ネットワークプログラミング
第9回
2014/11/18
tcp通信oos

岩井将行

授業資料

- http://www.cps.im.dendai.ac.jp

講師紹介

- http://cps.im.dendai.ac.jp/index.php?Members%2Fiwai

- 岩井研究室
- http://cps.im.dendai.ac.jp

- 岩井研究室の研究分野
- http://cps.im.dendai.ac.jp/index.php?Research%2FTopics

- 連絡先　1号館11F 11107b
- 　　iwaiあっと im.dendai

TA･SA･副手

- たじまっち
- みむらくん
- すぎおくん

成績

- 毎回取り組み姿勢（出席）
- 毎回課題（演習のみ）
- 中間試験 （座学）
- 最終試験（座学）
- 最終課題（演習のみ）

- ★演習は演習最終発表会を加味

配列の宣言

- 型　配列名[] = new 型[n];

```
int  tensu[] = new int[100];
```

型[]　配列名 = new 型[n];

```
int[]  tensu = new int[100];
```

0 ～ n-1 の n個の配列ができる。

配列の添字は0から始まる
配列 xData の大きさが3のとき、使えるのは、
xData[0]、xData[1] 、xData[2]。xData[3]は使えない。

配列の宣言

・ int mathScore[] = new int[5];
と宣言すると、

```
  mathScore[3] =  82;
for(int i = 0; i < 5; i++) {
    mathScore[i] = 10;
}
```

のような代入等が可能となる。

配列の問題

TwoArray.javaを改造して

// 点数の低いほうを表示するプログラムを作成せよ。

// 例： 0番さんは英語の方が点が低い

// 　　1番さんは数学の方が点が低い

// if 文の比較と、forループで作る

// ヒント String[] kamoku = {"数学", "英語"}; とすると、kamoku[0] とか kamoku[1] が使える

// 数学と英語の比較は for文を使わなくても if(score[i][0] > score[i][1]) といった感じで

// 比較が終わるTwoArrayMin.java

//1番目を一郎、2番目を二郎（ロッター）、３番目を三番艦、４番目をスシロー、5番目を吾郎メンバーとして名前を配列の最初に記録し、名前も出力させよ。TreeArray.java

配列の長さ

- 配列名.length
- 例えば、xData の長さは、xData.length で求められる。

配列の初期化

配列の型[]　配列　= {要素, 要素, 要素};

例

```
int[]  xData = {90, 85, 65};
```

インナークラスを理解しよう。

- ClassA.java

- FacesMain.javaをインタークラス化しよう。
- FacesMain.javaをインナークラスで書き換えよう。

FaceMain.java

- FacesMainをオブジェクトの配列で書き換えよう。

- FaceObjインナークラスを生成しよう。

MovingBall

Thread,Runnable

RandSwitch

配列

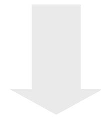
プロトコル
TCP/IP

トランスポートプロトコル

- トランスポート層はOSI7階層モデルのトランスポート層に位置している。ネットワーク層ではホスト同士の通信について定義していたが、信頼性は保証していない。
- また、送信した順序どおりにデータが届くという保証もない。そこで、トランスポート層ではホスト間で信頼のおける通信路を提供するための定義をおこなっている。
- トランスポート層のプロトコルには**TCP(Transmission Control Protocol)**と**UDP(User Datagram Protocol)**があげられる。
-

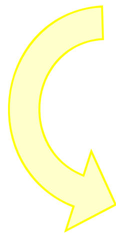
パケット通信

長いデータは通信できない



小さな複数の情報に分割

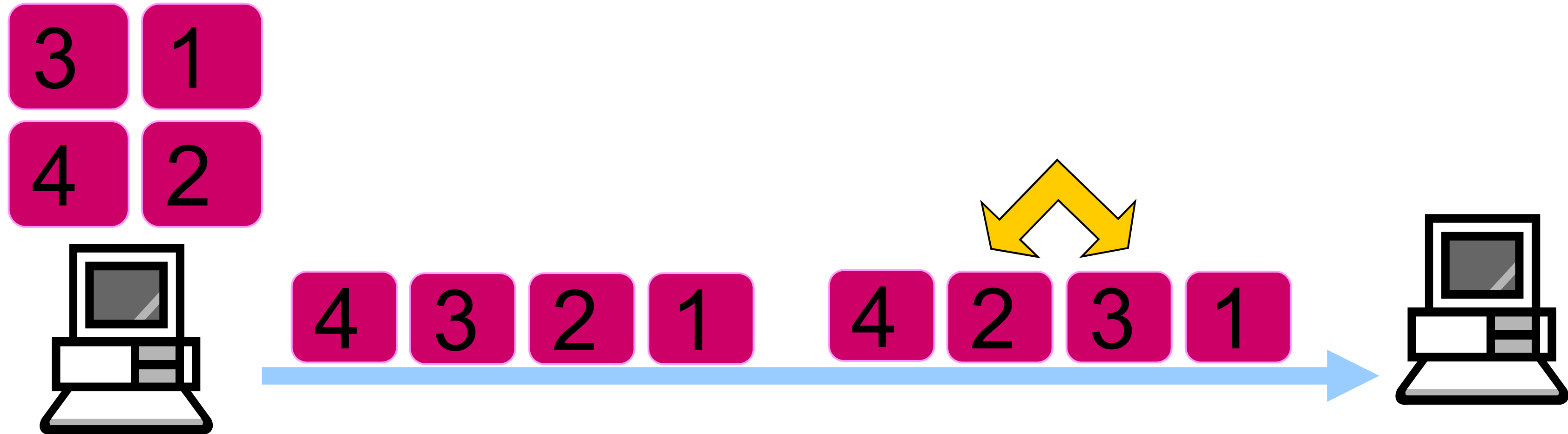
データM



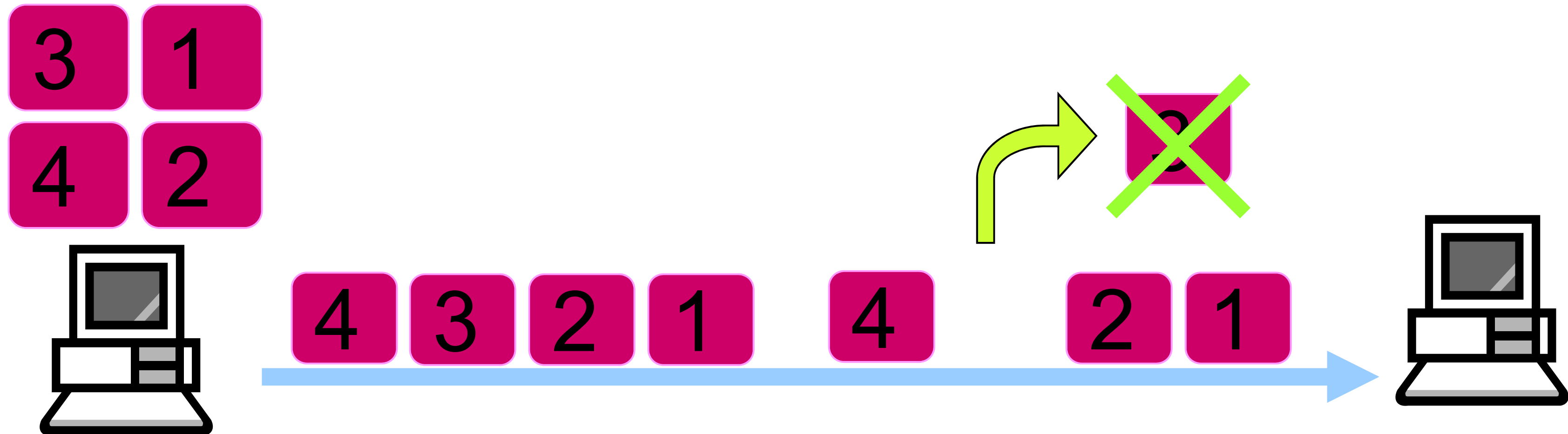
データm_1 データm_2 データm_3

パケット通信の問題点

- パケットの順番



- パケットの欠落



TCP

- TCPの主な特徴を以下に記す。
- パケットの破壊、重複、喪失、順序の入れ替わりをチェックする。また応答確認、再送信を行い、通信の信頼性を高める。
- ホスト間で通信する前に仮想的な通信路を作る（コネクション指向）。
- 始点と終点間で多重化を行うことで、同時に複数の通信が行うことができる。
- 送信のフローコントロール制御を行う。

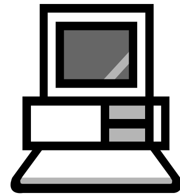
TCP

TCP : Transmission　Control Protocol

● インターネットの通信プロトコルとして最も普及

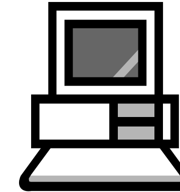
※プロトコル

×

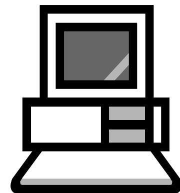


こんにちは。

Bonjour.

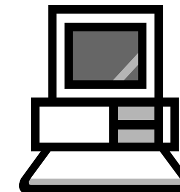


○



こんにちは。

こんにちは。



UDP

- UDPの主な特徴を以下に示す。
- 複雑な制御は提供せず、コネクションレスの通信を行う。
- ネットワークが混雑していたとしても、送信量を制御することは無い。
- パケットが失われても再送制御しない。
- 処理がTCPに比べて軽量なので高速に動作する。

TCPとUDP

- TCPとUDPの違い

	TCP	UDP
信頼性	高信頼	低信頼
転送速度	低速	高速
転送形式	コネクション型	コネクションレス型
その他	端末間同士のデータ転送	上位レイヤからの送信要求が簡潔

TCP v.s. UDP

- TCPはトランスポート層で信頼性のある通信を実現する必要がある場合に利用される。TCPはコネクション指向で順序制御や再送制御を行なうためアプリケーションに信頼性のある通信を提供することができる。

- UDPは高速性やリアルタイム性を重視する通信などに用いられる。
  - 例としてリアルタイムのストリーミングを挙げる。
  - もしTCPを利用した場合、パケットが途中で失われた場合に再送処理を行なうため、その間画像や音が停止するなどの不具合が生じてしまう。これに対して、UDPは再送処理を行なわないのでパケットは送信され続ける。もし多少のパケットが失われていたとしても、一時的に画像や音声が乱れるだけである。よって、ストリーム配信サービスではUDPの方が優れているといえる。

ソケット通信

プログラム同士の通信は

- ソケットを使ってデータの送受信
- ソケットを使った通信

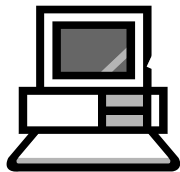
ソケット通信

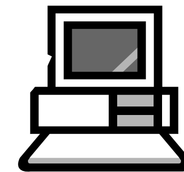
ソケット

意味：「接続の端点」

コンピュータとTCP/IPを
つなぐ出入り口

ソケット





ソケット通信

●ソケットを使って通信を行うには
　２つのプログラムが必要

クライアントプログラム

ソケットを用意して
サーバに接続要求を行う

サーバプログラム

ソケットを用意して接続要求を待つ

ソケット通信の全体の流れ

クライアント		サーバ
ソケット生成(socket)		ソケット生成(socket)
サーバを探す (gethostbyname)		接続の準備(bind)
		接続待機(listen)
接続要求(connect)	識別情報	接続受信(accept)
データ送受信(send/recv)		データ送受信(send/recv)
ソケットを閉じる(close)		ソケットを閉じる(close)

識別情報

- 正しくデータを受け渡しするために
  通信する相手を識別する

IPアドレス

コンピュータを識別

コンピュータのアドレス

ポート番号

プログラムを識別

プログラムの識別番号

ウェルノウン　ポート

よく使われているプログラムの
ポート番号は決まっている

ポート番号	プログラム
21	ftp
22	ssh
23	telnet
80	http(web)

1024番以下は全て決められている

クライアントプログラム（Java）

```
import java.io.*;
import java.net.*;
import java.lang.*;

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

        try{
            //ソケットを作成
            String host="localhost";
            Socket socket = new Socket( host, 10000 );

            //入力ストリームを作成
            DataInputStream is =  new DataInputStream
                        new BufferedInputStream(
                            socket.getInputStream()));
```

```
            //サーバ側から送信された文字列を受信
            byte[] buff = new byte[1024];
            int a = is.read(buff);
            System.out.write(buff, 0, a);

            //ストリーム，ソケットをクローズ
            is.close();
            socket.close();

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

サーバプログラム（Java）

```
//Server.java

import java.net.*;
import java.lang.*;
import java.io.*;

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

            try{
                //ソケットを作成
                ServerSocket svSocket = new ServerSocket(10000
                //クライアントからのコネクション要求受付
                Socket cliSocket = svSocket.accept();

                //出力ストリームを作成
                DataOutputStream os = new DataOutputStream(
                            new BufferedOutputStream(
                                cliSocket.getOutputStream()));

                //文字列を送信
                String s = new String("Hello World!!¥n");
                byte[] b = s.getBytes();
                os.write(b, 0, s.length());
```

```
//ストリーム，ソケットをクローズ
                os.close();
                cliSocket.close();
                svSocket.close();

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

サーバプログラム(Java)

ソケット作成，コネクション要求受付待機

```
ServerSocket svSocket =
  newServerSocket(10000);
Socket cliSocket = svSocket.accept();
```

出力ストリーム作成

```
DataOutputStream os =
  new DataOutputStream(
    new BufferedOutputStream(
      cliSocket.getOutputStream()));
```

クライアントプログラム（Java）

ソケットを作成

```
Socket socket = new Socket(
            "hoge.com", 10000 );
```

入力ストリームを作成

```
DataInputStream is =
  new DataInputStream (
    new BufferedInputStream(
      socket.getInputStream()));
```

クライアントプログラム（Java）

サーバ側から送信された文字列を受信

```
n = is.read(buff);
System.out.write(buff, 0, n);
```

ストリーム，ソケットをクローズ

```
is.close();
socket.close();
```

サーバプログラム（Java）

ソケット作成，コネクション要求受付待機

```
ServerSocket svSocket =
  newServerSocket(10000);
Socket cliSocket = svSocket.accept();
```

出力ストリーム作成

```
DataOutputStream os =
  new DataOutputStream(
    new BufferedOutputStream(
      cliSocket.getOutputStream()));
```

InetAddress

- http://docs.oracle.com/javase/jp/6/api/java/net/InetAddress.html
- **IPアドレスを扱うクラス**
- **java.net**
  クラス InetAddress
- java.lang.Object **java.net.InetAddress すべての実装されたインタフェース**
  :Serializable**直系の既知のサブクラス**
  :Inet4Address, Inet6Address

InetAddress

- StringgetHostName()
  この IP アドレスに対応するホスト名を取得します。
- static InetAddressgetByAddress(String host, byte[] addr)
  指定されたホスト名および IP アドレスに基づいて InetAddress を作成します。

- static InetAddressgetLocalHost()
  ローカルホストを返します。

InetAddress

- static InetAddress**getByName**(String host)
  指定されたホスト名を持つホストの IP アドレスを取得します。

- boolean**isReachable**(int timeout)
  そのアドレスに到達可能かどうかをテストします

- String**toString**()
  この IP アドレスを String に変換します。

Objectの配列

Objectの配列：FaceObjKadaiAns.java

- FaceObjAns**[]** fobjs = **new FaceObjAns[**<u>**9**</u>**];**

```
for (int j = 0; j < 3; j++) {//行
    yStart = j * 220 + 50;
    for (int i = 0; i < 3; i++) {//列
        xStart = i * 220 + 40;
        fobjs[i + 3 * j] = new FaceObjAns(xStart, yStart);
    }
}
```

描画

```
// 9個数の顔を書く
for (int i = 0; i < fobjs.length; i++) {
    fobjs[i].makeFace(g);
}
```

FaceObjectのコンストラクタ

```
class FaceObjAns {
// コンストラクタ
int h;
int w;
int xStart = 0;
int yStart = 0;
public FaceObjAns(int x, int y) {
    h = 200;
    w = 200;
    this.xStart = x;
    this.yStart = y;
}
// 個々にメソッドを追加
public void makeFace(Graphics g) {
    makeRim(g);
    makeEyes(g, 20);
    makeNose(g, 40);
    makeMouth(g, 80);
}
```

Interface

- 内容に抽象メソッドしか持たない**クラスのようなもの(バールのようなもの)**をインタフェースと呼びます。
- クラスと並んで、パッケージのメンバーとして存在します。
- インタフェースはクラスによって**実装(implements)** され、
- 実装クラスはインタフェースで宣言されてい抽象メソッドを実装します。

インタフェースの複数実装

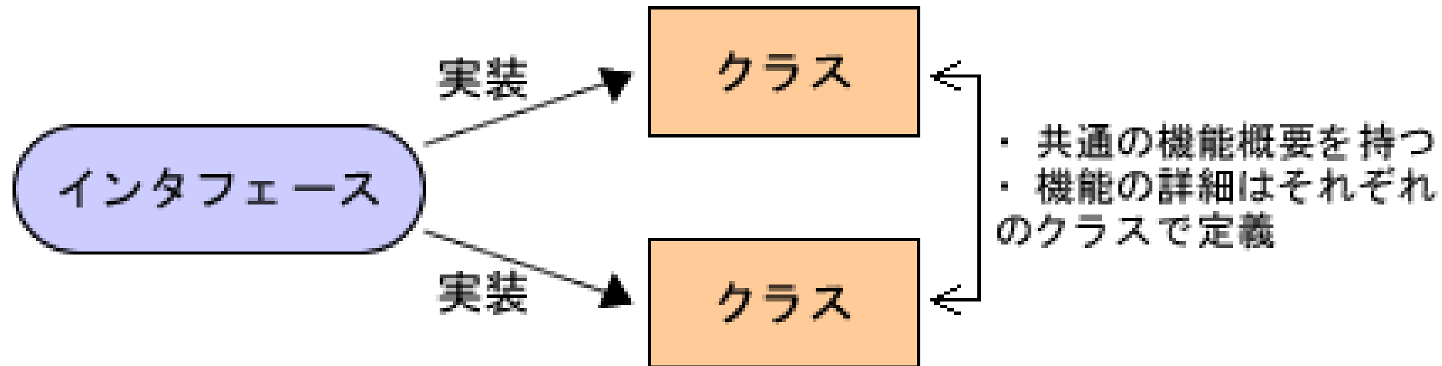
- クラスの場合は、単一のクラスしか継承 (extends) できせんが、インタフェースの場合は、複数のインタフェースを実装 (implements) することができます。

class InterfaceImpl ***implements*** Interface1, interface2, interface

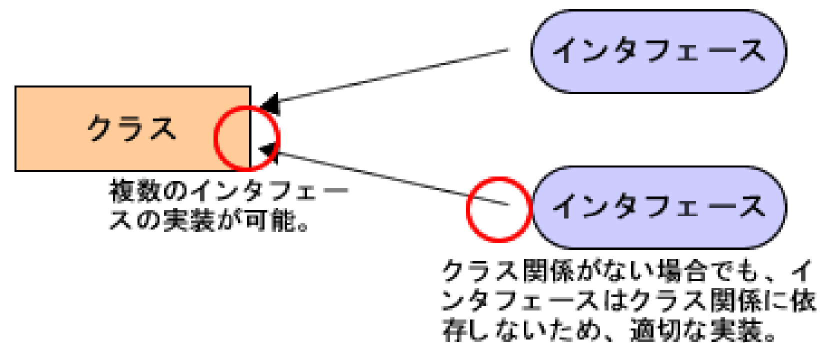
... }

interface の修飾子は public のみ。

インタフェースのメリット



- Javaは複数のスーパークラスを継承することはできません。Javaでは複数のスーパークラスの継承（多重継承）が認められていないため。



Plugin hybrid車は災害時にバッテリとして利用可能

ICar.javaインタフェース

- 車輪in getNumOfTiers()がある。
- スピートを設定する
  - void setSpeed (int sp)
  - int getSpeed()
  - void printCarName()

IElectricCharge.javaインタフェース

- void chargeBattery(int b)
- int getAllBattery()
- int consumeBattery(int b)

HybridCarImpl.java

- を実装してください。
- Yourには自分の名前を入れてください。
- 例　MasaHybridCarImpl.java

呼び出しのMainCall.javaを実装しよう。

- Hint
- MasaHybridCarImpl masaCar= new MasaHybridCarImpl();

- ICar car=(ICar) masaCar;
- car.setSpeed(); car.printCarName();
- IElectricCharge charger =(IElectricCharge) masaCar
- carger.chargeBattery(100);

Thread

Thread,Runnable

MovingBall

ThreadSleep　停止

- 300ミリ秒処理を停止する。

```
try{
  Thread.sleep(3000);
  //3000ミリ秒Sleepする
}catch(InterruptedException e){}
```

Threadの2種類の作りかた

- 1)**implements Runnable**

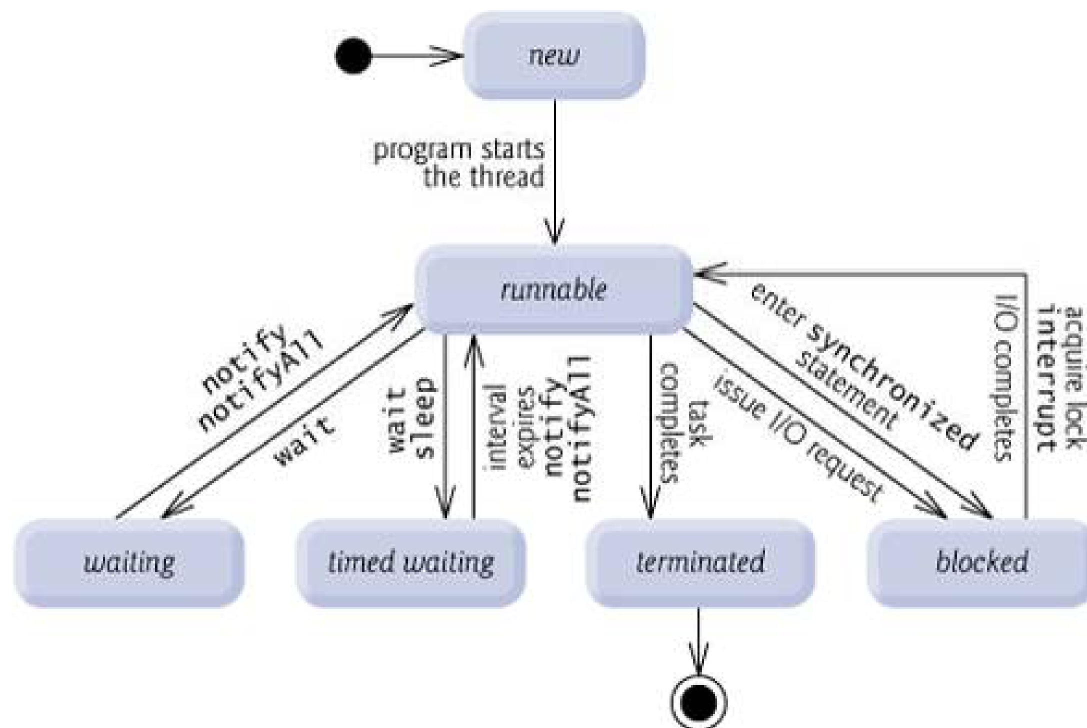
Runnableを実装したクラスをThreadクラスのコンストラクタとして渡す。start()で開始。

```
CountTenRunnable ct = new CountTenRunnable();
Thread th = new Thread(ct);
th.start();
```

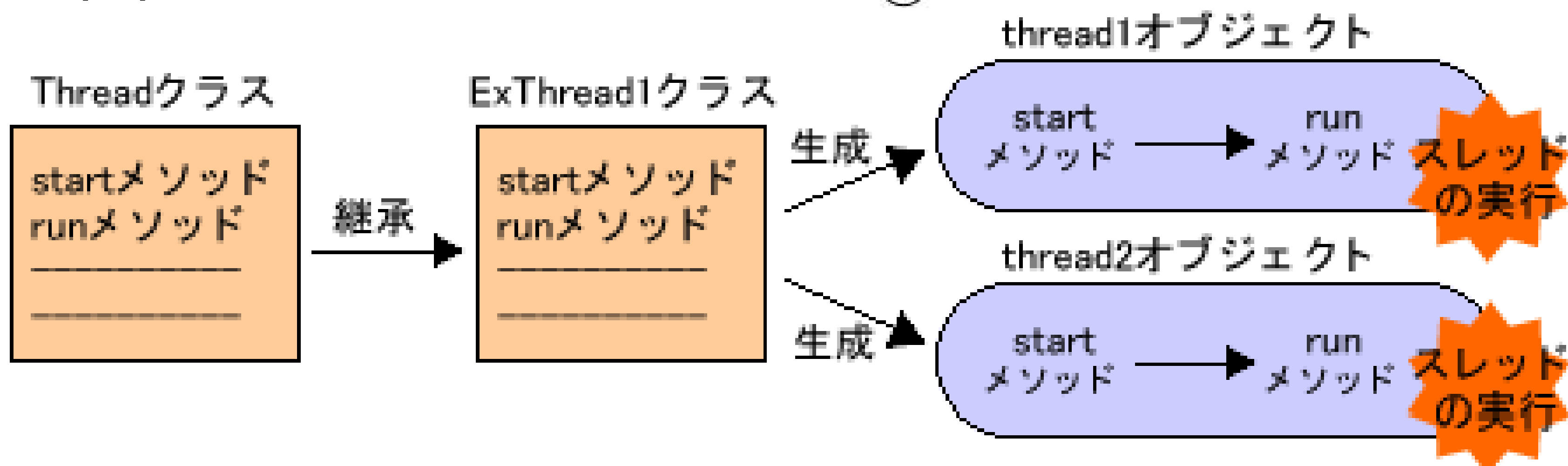
- 2)**extends Thread**

Threadを拡張したクラスをnew してstart()メソッドを呼び出す。

- (1)



- (2)



```
class MainThread{
    public static void main(String args[]){
     /* 別スレッドとして動作させるオブジェクトを作成 */
    SubThread sub = new SubThread(); /* 別のスレッドを作
   成し、スレッドを開始する */
    Thread thread = new Thread(sub) thread.start();
    }
   }
   class SubThread implements Runnable{
         public void run(){ }
   }
```

動かしてみよう

- CountTest.java
- CountTenRunnable.java
- CountTesterTweThreads.java

MovingBall

```
MovingInnerFFrame f = new MovingInnerFFrame();
Thread th = new Thread(f);
th.start();


class MovingInnerFFrame extends Frame
    implements Runnable {

    public void run() {}
}
```

第5回演習課題

- MovingBallを改造し3色以上の複数のボールを表示せよ。
- 壁の跳ね返りを画面のサイズに修正せよ。

- コンソールに残り秒数カウントダウンしてゲームが終了するようにThread.sleep()メソッドを用いよ。
- なお終了時間は20秒とせよ。
- ヒント
  - Thread.sleep(1*1000);

MovingBallを配列にしよう。

Singleton

Singleton

- たった一つのインスタンスしか作らせないようにするパターン。
普通はインスタンスを沢山作る
- 場合によってはインスタンスを一つしか作らせたくない
- プログラマ任せにすると、間違ってnewを複数回呼び出してしまう。
Singltonパターンを適用すると、指定したクラスインスタンスが１つしか存在しないことを保証する。

Singleton

Singleton
- singleton
- Singleton()
+ getInstance()

```
public class MyFirstSingleton {
/* 唯一のインスタンス。*/
private static final MyFirstSingleton instance = new MyFirstSingleton();
/** * コンストラクタ。*/
private MyFirstSingleton() { }
    /** * このクラスの唯一のインスタンスを返す。*/
    public static MyFirstSingleton getInstance() {
    return instance;
    }
}
```

Singleton

- private な static 変数を定義して初期化
  インスタンスは、このクラスのロード時に一度だけ生成処理。
  コンストラクタはpublicでなく外部に公開せず private。外部からインスタンス生成されることを防ぐ。privateにしないと外部から new とされてしまいインスタンスが自由に作れてしまう
  getInstance() を作成し外部に公開します。
  作られた唯一のインスタンスを返却することがこのメソッドの役目。public メソッドにしてどこからでも呼び出せるように。
  生成のタイミングは、初めて getInstance() をが呼ばれた時です。

MyFirstSingletonCall

```
void Test() {
 /* new できません。コンパイルエラーとなります。 */
// MyFirstSingleton mys = new MyFirstSingleton ();
// この時点で  インスタンス生成＆返却
MyFirstSingleton mys2 = MyFirstSingleton.getInstance();
// 同じインスタンス返却
MyFirstSingleton mys2 = MyFirstSingleton.getInstance();
if (mys1 == mys2) {
        System.out.println("同じインスタンスです。");
    }
}
```

Swing

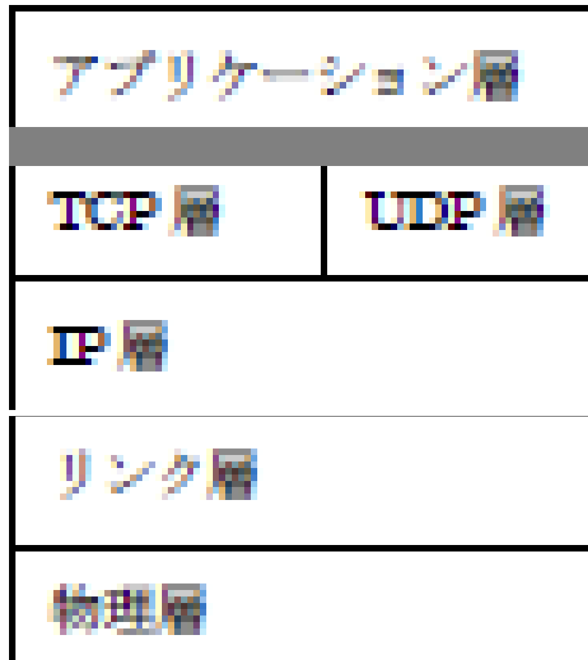
- SwingAnimationBasic.java
- SwingAnimationFaceObj.java

動かしてみよう。

- SwingAnimationBasic
- SwingAnimationFaceObj

- Swing　より美しいグラフィックスを提供 java2D
- ちらつき防止
- http://www.java2s.com/Tutorial/Java/0240__Swing/Catalog0240__Swing.htm

Socket interface



ソケットインターフェイス

UDP通信

- UDP通信では接続という概念はない。
- ユーザは毎回データを送信し、また毎回自分のソケット、及び相手のソケットとアドレスを指定することになる。TCPでは最初に相手のソケットとの間の接続を行い、双方のソケット間の接続が確立されたら、互いの通信が可能になる。
- TCPではそのような制限はない。
- TCPでは接続が確立されたら2つのソケットはストリームのように振る舞う。TCPでは受信したデータの信頼性と順序が保障される。

DatagramPacket

- SocketやServerSocketでは、データのやりとりは入出力ストリームに抽象化されていたため、パケットという概念が見えてこなかった。
- DatagramPacketとDatagramSocketの場合はUDPパケットはUDPパケットとして抽象化されており、パケットにデータも送信先アドレスも含める必要がある。
- SocketやServerSocketとは別物

UDPServer

```
int serverPort = 5000;

DatagramSocket socket = new
   DatagramSocket(serverPort);

DatagramPacket receivePacket = new
   DatagramPacket(new byte[DMAX], DMAX);

socket.receive(receivePacket);


socket.close();
```

Udp Client

```
String servhostname="localhost";

InetAddress serverAddress =
    InetAddress.getByName(servhostname);
String message="hello UDP from yourname";
byte[] bytesToSend = message.getBytes();
int serverPort = 5000;
DatagramSocket socket = new DatagramSocket();
DatagramPacket sendPacket = new
    DatagramPacket(bytesToSend, bytesToSend.length,
    serverAddress, serverPort);
socket.send(sendPacket);
socket.close();
```

ソケット通信

プログラム同士の通信は

- ソケットを使ってデータの送受信
- ソケットを使った通信

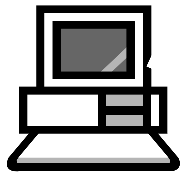
ソケット通信

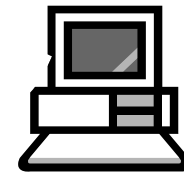
ソケット

意味：「接続の端点」

コンピュータとTCP/IPを
つなぐ出入り口

ソケット





ソケット通信

- ソケットを使って通信を行うには
  ２つのプログラムが必要

クライアントプログラム

ソケットを用意して
サーバに接続要求を行う

サーバプログラム

ソケットを用意して接続要求を待つ

ソケット通信の全体の流れ

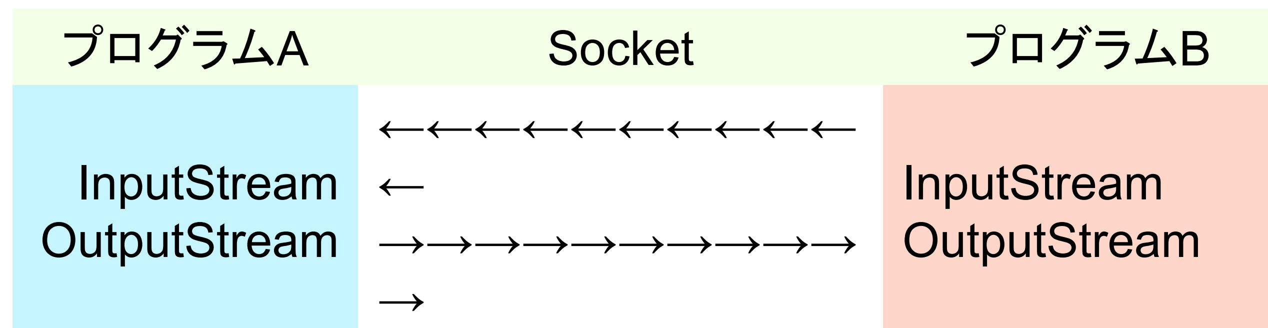
クライアント		サーバ
ソケット生成(socket)		ソケット生成(socket)
サーバを探す (gethostbyname)		接続の準備(bind)
		接続待機(listen)
接続要求(connect)	識別情報	接続受信(accept)
データ送受信(send/recv)		データ送受信(send/recv)
ソケットを閉じる(close)		ソケットを閉じる(close)

Sokect通信

- 双方がデータのやりとりを行う場合、双方を結んで情報を運ぶための「線」が必要となります。Javaにおいてはこの「線」のことを***「Socket（ソケット）」***という概念で表します。「Socket」は逆方向の情報の流れ（Stream）をもつ二本の通信線を一本に束ねたものだと考えられます。

InputStream, OutptStream

- 仮にAとBという二つのプログラムが通信を行うとすれば、一方の通信線がAにとっての「InputStream」でありかつBにとっての「OutptStream」、もう一方の通信線はAにとっての「OutputSteam」かつBにとっての「InputStream」となる。
- この2本の通信線を束ねる「Socket」を介して情報のやりとりが行われる。



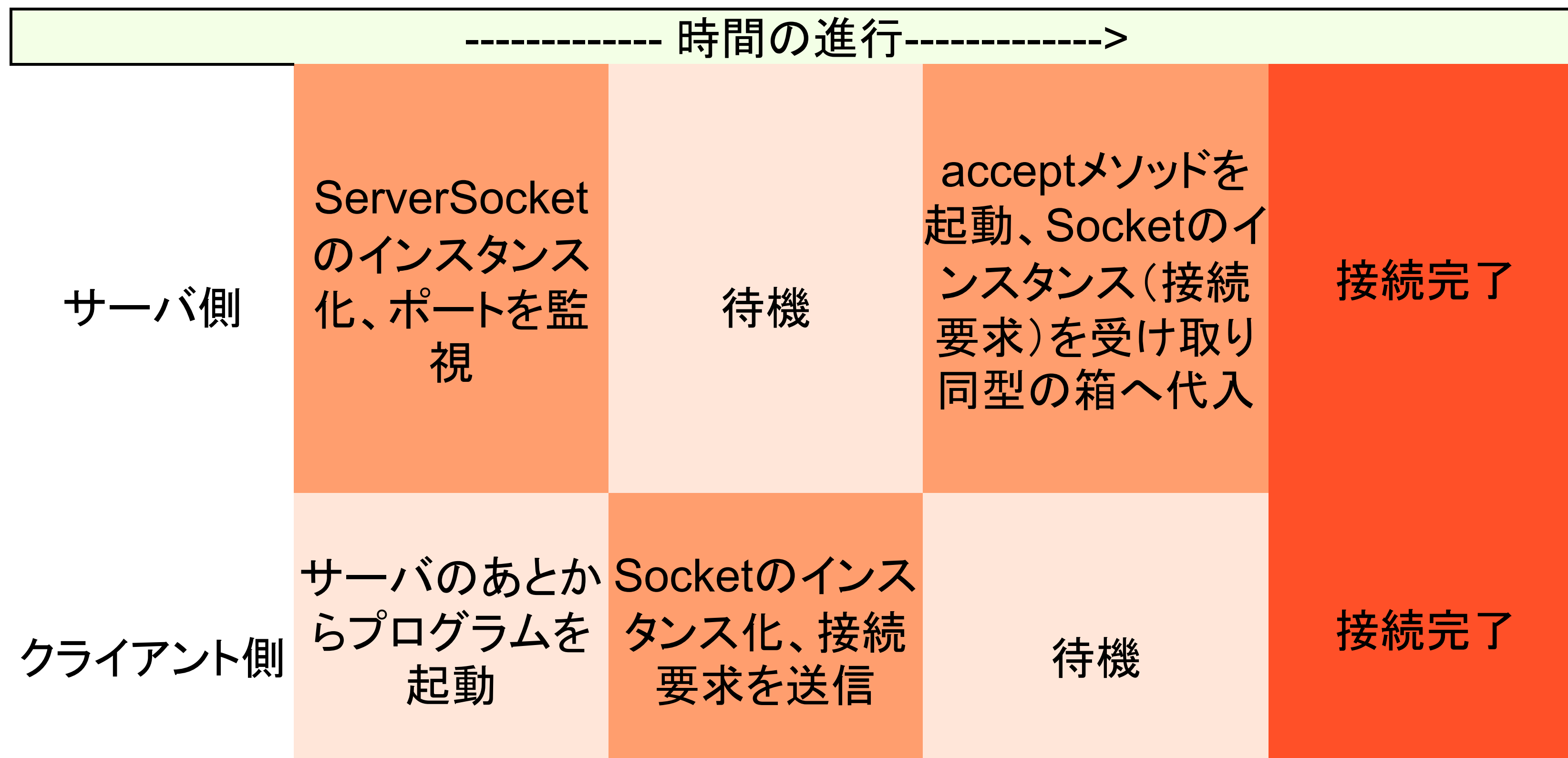
Socketとport番号

- ***クライアント側からサーバ側に向かって「Socket」が送り込まれて、初めて通信が開通***

- クライアントが「Socket」を送り込む際には、必ず「どのサーバコンピュータ」の「何番の通信口」に向かって送り込むのかを明らかにしなければなりません。
- その際の「どのサーバコンピュータ」のことを一般に***「サーバ名」***、「何番の通信口」のことを***「ポート番号」***、と呼びます。サーバには複数の「通信口＝ポート」があり、番号で管理しています。サーバの側は、プログラムによって決まった番号のポートを監視しています。
- クライアントはサーバが監視するポート番号に「Socket」を送らなければなりません。

ServerSocketクラス、Socketクラス

- **ServerSocketクラス**　サーバの側に用いられます。インスタンス化の際に引数として「ポート番号（整数型）」を要求します。インスタンス化が済んだ時点でポートの監視が始まります。そのとき、もしもクライアントから接続要求（「Socketクラス（後ほど説明）」のインスタンスが送られてくる）があれば、「acceptメソッド」で受け取ります。この時点で通信の準備が完了します。
- **Socketクラス**　クライアントの側に用いられます。インスタンス化の際に引数として「サーバ名（文字列型）、ポート番号（整数型）」の順に二つの引数を要求します。インスタンス化が行われた時点で接続要求がサーバに送られます。



------------- 時間の進行------------->				
サーバ側	ServerSocketのインスタンス化、ポートを監視	待機	acceptメソッドを起動、Socketのインスタンス（接続要求）を受け取り同型の箱へ代入	接続完了
クライアント側	サーバのあとからプログラムを起動	Socketのインスタンス化、接続要求を送信	待機	接続完了

Objectのやりとり

- OutputStreamへのデータの書き込み → データの送信
- InputStreamからのデータの読み込み → データの受信

- **OutputStream/InputStreamの取得** OutputStream/InputStreamの取得を行うためには、Socketクラスのメソッドである***「getOutputStream/getInputStream」メソッド***を利用します。これらのメソッドが起動されると、返値として、取得したOutputStream/InputStreamのインスタンスが返されます。
- このインスタンスを引数にして、Streamを用いてデータの通信を行うクラスをインスタンス化することで、OutputStream/InputStreamが取得されデータ送受信の準備が完了します。
- 通信の際にやりとりするデータの型が「任意のクラスのインスタンス」の場合、データ通信は「***ObjectOutputSream/ObjectInputSreamクラス***」を用いて行います。従ってgetOutputStream/getInputStreamメソッドの返値であるOutputStream/InputStreamインスタンスを引数として、これらのクラスのインスタンス化を行うことで、OutputStream/InputStreamが取得されデータ送受信の準備が完了します。

ObjectOutputStream、ObjectInputStream

- **データ送信の準備**（但しObjectOutputStreamのインスタンス名をoos、Socketのインスタンス名をsocketとする）
- ObjectOutputStream oos = new ObjectOutputStream(socket.getOutputStream());
- **データ受信の準備**（但しObjectInputStreamのインスタンス名をois、Socketのインスタンス名をsocketとする）
- ObjectInputStream ois = new ObjectInputStream(socket.getInputStream());

OutputStreamへのデータの書き込み（送信）

- OutputStreamへのデータの書き込みは、「ObjectOutputStream」の「writeObject」メソッドを利用して行います

- **データの書き込み（送信）**
- oos.writeObject(送信したいインスタンス名);
- oos.flush();

InputStreamからのデータの読み込み（受信）

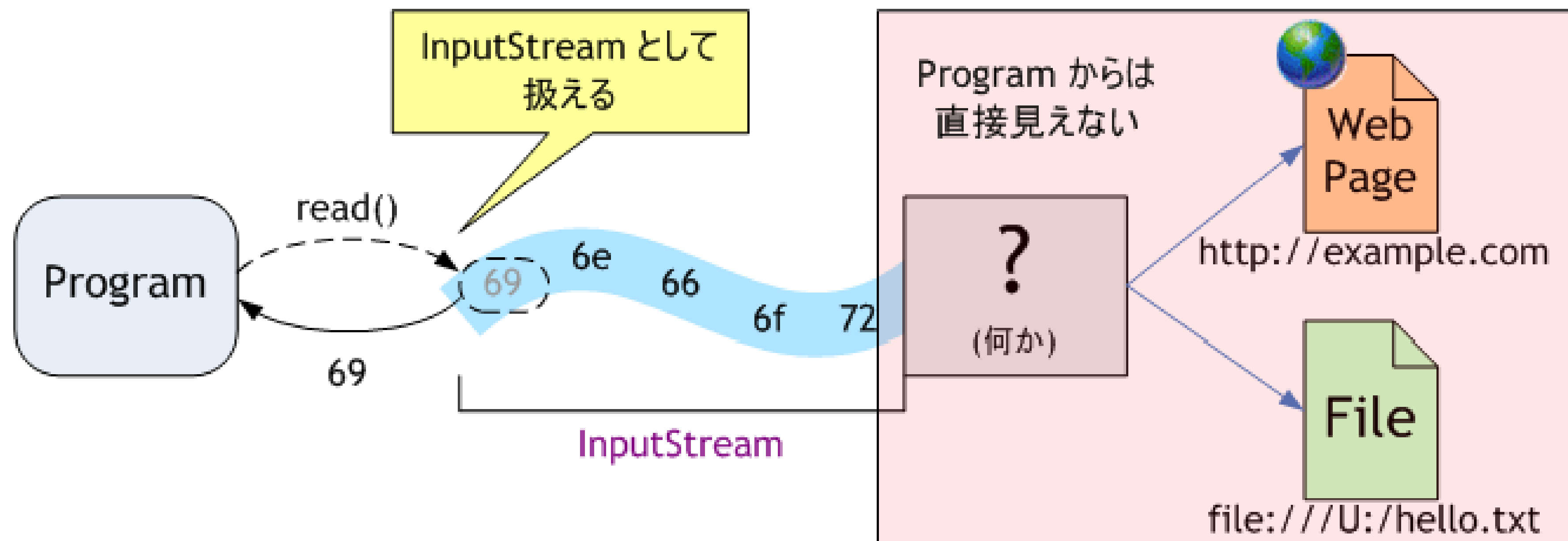
- InputStreamからのデータの読み込みも、ファイルからの読み込みと同様「ObjectInputStream」の「readObject」メソッドを利用します。

- **データの読み込み（受信）**（但し、ObjectInputStreamのインスタンス名をois、読み込むデータをVector型のインスタンスとする）
- Vector vec = (Vector)ois.readObject();

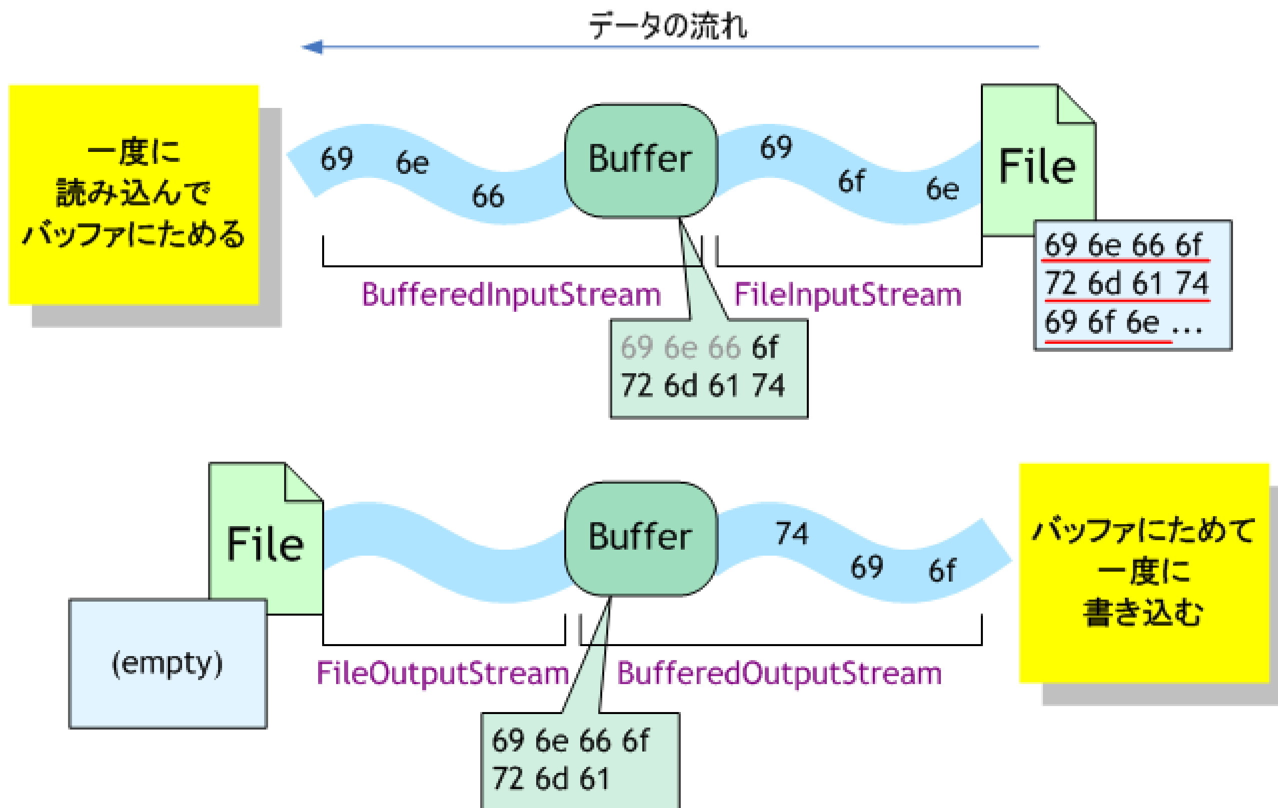
通信終了の合図

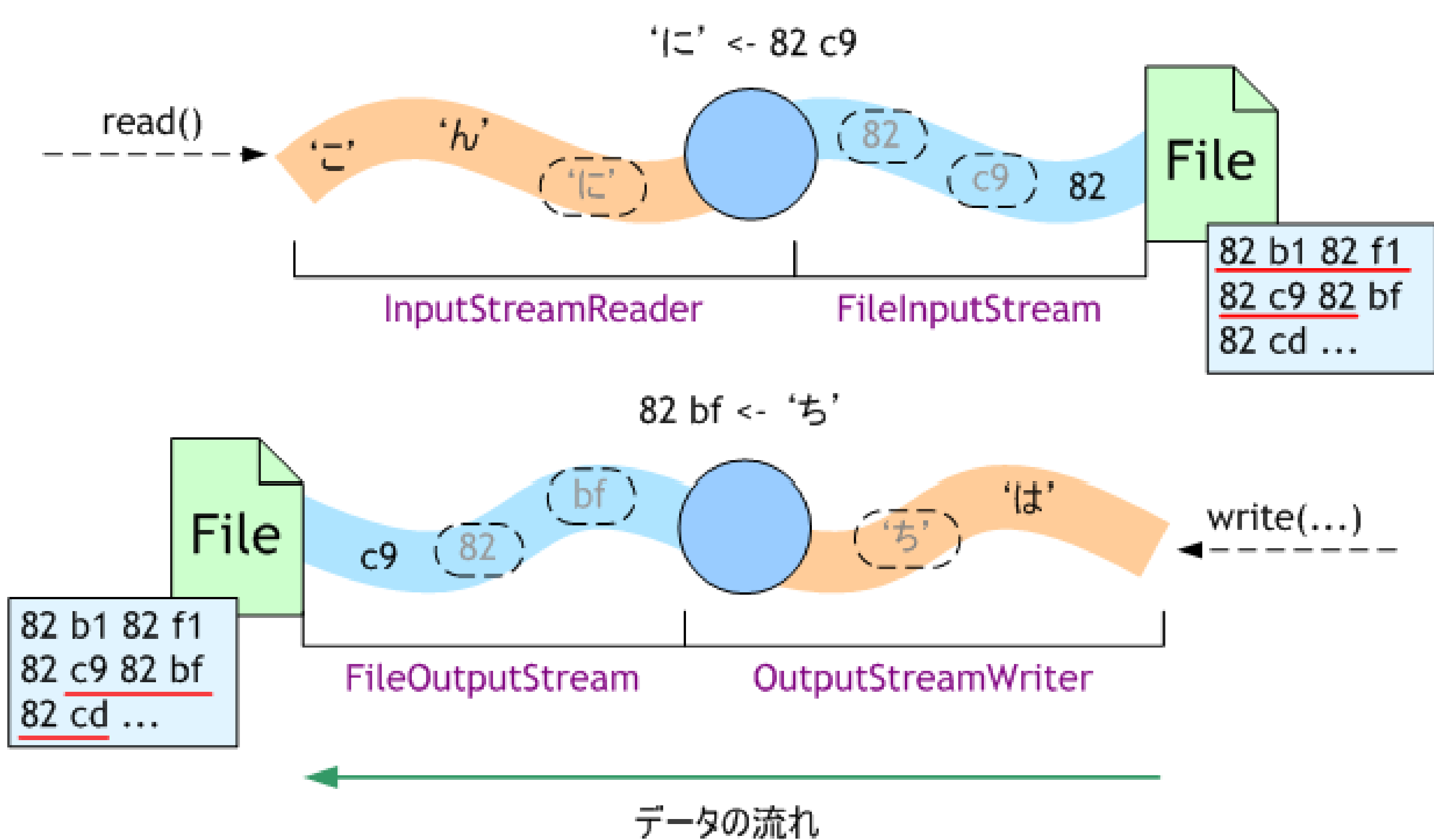
- **通信終了の合図**（但し、ObjectOutputStreamのインスタンス名をoosとする）
- oos.close();
- socket.close();

IO Stream



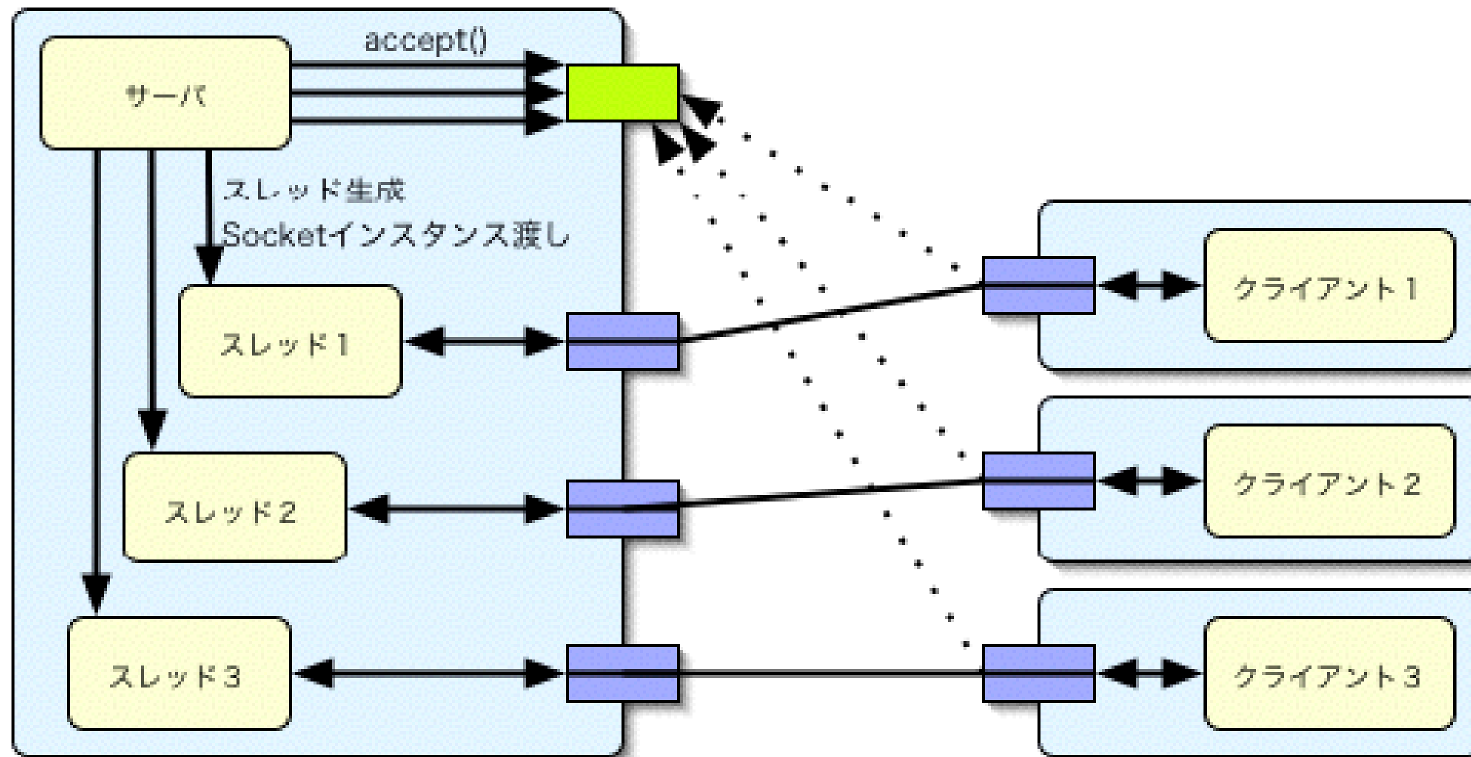
BufferdI/OSream





'に' <- 82 c9
read()
'こ'
'ん'
'に'
82
c9
82
File
82 b1 82 f1
82 c9 82 bf
82 cd ...
InputStreamReader
FileInputStream
82 bf <- 'ち'
File
c9
82
bf
'ち'
'は'
write(...)
82 b1 82 f1
82 c9 82 bf
82 cd ...
FileOutputStream
OutputStreamWriter
データの流れ

MultiThread



動かしてみよう

- CountTenRunnable

try-catch

```
try {
    例外のスロー
} catch (Exception e ) {
    例外のキャッチ
}
```

例外を投げる

```
throw new Exception();

Public void method1 (int x) throws Exception
  {
    throw new Exception();
}
```

例外を投げる。

```
// NumberFormatException を捕捉するための try-catch
        try {
            n = Integer.parseInt(input);
        }
        catch (NumberFormatException e) {
            // 数値の形式が不正である場合は、入力自体が不正
            throw new IllegalInputException("不正な入力 " + input);
        }

        if (n < 0) {
            // 負の値が入力された場合は、不正な入力
            throw new IllegalInputException("不正な入力 " + input);
        }

        System.out.println("入力された正の値は" + n);
```

例外クラス

```
public class IllegalInputException extends
  Exception {
   public IllegalInputException(String
  message) {
      // 親クラスのコンストラクタにメッセージを
  渡す
      super(message);
   }
}
```

スタックトレース

- e.printStackTrace();　　を使ってみる

うごかしてみよう。

- MultiServerSample.java
- MultiClientSample.java