happened!");
```

```
System.out.println("Still available: "+f1.available());
```

```
// Skip another 1/4: skip()
```

```
f1.skip(size/4);
```

```
System.out.println("Still available: "+f1.available());
```

```
f1.close();
```

```
}
```

```
}
```



```
import java.io.*  
  
class FileOutputDemo {  
    public static byte getInput()[] throws Exception {  
        byte buffer[] = new byte[12];  
        for (int i=0;i<12;i++)  
            buffer[i]=(byte) System.in.read();  
        return buffer;  
    }  
}
```





```
public static void main(String args[]) throws Exception
{
    byte buf[]=getInput();
    OutputStream f0=new FileOutputStream("file1.txt");
    OutputStream f1=new FileOutputStream("file2.txt");
    OutputStream f2=new FileOutputStream("file3.txt");
    for (int i=0;i<12;i+=2) f0.write(buf[i]);
    f0.close();
    f1.write(buf);
    f1.close();
    f2.write(buf,12/4, 12/2);
    f2.close(); }
}
```



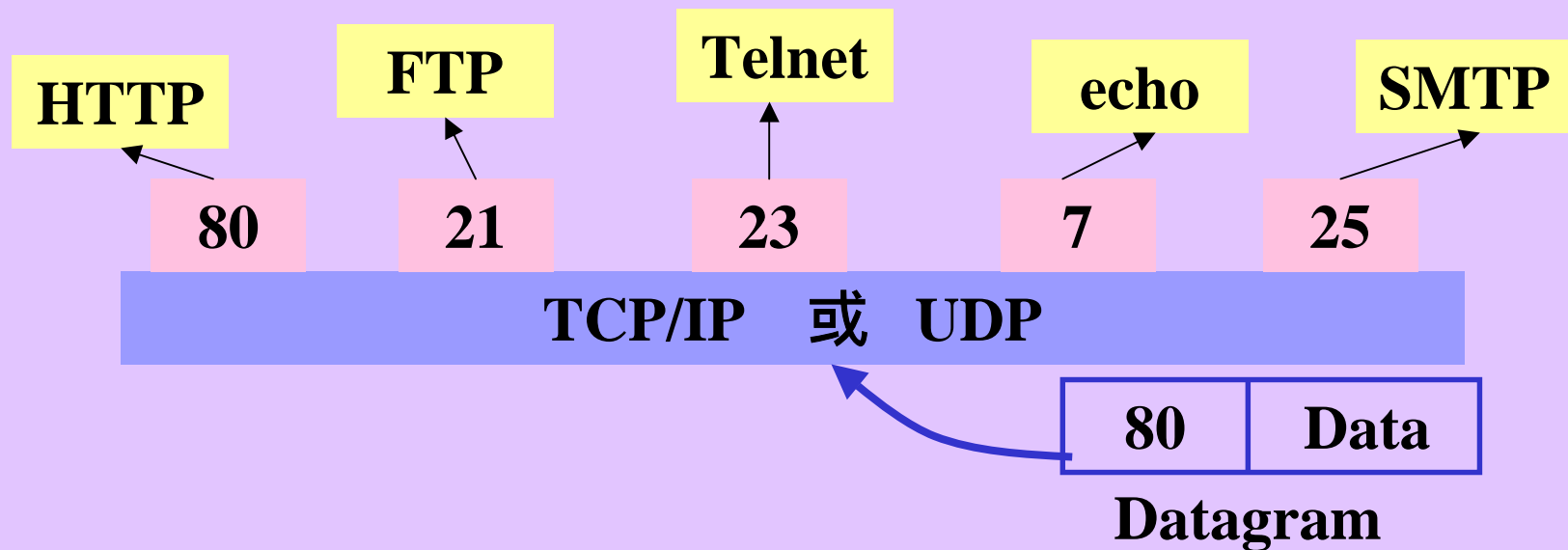
- 随机读写是指可以在文件中设置读写文件指针并执行相应的操作
- RandomAccessFile 类实现 *DataInput* 和 *DataOutput* 接口, 所以可以支持数据格式的读写操作
- 创建RandomAccessFile 对象 : P285
- 对文件位置指针的操作 : P285-286
- Example : P286-287



Telnet	FTP	HTTP	SMTP	...	 应用层 
套接字 Socket					
TCP/UDP					传输层
IP/ICMP					网络互联层
数据链路层与物理层					低层

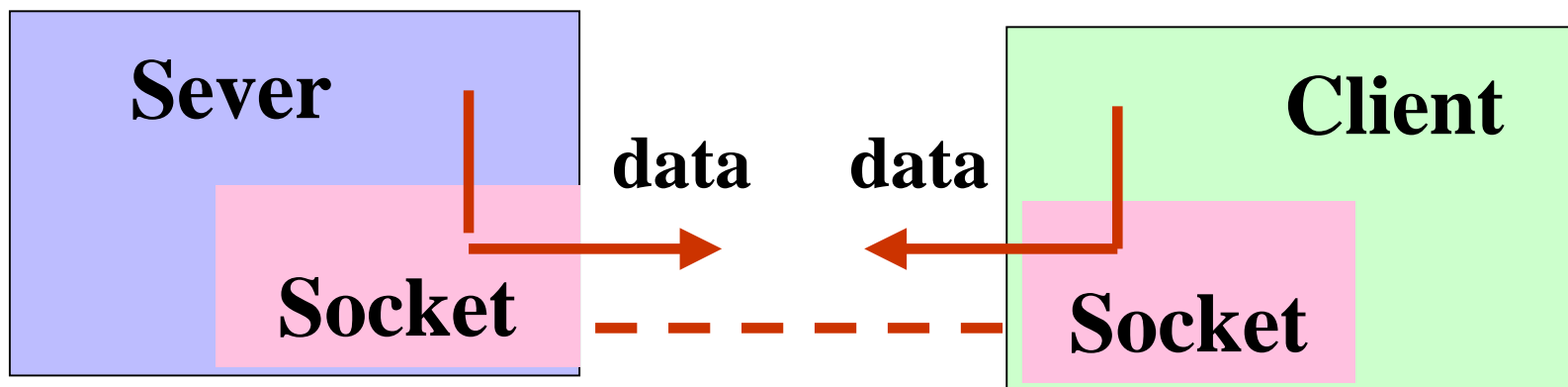


端口号: 在TCP/IP 系统中的16 位数字 (范围 0-65535). 实际上, 低于1024 的端口号保留用于特定的服务

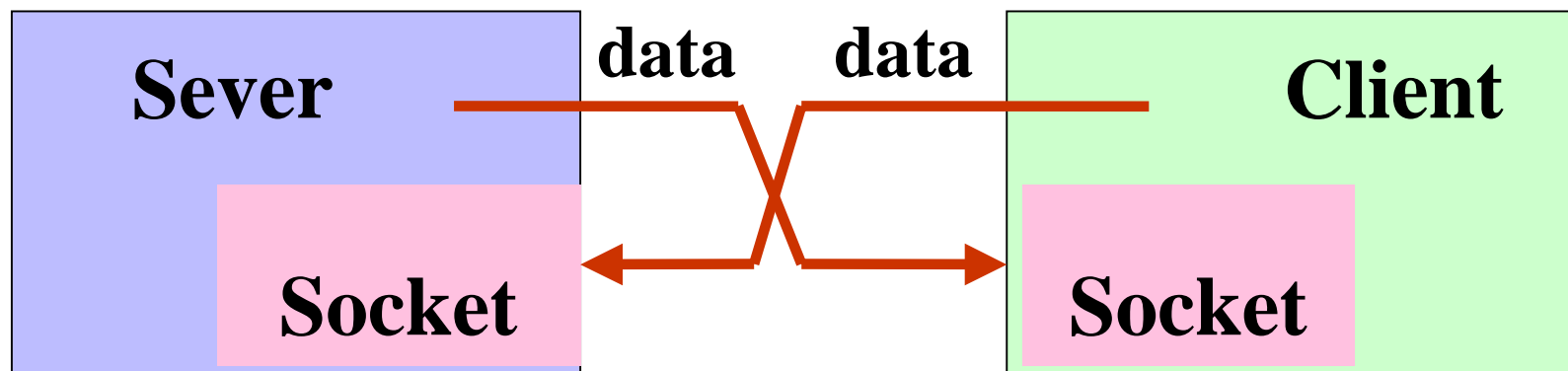


- 套接字不是一个硬件概念，是TCP/IP的编程接口，套接字包含主机地址和服务端口号
- 当在网络上进行通信时，Java 使用流模式. 一个套接字有两个流, 一个是输入流另一个是输出流
- 一个程序通过网络发送数据到另一个程序可以简单地通过向与套接字相连的输出流里写数据
- 程序读由网络另一端发送的数据只需要简单地从与套接字相连的输入流中读取数据





面向连接的Client/Server通信



面向无连接的Client/Server通信

•java.net 提供了如下几方面的类来帮助用户完成网络操作：

- 网络地址转换：完成域名与IP地址的转换
- 面向连接的通信：为服务器与客户程序提供类库支持
- 面向无连接的通信：提供数据报方式通信所需的类库支持
- Web连接：为浏览器-服务器模式的较高层次的连接提供类库的支持



- 确定主机

- 为了在IP网络上与其他主机进行通信，我们首先必须知道主机的IP地址

- Java的 `InetAddress`类用于代表 IP 地址

- 没有显式的构造器，通过静态方法得到：

`getLocalHost`, `getByName`, `getAllByName`

- Example

```
InetAddress Address=InetAddress.getLocalHost();  
System.out.println(Address); //harvard/202.119.36.48  
InetAddressAddress=InetAddress.getByName("harvard");  
System.out.println(Address);
```

- 例如： P289

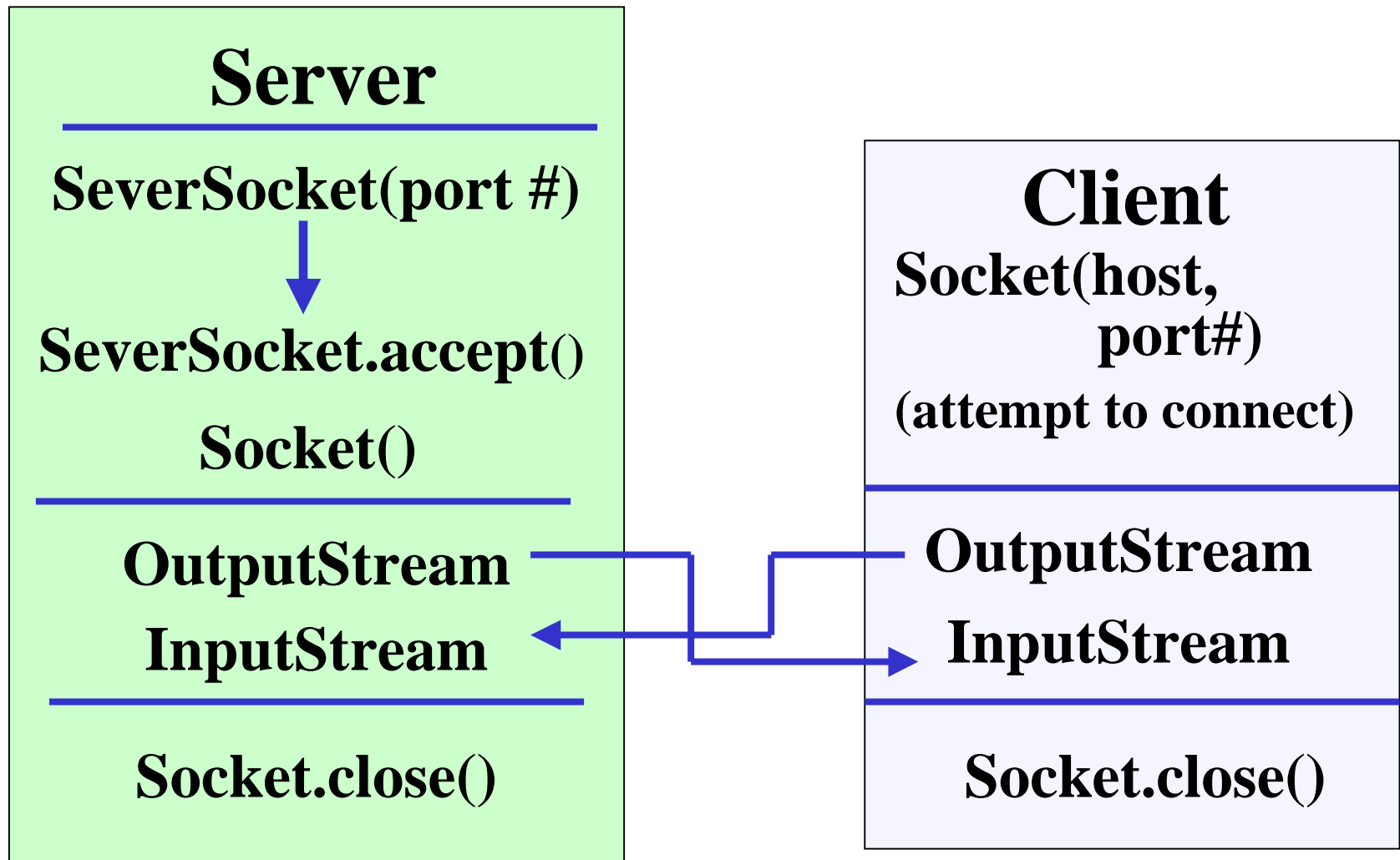


- 两个类
 - 类Socket, 实现客户端 socket 操作
 - 类ServerSocket, 实现服务器 socket 操作
- 客户端编程
 - 建立一个新的 socket
 - 为这个新的socket 建立输入输出流
 - 根据协议对socket 进行读写
 - 关闭 socket 和输入输出流



- **服务器端编程**
 - 建立一个服务器 socket 和一个通信用 socket
 - 允许服务器 socket 侦听
 - 为通信用的 socket 建立输入输出流
 - 从输入流读数据或向输出流写数据
 - 关闭所有对象
- **Example :P291-296**





- 建立 sockets : P291

`Socket(InetAddress address, int port);`

`Socket(InetAddress address, int port, boolean stream);`

`Socket(String host, int port);`

`Socket(String host, int port, boolean stream);`

stream: stream sockets or datagram sockets?

`ServerSocket(int port);` // count: maximum of connections

`ServerSocket(int port, int count);`

Client end

```
try { Socket socket = new Socket("harvard",4700);  
    } catch (Exception e) { System.out.println("...");}
```



Server end

```
ServerSocket server=null;  
try { server = new ServerSocket(4700);  
} catch (Exception e) { System.out.println("...");}  
Socket socket=null;  
try { socket=server.accept();  
} catch (Exception e){ System.out.println("...");}
```

•打开输入输出流

—getInputStream/getOutputStream in the Socket class

—PrintStream os=new PrintStream(new
BufferedOutputStream (socket.getOutputStream()));

—DataInputStream is=new DataInputStream(new
socket.getInput Stream());



- 关闭Sockets
 - `os.close`; `is.close`; `socket.close`
 - 输入输出流必须在 `sockets` 之前关闭
- 服务器/客户端编程示例

```
import java.io.*;
import java.net.*;
class NotebookServer {
public static void main(java.lang.String[] args) {
    try { ServerSocket server=new ServerSocket(4700);
        Socket socket=server.accept();
        DataInputStream in = new
            DataInputStream(socket.getInputStream());
        PrintStream out = new PrintStream(socket.getOutputStream());
        out.println("Hello! Welcome you to use our notebook.\r");
        out.println("You can enter bye to exit. \r");
```



```
boolean done=false;
while(!done) { String s=in.readLine();
    if (s==null) done=true; //将信息返回给客户机
    else { out.println('Echo:'+s+'\r');
//将信息显示在服务器中    System.out.println(s+'\r');
    if (s.trim().equals("bye"))
//客户输入bye则退出        done=true; }
    }
    socket.close(); }
    catch (Exception e) {
        System.out.println(e); }
}
}
```



```
import java.applet.*; import java.awt.*;
import java.net.*; import java.io.*;
public class NotebookClient extends Applet {
    TextArea ta;    ClientTextField tf; Socket socket;
    DataInputStream in;    PrintStream out;
    public void init() {
        setLayout(new BorderLayout());
        ta = new TextArea(5,40);
        ta.setEditable(false);
        add("Center",ta);
        tf=new ClientTextField(40);
        add("South",tf);
    }
}
```



```
public void send() {  
    String s=tf.getText();  out.println(s+"\r");  
    tf.setText(""); }  
public void start() {  
    try { socket = new Socket("202.119.36.237",4700);  
        in=new DataInputStream(socket.getInputStream());  
        out=new PrintStream(socket.getOutputStream());      }  
        catch (IOException e) {      }  
    Receiver r= new Receiver(in, ta);  
    r.start(); }  
public void stop() {  
    try {  socket.close(); }  
    catch (IOException e) {      } }  
}
```



- 两个类：P296-297

- **DatagramPacket**：由 **DatagramSocket** 传送的信息对象

- * **Public DatagramPacket(byte ibuf[], int ilength) //receive**

- * **Public DatagramPacket(byte ibuf[], int ilength, *InetAddress iaddr, int iport*) //send data ***

- **DatagramSocket**：在应用程序间进行通信

- * **Public DatagramSocket() throws SocketException**

- * **Public DatagramSocket(int port) throws
SocketException**

- * **Public DatagramSocket(int port , *InetAddress laddr*) throws SocketException**



– 客户端编程

- 建立一个新的 DatagramSocket
- 获得 internet 地址和通信端口
- 建立一个在DatagramSocket上的新的 DatagramPacket, 网络地址和端口, 并且能发送它
- 新的 DatagramPacket 也可以 接受

– 服务器端编程

- 首先, 启动一个服务器端线程
- 然后建立一个新的 DatagramSocket, 并且使用一个随机的端口
- 建立一个新的 DatagramPacket 并且能接受和发送数据
- 关闭所有的Socket

• Example P298-301



```
import java.net.*; //Sender.java
import java.io.*;
Public class Sender {
Public static void main (String args[]) throws IOException {
    if (args.length != 2) { //判断命令行参数是否符合要求
        System.out.println("Usage:java Sender <dest hostname> <port>");
        return; } //发送缓冲区
    byte sBuf[] = new byte[100];
    System.out.println("Give the message you'll send,"
        + "up to 100 bytes, ending with #");
    DataInputStream dataIn = new DataInputStream(System.in);
    int i;
```




```
for (i = 0; i < 100; i++) {//读取键盘输入，存入缓冲区
```

```
    byte inByte = dataIn.readByte();
```

```
    if ((char)inByte == '#') break;
```

```
    sBuf[i] = inByte; } //创建数据报套接字
```

```
DatagramSocket sendSocket = new DatagramSocket();
```

```
//创建一个数据报
```

```
DatagramPacket packet = new DatagramPacket(
```

```
    sBuf, i, InetAddress.getByName(args[0]),
```

```
    Integer.valueOf(args[1].intValue() );
```

```
sendSocket.send(packet) ; //发送
```

```
sendSocket.close(); //关闭 }
```

```
}
```



```
import java.net.*;    //Receiver.java
import java.io.*;

Public class Receiver {

Public static void main(String args[]) throws
IOException {
    if (args.length != 1) {
        System.out.println("Usage:java Receiver <port>");
        retuen; } //接收缓冲区

    byte rBuf[] = new byte[100];
    //创建一个用于接收数据的数据报
    DatagramPacket packet =
        new DatagramPacket(rBuf, rBuf.length);
```



//创建套接字

DatagramSocket receiveSocket = new

DatagramSocket(Integer.valueOf(args[0]).intValue());

//等待并接收数据报

receiveSocket.receive(packet);

//取数据报中的数据并打印

System.out.println(new String(packet.getData()));

//关闭套接字

receiveSocket.close(); }

}



- URL类的对象表示一个URL地址，利用这个地址可以访问远程的资源
- URL 由两部分组成：一部分是协议名字，如 http, ftp, gopher 和 news 等；另一部分是资源名，它由四个部分组成：
 - 主机名或 IP 地址
 - 端口号
 - 文件名
 - 文件内部的一个引用
- 建立URL 对象必须在 try-catch 块



- 绝对地址 :P301
 - `URL ai=new URL`
(“`http://aiake1.nju.edu.cn/html/people/teacher.htm`”)
 - `URL ai=new URL (http, aiake1.nju.edu.cn,`
`/html/people/teacher.htm)`
- 相对地址
 - `URL ai=new URL`
(“`http://aiake1.nju.edu.cn/`”)
 - `URL ai_teacher=new URL (ai,`
`“html/people/teacher.htm”)`



- 直接从 URL 读信息: P302

- *URL method:

- public final InputStream openStream()*

- *URL类可以简单方便地获取远程机器的信息

- 使用 URLConnection类: P303-306

- *URLConnection method:

- public URLConnection openConnection()*

- * URLConnection类可以支持向远方机器获取信息或传送信息



- 访问指定网页:P305-306
 - 方法 : Applet getAppletContext();
AppletContext showDocument()
 - Example: P305-306
- 获取指定URL处的图象
 - 方法 : Image getImage()
 - Example:P306-307
- 获取指定URL处的声音方法 :
 - 方法 : Applet getAudioClip(); play()
 - Example:P307-308

