ネットワークプログラミング
21005 2号館10階
第11回
2014/12/8

岩井将行

(もうすぐクリッスマス♪)

最終課題

- 最終課題！

- ネットワークのTCP/UDP通信をつかった通信プログラムを作りなさい。（1:1 1:n n:m マルチキャストのいずれの通信モデルでもよい） 企画案を全員12月15日までに作成すること。

最終課題チーム

- 一人チームでもよい。二人一組のチームでプログラムを作成することも勧めるが、その場合は役割分担を明確にわけること。クオリティは一人でやるチームの1.008倍以上を期待する。採点は別々に行うので役割分担を明確にすること。

最終課題発表日1/12

- 最終発表会は1/12日,1/19日に設定する。学籍番号順。pptの説明資料(企画書の発展版)。プログラムが当日うまくいかなかった場合の動作説明のスクリーンショット、アピールポイント、将来展望を明記すること。ソースはすべて提出すること。Eclipseのエクスポートでプロジェクトファイルとしてzipで提出してもよい。

期末テスト

- 1月12日　1時間　9:10から開始、持ち込み可、プログラムをフォルダに提出します。
- その後ミニプロジェクト発表会。

企画案評価

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

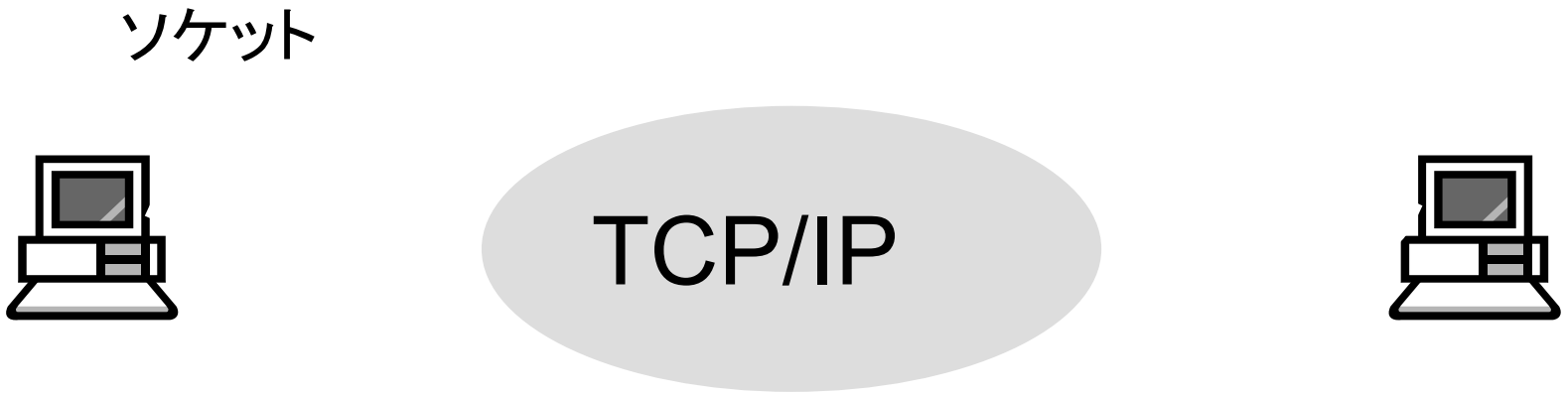
- ソケットを使ってデータの送受信
- ソケットを使った通信

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

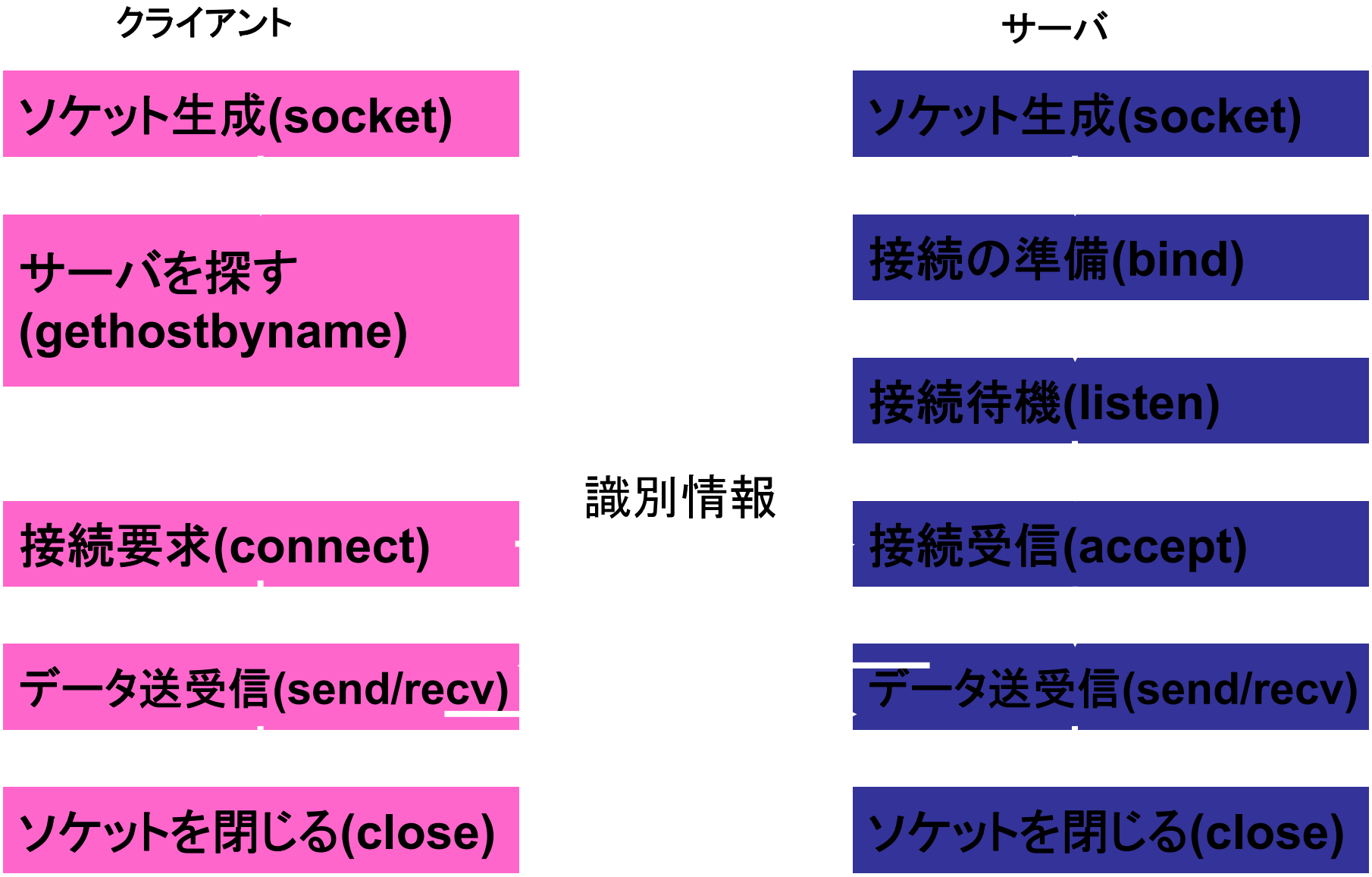
クライアントプログラム

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



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());
```

サーバプログラム（Java）

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();
```

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

- static String **getHostName**()
  この IP アドレスに対応するホスト名を取得。
- static InetAddress**getByAddress**(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 InetAddress**getLocalHost**()
  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[9];**

```
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

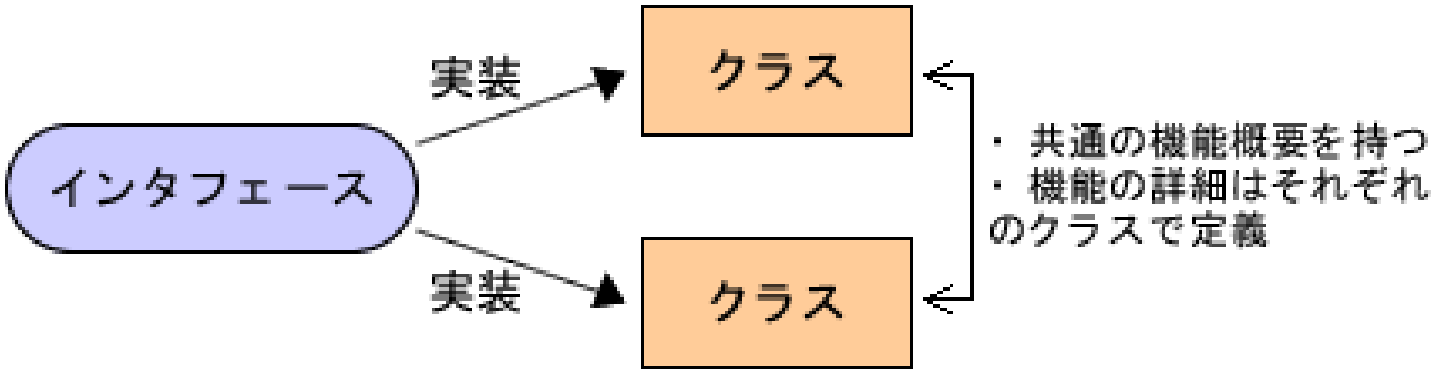
- クラスの場合は、単一のクラスしか継承 (extends) できせんが、インタフェースの場合は、複数のインタフェースを実装 (implements) することができます。

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

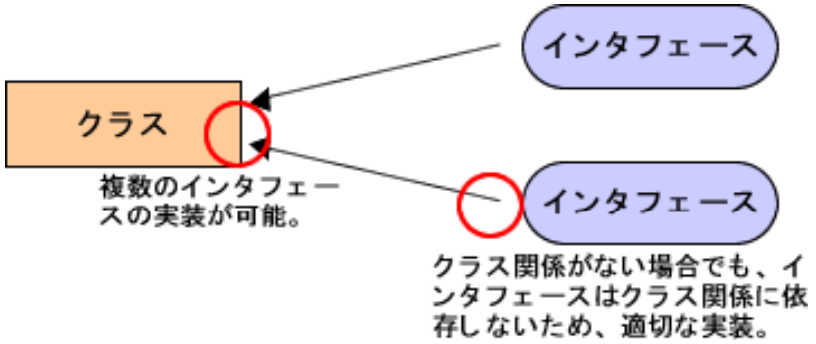
... }

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**

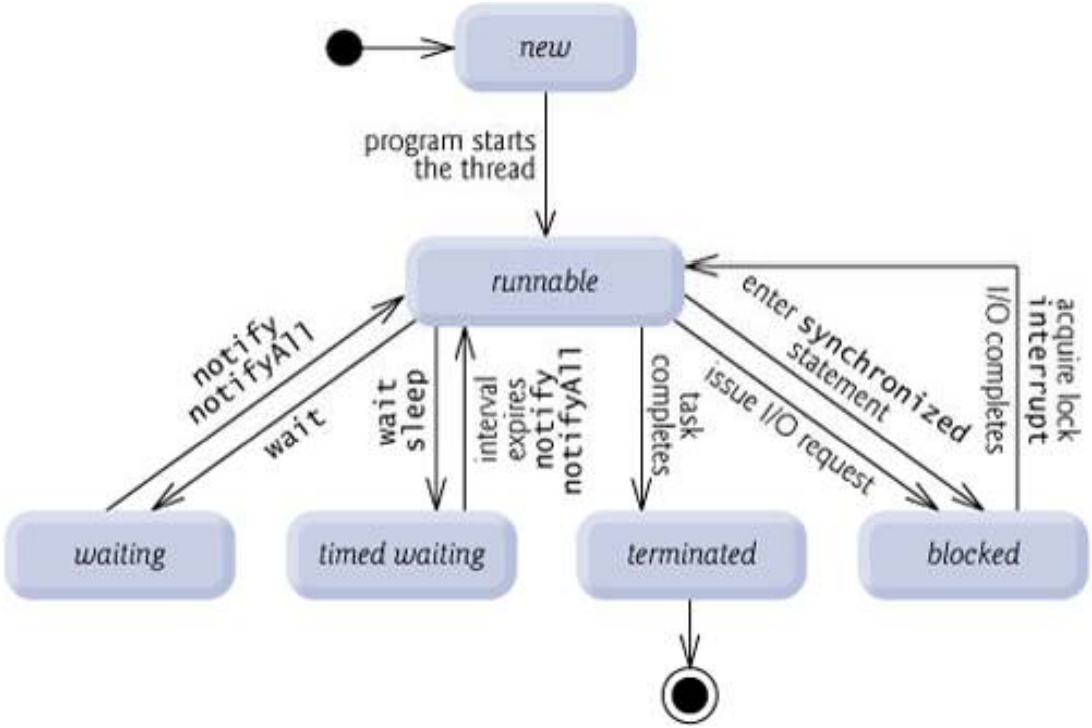
Runnableを実装したクラスをThreadクラスのコンストラクタとして渡す。start()で開始。

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

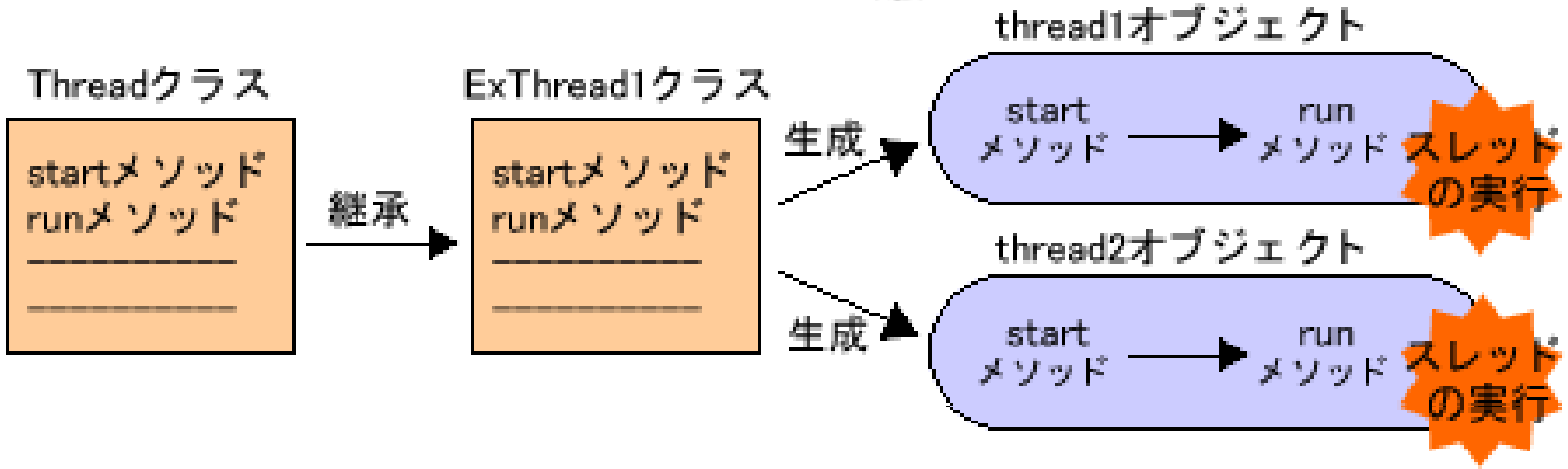
- 2)**extends Thread**

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

- (1)


new
program starts the thread
runnable
notify
notifyAll
wait
wait
sleep
interval expires
notify
notifyAll
task completes
issue I/O request
enter synchronized statement
I/O completes
acquire lock
interrupt
waiting
timed waiting
terminated
blocked

- (2)


Threadクラス
startメソッド
runメソッド
継承
ExThread1クラス
startメソッド
runメソッド
生成
生成
thread1オブジェクト
start メソッド
run メソッド
スレッドの実行
thread2オブジェクト
start メソッド
run メソッド
スレッドの実行

```
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(){ }
    }
```

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

MovingBall

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


class MovingInnerFFrame extends Frame
   implements Runnable {

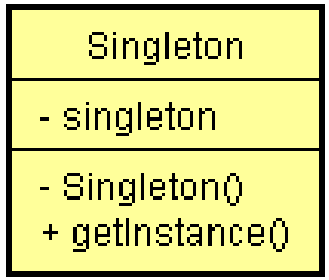
   public void run() {}
}
```

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




```
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

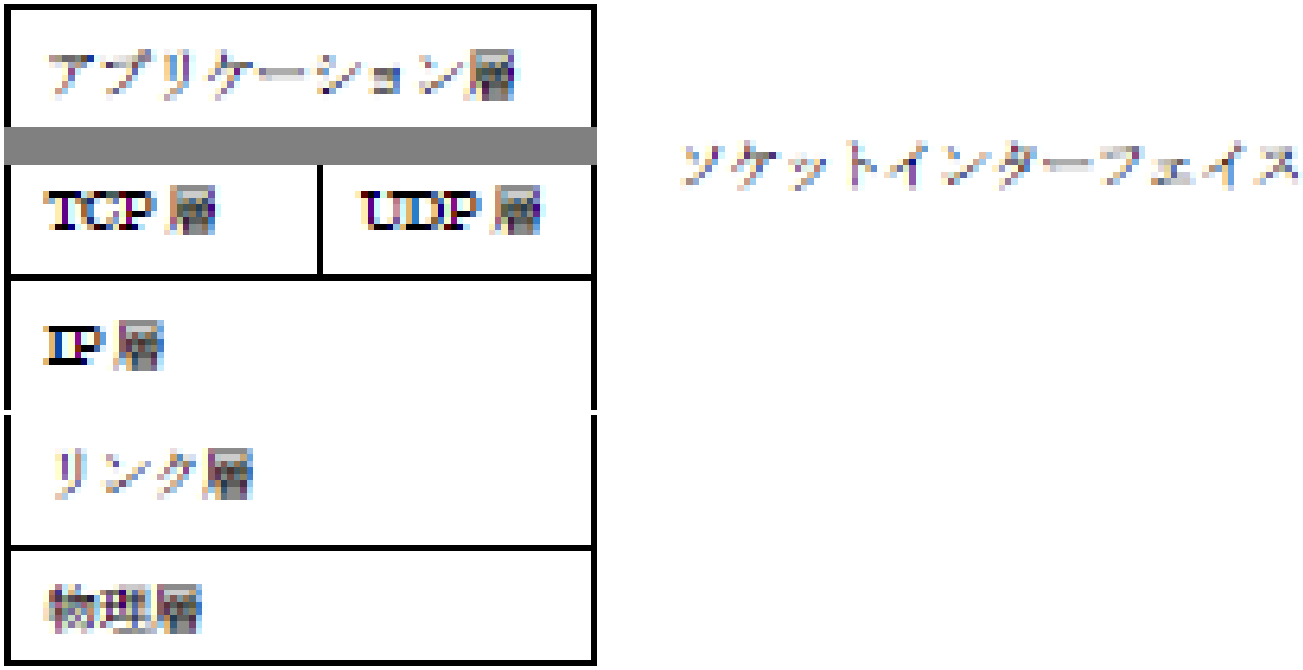
- SwingAnimationBasic.java
- SwingAnimationFaceObj.java

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



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

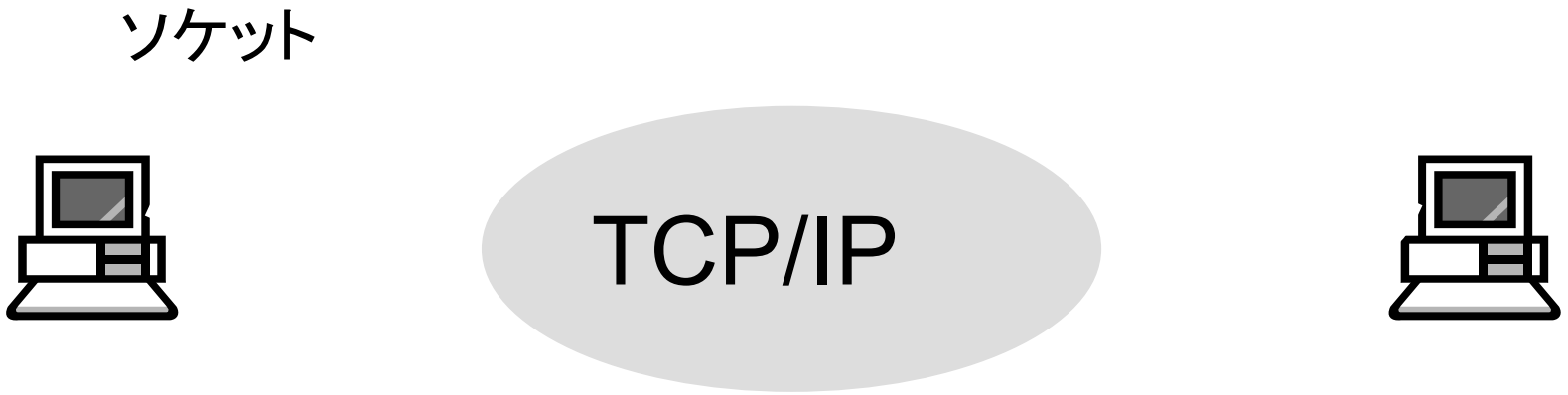
- ソケットを使ってデータの送受信
- ソケットを使った通信

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

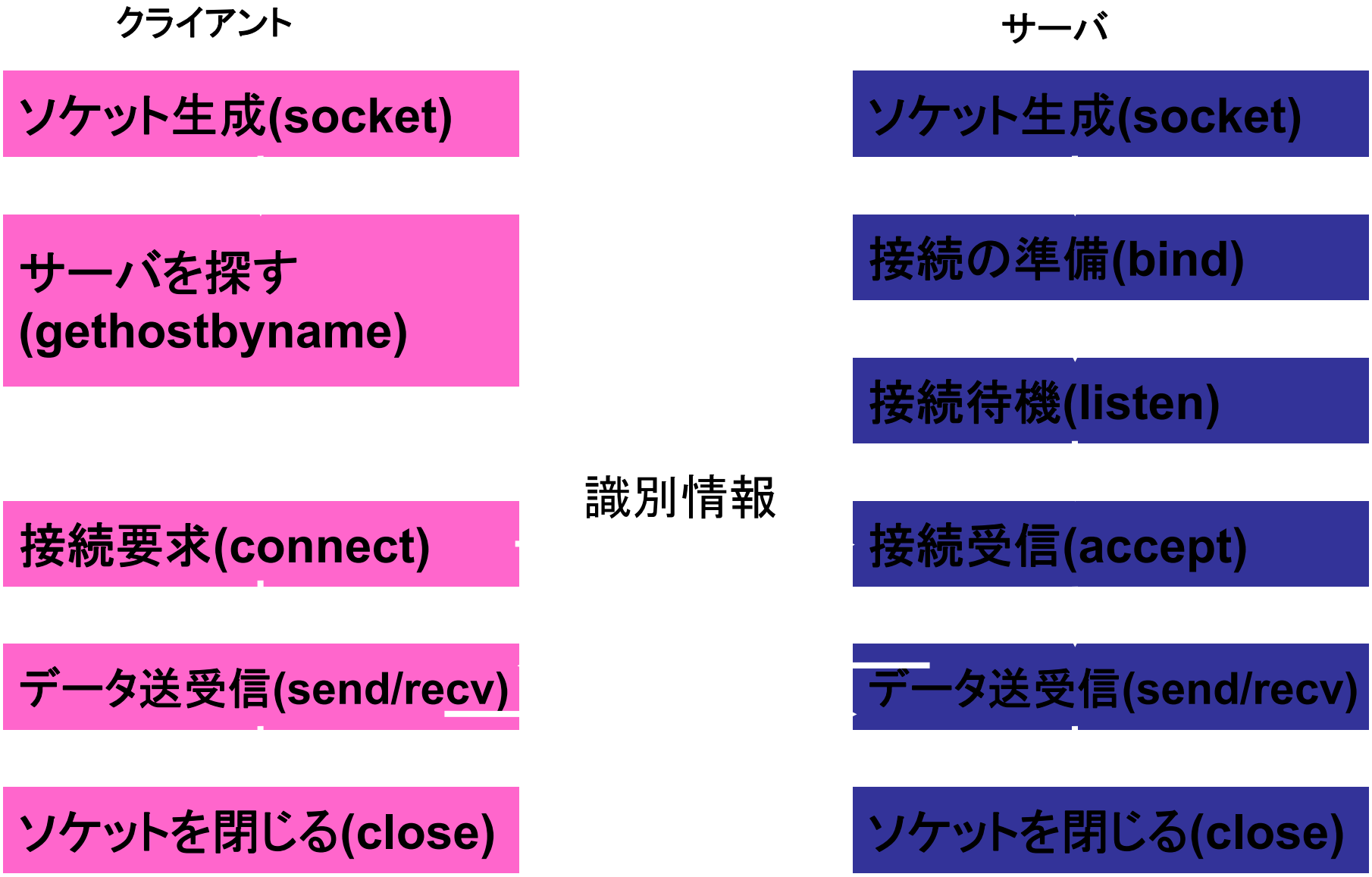
クライアントプログラム

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

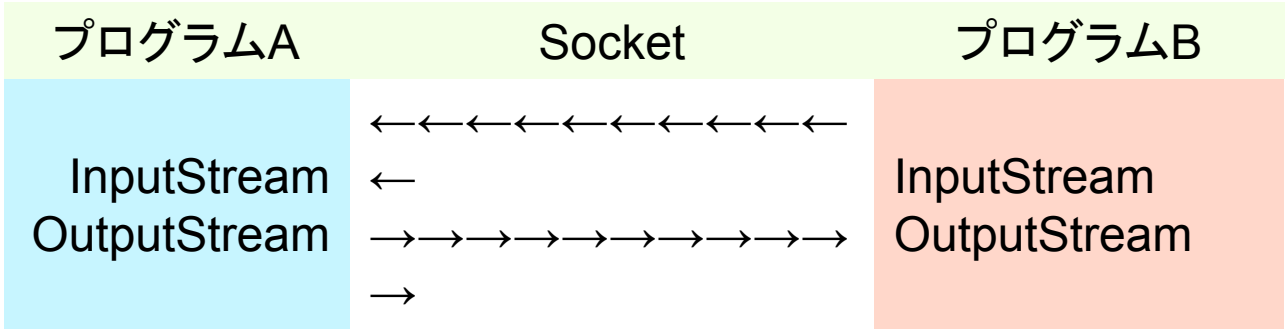


Sokect通信

- 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。



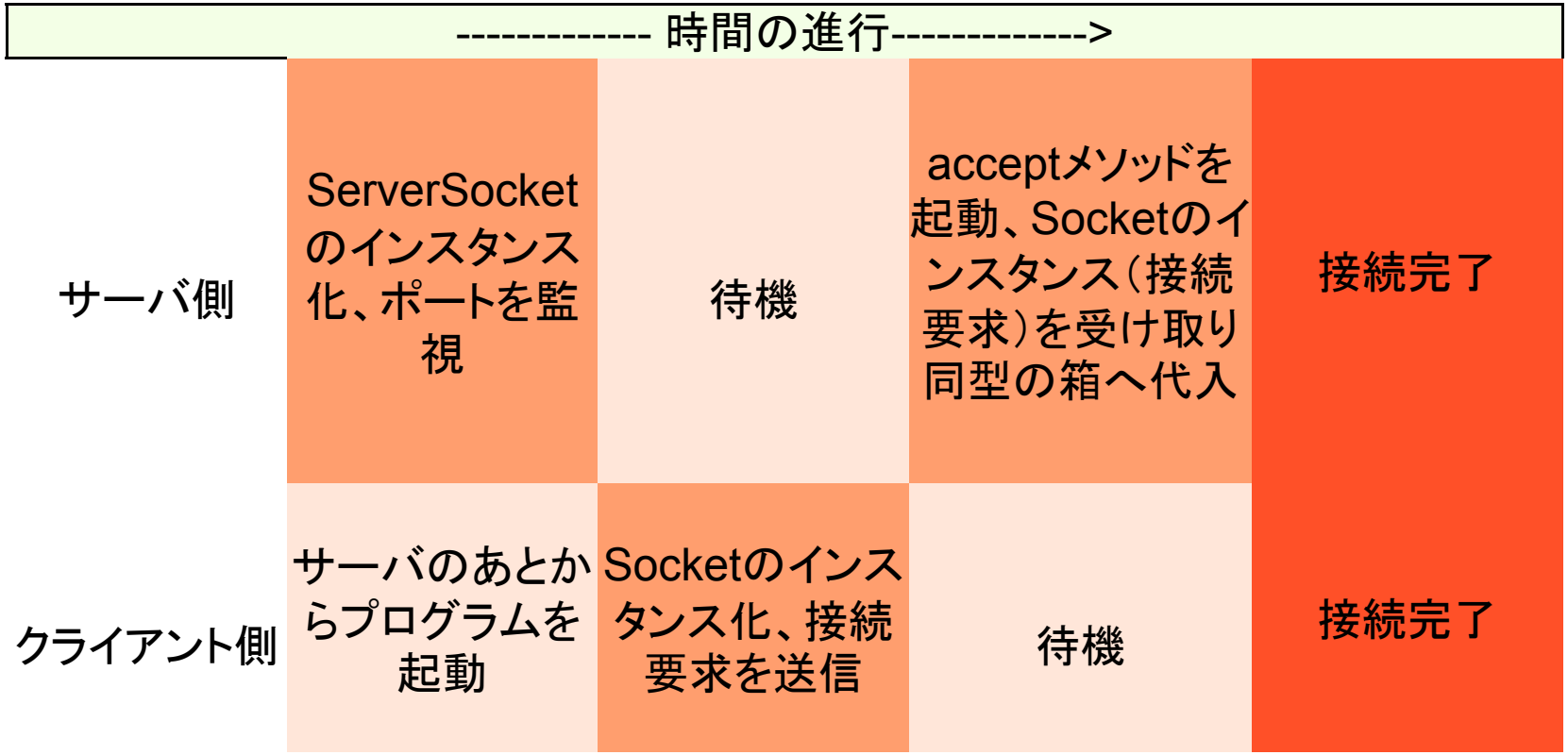
Socketとport番号

- ***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。

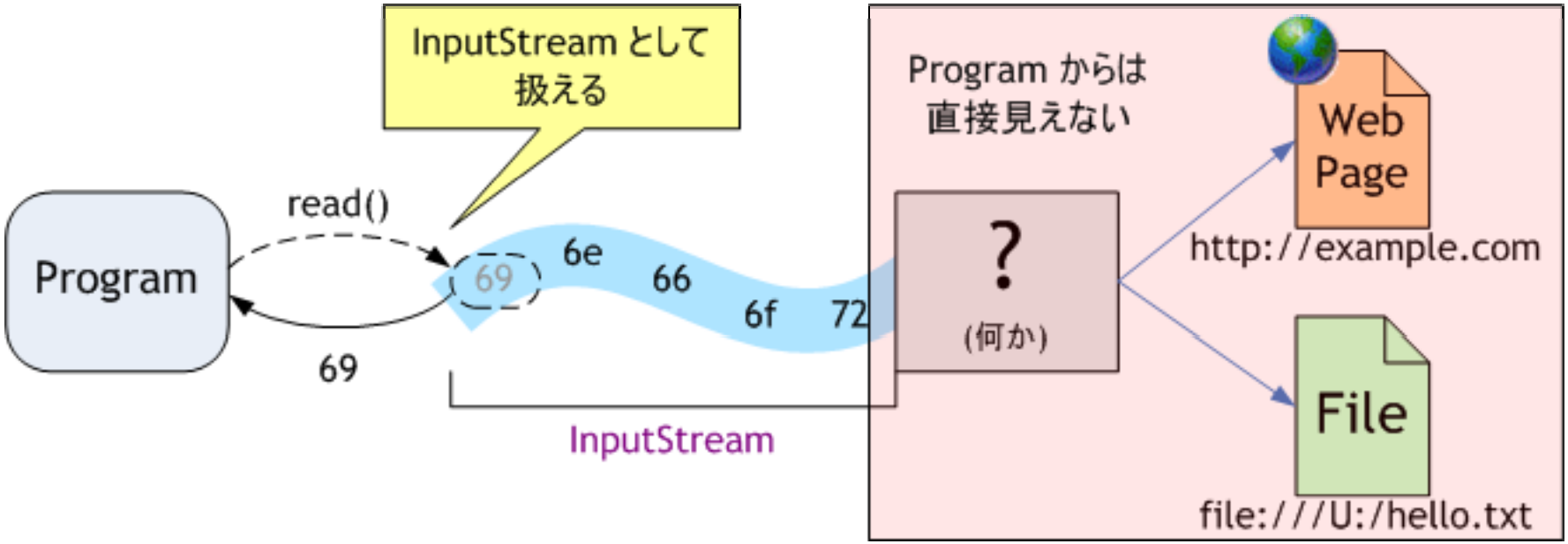
- **データの読み込み（受信）**（但し、ObjectInputStreamのインスタンス名をois、読み込むデータをVector型のインスタンスとする）
- Vector vec = (Vector)ois.readObject();

通信終了の合図

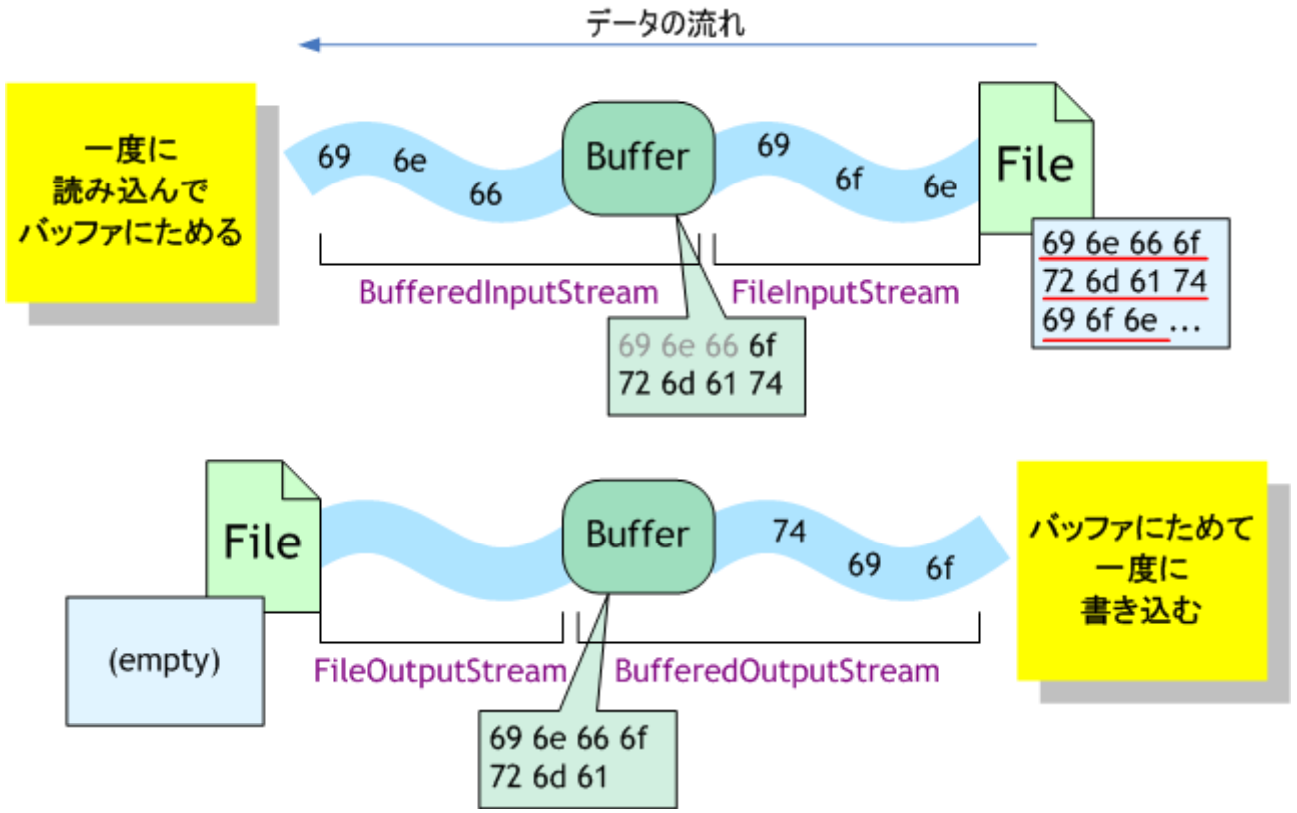
- **通信終了の合図**（但し、ObjectOutputStreamのインスタンス名をoosとする）
- oos.close();

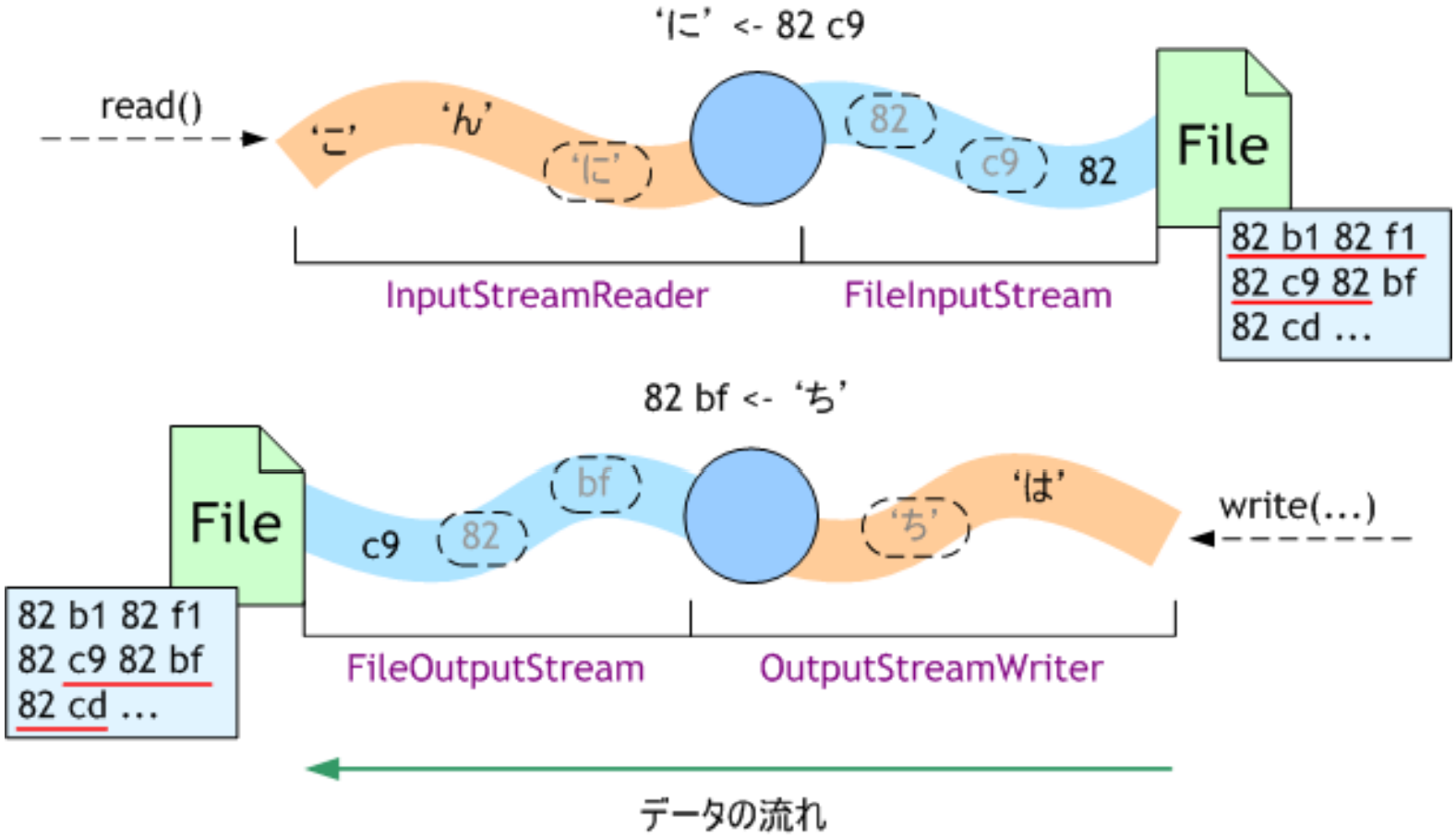
- socket.close();

IO Stream

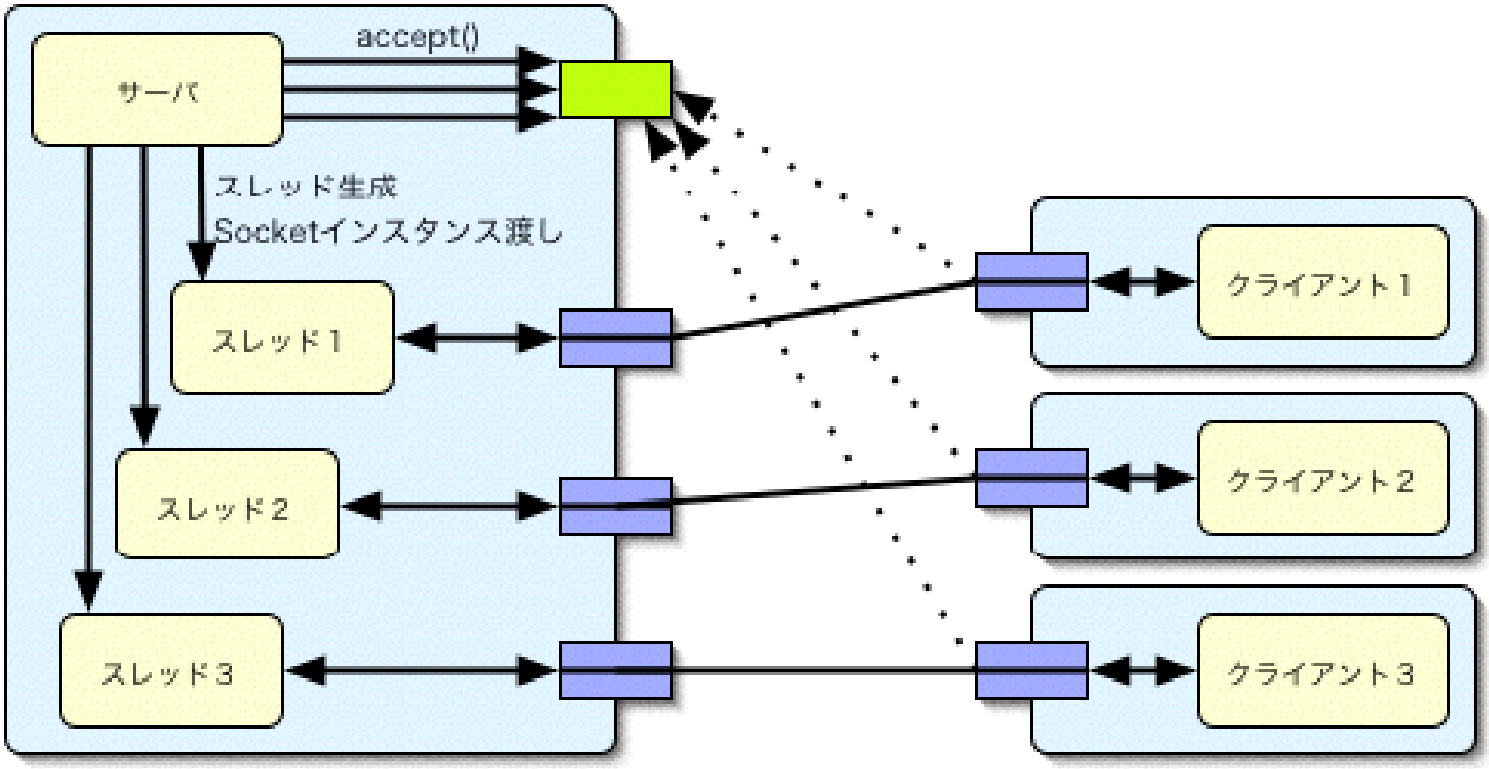

InputStream として
扱える
Program からは
直接見えない
Program
read()
69
6e
66
6f
72
69
InputStream
?
(何か)
Web
Page
http://example.com
File
file:///U:/hello.txt

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

Dinner

```
public class Dinner {

Dish dish1;
Dish dish2;
Dish dish3;

public static void main(String[] args) {

Dinner dinner = new Dinner();
dinner.eat3Dishes();
}

Dinner() {

dish1 = new Dish();
dish1.setName("特選シーザサラダ");
dish1.setValune(10);

dish2 = new Dish();
dish2.setName("銀しゃり");
dish2.setValune(2);

dish3 = new Dish();
dish3.setName("梅干し");
dish3.setValune(20);

}// Dinnerコンストラクターエンド

void eat3Dishes() {
String str = dish1.getName() + "=" + dish1.getValune() +
        ","
+ dish2.getName() + "=" + dish2.getValune() + ","
+ dish3.getName() + "=" + dish3.getV ();
System.out.println("たかしへ、ママです。今日の晩御飯は
      " + str + "よ");
}// eat end

// cook3Dishes()

}
```

Dish

```
ublic class Dish {

private String name = "noname";
private int valune = 0;

public String getName() {
return name;
}

public void setName(String name) {
this.name = name;
}

public int getValune() {
return valune;
}

public void setValune(int valune) {
this.valune = valune;
}

// public void cook(){}

}// Dishend
```

DinnerFullCourse

```
package guichat;

public class DinnerFullCourse {

Dish[] list = new Dish[5];// [0]-[4]の計5個

public static void main(String[] args) {

DinnerFullCourse fullcourse = new DinnerFullCourse();
fullcourse.eatAll();
}

DinnerFullCourse() {

list[0] = new Dish();
list[0].setName("特選シーザサラダ");
list[0].setValune(10);
list[1] = new Dish();
list[1].setName("銀しゃり");
list[1].setValune(20);
list[2] = new Dish();
list[2].setName("梅干し");
list[2].setValune(50);
list[3] = new Dish();
list[3].setName("冷めた感じ特選風スープ");
list[3].setValune(1);
list[4] = new Dish();
list[4].setName("締めとしての銀しゃりのお茶漬け");
list[4].setValune(20);
}// DinnerFullCourse()コンストラクターエンド

void eatAll() {
String str = "";

for (Dish element : list) {
str += element.getName() + "=" + element.getValune() +
       ">";
}

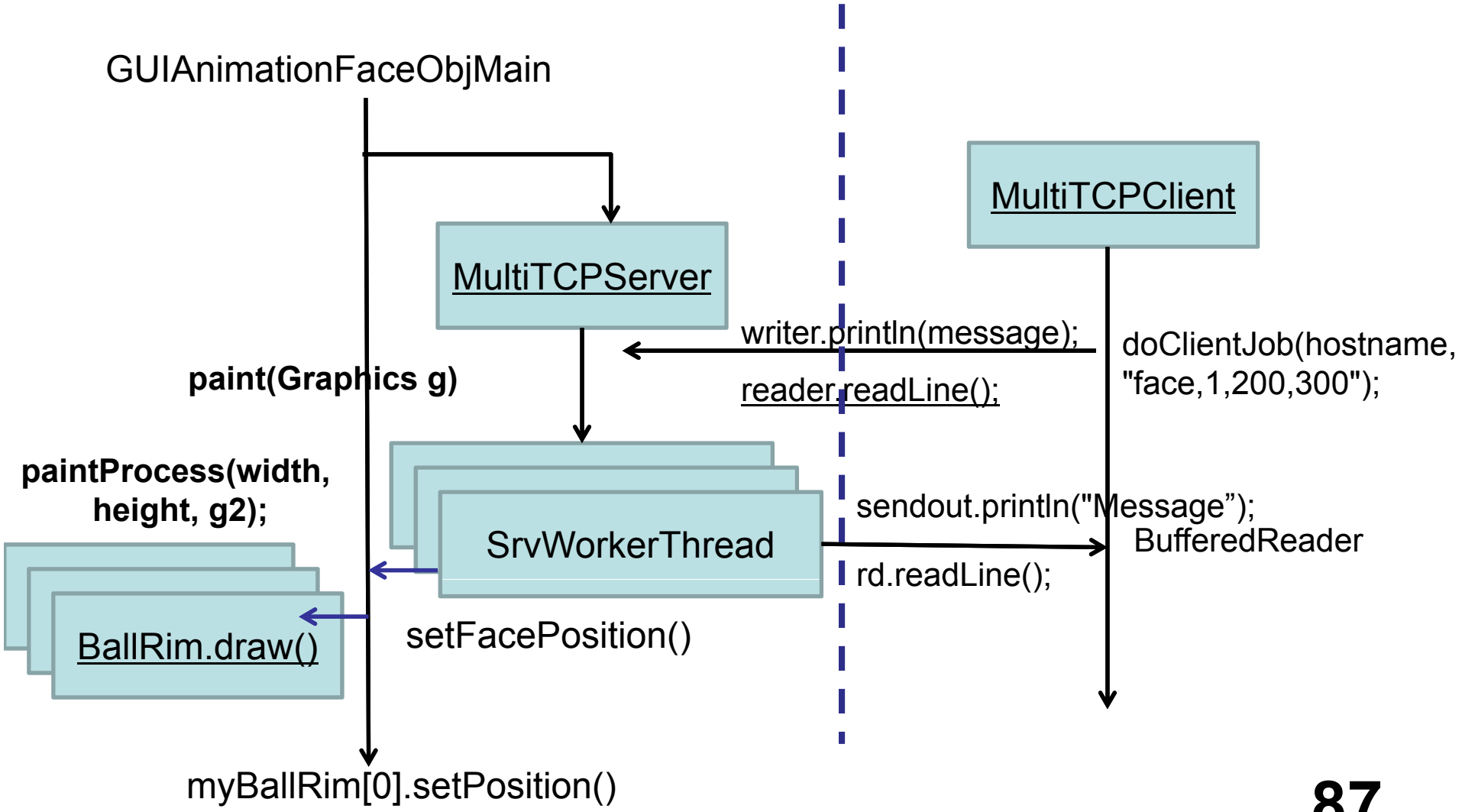
System.out.println("たかしへ、ママ2ですJ( '−`)し 今日の
      晩御飯コースは" + str + "よ");
}

}
```

動かしてみよう

- GUIAnimationFaceObjMain
  - サーバ（メッセージをうけとる。GUIを表示）
- MultiTCPClient
  - クライアント（メッセージをおくる。）

GUIAnimationFaceObjMain



課題

-guichatパッケージのGuiAnimation+Serverを改造して

-クライアントから表情を指定するとその表情に顔が変化(眉毛の角度)するように改造せよ。

感情は

-smile

-angly

- normalを実装すること

NIO2

ノンブロッキングIO New IO

Nioの前に

- Java1.7以降の改善点

Diamond operator

- 旧式の書き方

```
HashMap<String, Stack<String>> map =
    new HashMap<String, Stack<String>>();
```

新しい書き方

-
```
HashMap<String, Stack<String>> map =
    new HashMap<>;
```

Strings switch

旧式の書き方

```
if (input.equals(“yes”)) {
    return true;
} else if (input.equals(“no”)) {
    return false;
} else {
    askAgain();
}
```

新しい書き方

```
switch(input) {
    case “yes”: return true;
    case “no”: return false;
    default: askAgain();
}
```

動的リソース管理

- 旧式の書き方

```
public void oldTry() {
    try {
        fos = new
FileOutputStream("movies.txt");
        dos = new
DataOutputStream(fos);
        dos.writeUTF("Java 7 Block
Buster");
        } catch (IOException e) {
        e.printStackTrace();
        } finally {
        try {
            fos.close();
            dos.close();
            } catch (IOException e)
{
            // log the exception
        }
    }
}
```

- 新しい書き方

```
public void newTry() {
    try (FileOutputStream fos = new
FileOutputStream("movies.txt");
    DataOutputStream dos = new
DataOutputStream(fos)) {
        dos.writeUTF("Java 7 Block
Buster");
        } catch (IOException e) {
        // log the exception
    }
}
```

Copying a File (正しいが間違った書き方)

```
InputStream in = new FileInputStream(src);
OutputStream out = new FileOutputStream(dest);

try {
  byte[] buf = new byte[8192];
  int n;
  while (n = in.read(buf)) >= 0)
    out.write(buf, 0, n);
} finally {
  in.close();
  out.close();
}
```

Copying a File (正しいが複雑)

```
InputStream in = new FileInputStream(src);
try {
  OutputStream out = new FileOutputStream(dest);
  try {
    byte[] buf = new byte[8192];
    int n;
    while (n = in.read(buf)) >= 0)
      out.write(buf, 0, n);
  } finally {
    out.close();
  }
} finally {
  in.close();
}
```

Automatic Resource Management

```
try (InputStream in = new FileInputStream(src),
     OutputStream out = new FileOutputStream(dest))
{
  byte[] buf = new byte[8192];
  int n;
  while (n = in.read(buf)) >= 0)
    out.write(buf, 0, n);
}
```

Underscores in numbers

- Instead of

```
int million = 1000000;
```

  you can now say

```
int million = 1_000_000;
```

Multi-catch

```
public void newMultiMultiCatch() {
    try {
        methodThatThrowsThreeExceptions();
    } catch (ExceptionOne e) {
        // deal with ExceptionOne
    } catch (ExceptionTwo | ExceptionThree e) {
        // deal with ExceptionTwo and ExceptionThree
    }
}
```

New java.nio.file package

- A new java.nio.file package consists of classes and interfaces such as Path, Paths, FileSystem, FileSystems and others.

- ```
  public void pathInfo() {
      Path path = Paths.get("c:¥¥Temp¥¥temp");
      System.out.println("Number of Nodes:" +
                         path.getNameCount());
      System.out.println("File Name:" +
                         path.getFileName());
      System.out.println("File Root:" +
                         path.getRoot());
      System.out.println("File Parent:" +
                         path.getParent());
  }
  ```
  ```

File change notifications : WatchService

- ファイル変更を通知してくれる
- java.nio.file.WatchService
- 登録されたオブジェクトの変更およびイベントを監視する監視サービスです。たとえば、ファイルマネージャーでは、ファイルが作成または削除されたときにファイルリストの表示を更新できるように、監視サービスを使ってディレクトリの変更を監視することがあります。
- register メソッドを呼び出して Watchable オブジェクトを監視サービスに登録すると、その登録を表す WatchKey が返されます。オブジェクトのイベントが検出されると、その鍵は signalled になり、現在 signalled になっていない場合は、poll または take メソッドを呼び出して鍵の取得やイベントの処理を行うコンシューマが取得できるように、監視サービスのキューに入れられます。イベントの処理が完了すると、コンシューマはその鍵の reset メソッドを呼び出して鍵をリセットします。これにより、さらにイベントがあれば、その鍵は signalled になり、再度キューに入れられるようになります。
- 監視サービスへの登録は、鍵の cancel メソッドを呼び出すことにより取り消されます。

java.util.concurrent

- concurrent. **TimeUnit**
- TimeUnit は MILLISECONDS や MICROSECONDS から DAYS や HOURS に至るまで、あらゆる時間単位に対応することができます。これはつまり、必要な時間間隔のほとんど全種類を TimeUnit によって処理できる。※HOURS を簡単に MILLISECONDS に変換することができるなど

concurrent.
CopyOnWriteArrayList

- 配列をまったく新規にコピーする操作は、通常使用するには時間やメモリーがあまりにも余分
- ArrayListも変更などを考えるとCopyのコストは高い
- CopyOnWriteArrayList
- 「配列に対して変更を行うすべての操作 (add、set など) を、配列を新規コピーすることで実装しているスレッド・セーフな ArrayList である」

Concurrent.BlockingQueue

- BlockingQueue インターフェースの各項目は先入れ先出し (FIFO) の順序で保存されます。
- 特定の順序で挿入された項目は挿入時と同じ順序で取り出されます。さらに、空のキューから項目を取得しようとすると、その項目を取 得できるようになるまで、呼び出し側スレッドがブロックされることも保証されます。
- 同様に、キューが一杯の場合に項目を挿入しようとすると、そのキューの ストレージに空きができるまで呼び出し側スレッドはブロックされます。
- **※SynchronousQueue**

Concurrent.ConcurrentMap

- Mapは並列処理に対して小さなバグがある
- ConcurrentMap インターフェースには 1 つのロックで 2 つのことを実行するように設計されたメソッドがいくつか追加でサポート。
- 例えば putIfAbsent() は最初にテストを実行し、キーが Map の中に保存されていない場合にのみ put 操作を行います。

`java.lang.invoke`

There are JVM versions of Ruby, Python, and Clojure, among others

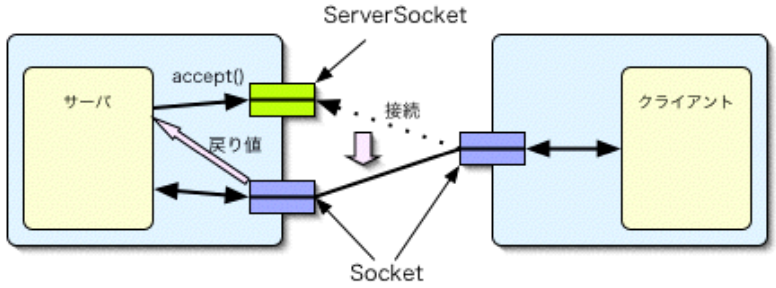
However, these are *dynamic* languages, for which the JVM does not provide great support (Java is a *static* language)

The new `java.lang.invoke` package doesn't help Java, but it provides better support for dynamic languages

NIO

従来のjava.netパッケージのソケットの欠点

- 従来の java.net パッケージを使用した入出力では、ソケットの accept メソッドや read メソッドなどを呼び出すと接続や入力があるまで処理が待ち状態になった。
- このような入出力待ちの動作のことをブロックという
- 接続の待ちうけでブロックが発生するため、複数のネットワーク接続を同時に処理するサーバアプリケーションを実装するにはマルチスレッドが必要。
- しかしスレッドの生成はそれなりにコストのかかる処理であり、アクセスの多いサーバ
きないくらい大きくっていた



nio

- そこで NIO ではブロックの発生しない入出力を実現する方法が提供された。

- ブロックされない入出力を利用すると、1つのスレッドでも複数の入出力を見かけ上同時に処理することができるようになる。

- NIO でネットワーク入出力を行うためには SocketChannel や ServerSocketChannel を利用します。これらはそれぞれ、java.io パッケージの Socket や ServerSocket に相当します。
これらのクラスはブロックする入出力とブロックしない入出力のどちらも利用することができる。

- Socket=> SocketChanel
- ServerSocket=>ServerSocketChanelに対応

selector

- ノンブロッキングモードのチャネルでは、入出力操作を行っても処理がブロックされない。
- たとえば ServerSocketChannel で accept() メソッドを呼び出した場合、接続があってもなくてもただちにメソッドの実行が終了してしまう。
- これではいつ入出力を行うメソッドを呼び出したらよいのか不明。
- 利用可能なチャネルを取得するための仕組：セレクタSelector)。
- セレクタにチャネルを登録しておき、そこか ら利用可能となったものを取り出して入出力を行います。
  セレクタには複数のチャネルを登録することが可能で、一つのスレッドでも見かけ上複数の入出力を同 時に行うことが可能。

SelectorProvider

- Selector は SelectorProvider により生成されます。

- SelectorProvider.provider() メソッドで取得し、Selector は SelectorProvider クラスの openSelector() メソッドで取得
  - Selector selector = SelectorProvider.provider().openSelector();
  - または
  - Selector selector = Selector.open();

SelectableChannel.register()

- //セレクタにチャネルを登録するには、SelectableChannel の register() メソッドを呼び出す
- SelectionKey register(Selector sel, int ops, Object att)
- SelectionKey register(Selector sel, int ops)

値	説明
SelectionKey.OP_ACCEPT	ソケットの接続受け付け操作
SelectionKey.OP_CONNECT	ソケットの接続操作
SelectionKey.OP_READ	読み込み操作
SelectionKey.OP_WRITE	書込み操作

いよいよselectorを呼び出す

- セレクタから利用可能となったチャネルを取り出すためには、通常はまずSelector クラスの select() メソッドを呼び出す。
select() メソッドは利用可能なチャネルの個数を返す
- ここで「利用可能とは」、たとえば OP_ACCEPT を指定して登録したServerSocketChannel に接続要求がきた場合をさす
- select() メソッドには以下の3つのバリエーション

- int select()
- int select(long timeout)
- int selectNow()
-
select() メソッドは利用可能なチャネルが出現する、割り込みが入る、
- timeout で指定した時間（ミリ秒）経過するまで処理をブロックする。
selectNow() メソッドはブロックしない操作で、利用可能なチャネルが存在しない場合ただちに0を返す。

Key判定

メソッド	説明
isAcceptable()	新規接続が受け付け可能であるかどうかを判定する
isConnectable()	接続可能であるかどうかを判定する
isReadable()	データ受信処理が可能であるかどうかを判定する
isWritable()	データ送信処理が可能であるかどうかを判定する

一つ進んでは戻すの繰り返し

- doAccept() メソッドでは新規接続の受付処理を行っています
  - SocketChannel channel = serverChannel.accept();
  - String remoteAddress = channel.socket()
  - .getRemoteSocketAddress()
  - .toString();
  - System.out.println(remoteAddress + ":[接続されました]");
  - channel.configureBlocking(false);
  - channel.register(selector, SelectionKey.OP_READ);
    戻す！！

やっと会えたね、碇シンジ君。

- **else if (key.isReadable()) {**
- doRead((SocketChannel) key.channel());
- }

NIO2:Asynchronous：非同期ソケットチャネル

- ava7にて NIO2 として不完全だった NIO 系ライブラリが拡張されました。非同期 SocketChannel を使って簡単
- Java6までの ServerSocketChannel と SocketChannel に対応する Asynchronous 系のクラスが追加されました。
- AsynchronousServerSocketChannel
- AsynchronousSocketChannel
- Asynchronous 系はこのほかに。
- AsynchronousFileChannel
- AsynchronousDatagramChannel

参考資料

- https://docs.oracle.com/cd/E26537_01/tutorial/essential/io/fileio.html
- http://itpro.nikkeibp.co.jp/article/COLUMN/20110927/369451/
- http://www.ne.jp/asahi/hishidama/home/tech/java/files.html
- https://github.com/nivance/Java7Demos/blob/master/nio/multicast/Sender.java
- http://d.hatena.ne.jp/Kazuhira/20130927/1380296230
- http://www.ibm.com/developerworks/library/j-nio2-1/
- http://etc9.hatenablog.com/entry/20110914/1316021631
  http://docs.oracle.com/javase/jp/8/technotes/guides/io/example/NBTimeServer.java
- http://www.javaworld.com/article/2078654/java-se/java-se-five-ways-to-maximize-java-nio-and-nio-2.html
- http://www.torutk.com/projects/swe/wiki/Java%E3%83%97%E3%83%AD%E3%82%B0%E3%83%A9%E3%83%9F%E3%83%B3%E3%82%B0_TCP%E9%80%9A%E4%BF%A1_%E3%83%A9%E3%82%A4%E3%83%96%E3%83%A9%E3%83%AA%E4%BD%BF%E7%94%A8%E3%82%A4%E3%83%A1%E3%83%BC%E3%82%B8
- http://www.slideshare.net/BalamuruganSoundararajan/nio-and-nio2
- http://www.powershow.com/view/233c8-YTQ5Z/Java_NIO_powerpoint_ppt_presentation
- http://etc9.hatenablog.com/entry/20100208/1265647084
- http://etc9.hatenablog.com/entry/20100209/1272986093
- http://etc9.hatenablog.com/entry/20110914/1316021631
- http://blog.elzup.com/%E3%82%A4%E3%82%A6%E3%82%A3%E5%95%8F%E9%A1%8C%E3%81%AE%E8%A7%A3%E3%81%8D%E6%96%B9/?fb_action_ids=648344635278324&fb_action_types=news.publishes&fb_ref=pub-standard

文字列の変換

- バイト配列型への変換

- byte配列型　byte[] buf = str.getBytes();
  - byte[] buf = str.getBytes("Shift_JIS");

- char配列型への変換
- char配列型
  - char c[] = str.toCharArray();

文字列の置き換えreplaceAll

- Java の String クラスには置換メソッドとして replaceAll() メソッドが用意されています。
- 文字列に指定した文字列が存在した場合に、指定した別の文字列に置換を行います。
- String str1 = "今日の天気は晴れです。";
- String str2 = "晴れ";
- String str3 = "雨";

- str1.replaceAll(str2, str3);

正規表現による置換

- Javaで正規表現のエスケープを行うには Pattern.quote(str)を使用

```
import java.util.regex.Pattern;
String str1 = "../test.html";
String str2 = "../";
String str3 = "http://www.example.com/";
str1.replaceAll(Pattern.quote( str2 ), str3);
```

文字列の一致を判定する

```
String strA = "文字列の検索サンプルです";
String strB = "検索";
if (strA.equals(strB)) {
System.out.println( "完全一致です" );
} else {
System.out.println( "完全一致ではありません"
   );
}
```

文字列を検索 indexOfメソッド

- 文字列を検索するには、indexOfメソッド
- public int indexOf(String str) indexOfメソッドは、ある文字列から別の文字列を左側から右側に検索し、見つかった位置を返す。該当する文字列がなかった場合は -1

```
String strA = "文字列の検索サンプルです";
String strB = "検索";
if (strA.indexOf(strB) != -1) {
System.out.println( "部分一致です" );
} else {
System.out.println( "部分一致ではありません" );
}
```

前方一致検索 startsWithメソッド

前方一致検索を行うには、startsWithメソッドを使用。

public boolean startsWith(String str)

一致した場合はtrue，不一致ならfalse

```
String strC = "文字列の検索サンプルです";
String strD = "文字列の";
if (strC.startsWith(strD)) {
  System.out.println( "前方一致（接頭辞）です" );
} else {
  System.out.println( "前方一致ではありません" );
}
```

後方一致検索 endsWithメソッド

startsWithメソッドとは逆に後方一致検索を行うには、
　endsWithメソッドを使います。

public boolean endsWith(String str)

一致した場合はtrue，不一致ならfalseになります。

```
String strX = "文字列の検索サンプルです";
String strY = "サンプルです";

if (strX.endsWith(strY)) {
  System.out.println( "後方一致（接尾辞）です" );
} else {
  System.out.println( "後方一致ではありません" );
}
```

正規表現での検索 matches

matchesメソッドは部分一致を検索するものではなく、
　完全一致を検索します。

```
String str51 = "文字列の検索サンプルです";
String str52 = "サンプル";

if (str51.matches(".*" + str52 + ".*")) {
  System.out.println( "部分一致です" );
} else {
  System.out.println( "部分一致ではありません" );
}
```

Javaで正規表現のエスケープ Pattern.quote(str)

```
import java.util.regex.Pattern;
String strA = "012345[a-z][0-9][a-z]012345";
String strB = "[a-z][0-9][a-z]";

if (strA.matches(".*" + Pattern.quote( strB ) + ".*")) {
  System.out.println( "部分一致です" );
} else {
  System.out.println( "部分一致ではありません" );
}
```

継承関係のプログラムを書いてみよう

- ClassExtA
  - methodA() println(“A is called”);
- ClassExtB extends ClassExtA
  - methodB() println(“B is called”);
- ClassExtC exntends ClassExtB
  - methodC() println(“C is called”);

- ClassExtD
  - main　ClassExtC c= new ClassExtC();
    c.methodC();

実験してみよう

- ClassExtAのインスタンスをClassExtCでキャストできるか
- ClassExtCのインスタンスを ClassExtA,ClassExtBでキャストできるか

- If文で演算子　instanceofで判定仕様