ネットワークプログラミング
第3回
2014/9/30

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

講義内容

第１回Java理解度チェック
第２回Javaプログラミング基礎2
第３回
TCP/IPの復習　TCPサーバ
第４回
TCPクライアント/サーバ通信　チャットプログラム
第５回
UDP通信
第６回
中間学力考査（持ち込み不可　紙提出）
第７回
スレッド基礎 サーバのスレッド化 マルチスレッド
第８回
デザインパターン
ファクトリメソッド　シングルトン
第９回
ノンブロッキングI/O　　Javaプログラミング応用
第１０回
マルチスレッド　スレッドプール
第１１回
Webクライアント
第１２回
WEBサーバ,プロジェクト設計
第１３回
プロジェクト実装1
第１４回
プロジェクト実装2
第１５回
学力考査（持ち込み可　プログラミング提出）

授業予定日日程

- http://www.soe.dendai.ac.jp/kyomu/portal/2014_schedule_t.pdf

- スケジュール †
- (1)9/17 第1回　Java理解度チェック
- (2)9/24 第2回　Javaプログラミング基礎1
- (3)10/1 第3回　Javaプログラミング基礎2　TCP/IPの復習　TCPサーバ
- (4)10/8 第4回　TCPクライアント/サーバ通信　チャットプログラム
- (5)10/15 第5回　UDP通信
- (6)10/22 第6回　中間学力考査（持ち込み不可　紙提出）

概要

- クライアント／サーバモデル、TCP/IPネットワークのアプリケーションプログラミングインタフェースの基本および、ネットワークアプリケーションを効率的に動作させるためのマルチスレッドプログラミングを講義する。この基本の後、チャット等の対話型アプリケーション、Twitter4J等のアプリケーション開発の実例を講義する。

ゴール

- 通信ネットワークを利用したアプリケーションソフトウェアを、TCP/IP を意識したレベルで作成できる力を養成することを目標とする。

コモンズの悲劇

- **コモンズの悲劇**（コモンズのひげき、The Tragedy of the Commons）とは、多数者が利用できる共有資源が乱獲されることによって資源の枯渇を招いてしまうという経済学における法則。**共有地の悲劇**ともいう。
- 生態学者ギャレット・ハーディン（1915-2003年）が 1968年に『サイエンス』誌に論文「The Tragedy of the Commons」を発表したことで、一般に広く認知されるようになった。

プログラムの実行

java Hello

- Hello.java を編集して表示される文字を変えてみよう

Hello.java

```
public class Hello {

    public static void main(String[] args) {

System.out.println(“Hello! ¥”岩井くん¥" ¥n ");
System.out.print("A");
System.out.print("B");
System.out.print("C");
    }
}
```

Java プログラムの基本

クラス

```
public class Hello {

    public static void main(String[] args){

    // この中に、処理内容を書きます

    }
}
```

メソッド

注意1: Hello の部分がプログラム名
注意2: String は、文字列。String[] 文字列の配列
注意3: args は、コマンド引数の配列
　　args[0], args[1]

基本型

- boolean　論理型　(true または false)
- char　整数型(文字型)　(0以上65535以下)
- byte　整数型　(-128から127まで)
- int　整数型　(符号付き32ビット)
- long　整数型　(符号付き64ビット)
- float　実数 (単精度浮動小数点型)
- double　実数 (倍精度浮動小数点型)

エラーを修正しましょう。

- Hello2.java
- Hello2d.java
- Hello3.java
- Hello4.java

文字列連結の練習

- 一行づつコメントアウトしながら実行してみよう。

- SystemOutTest

文字列連結の練習

```
System.out.println("1------------------------");
System.out.print('a');//空の文字列を描画
//System.out.println("2------------------------");
//System.out.print("");//空の文字列を描画
//System.out.println("3------------------------");
//System.out.print(" ");//文字列スペースを描画
//System.out.println("4------------------------");
//System.out.print("b");//文字列bを表示
//System.out.println("5------------------------");
//System.out.print("¥n");//改行文字列を表示
//System.out.println("6------------------------");
//System.out.print("c¥n");//cと改行文字を表示
//System.out.println("7------------------------");
//System.out.println();//改行コードを描画
//System.out.println("8------------------------");
//System.out.println("");//カラ文字列と改行コードを描画
//System.out.println("9------------------------");
//System.out.println(" ");//スペース文字列と改行コードを描画
//System.out.println("10------------------------");
//System.out.println("d");//文字列dと改行コードを描画
```

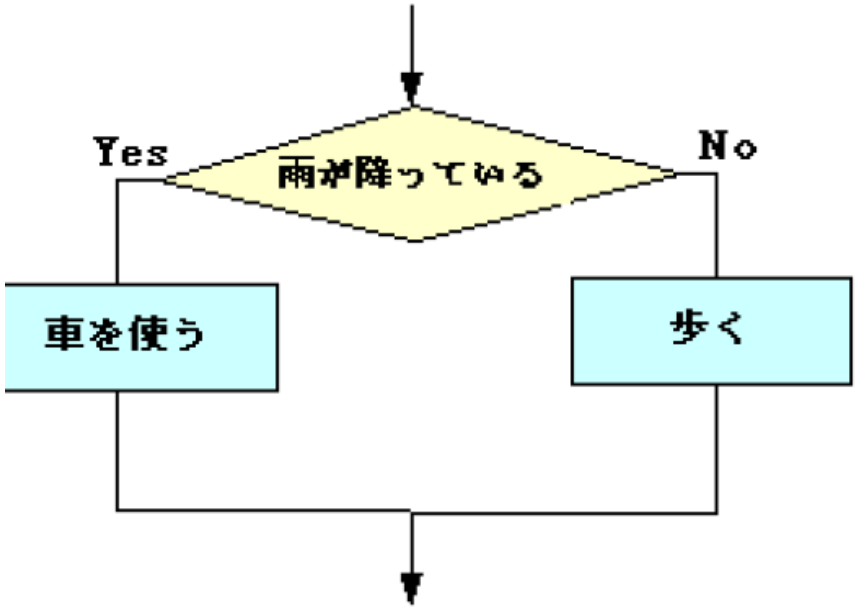
```
System.out.println("11----------------------");
System.out.println("e"+"f");//文字列eと文字列fを連結して描画ef　+は連結
System.out.println("12----------------------");
System.out.println("cat"+"dog"+"bird");//文字列catと文字列dogなどを連結して描画
System.out.println("13----------------------");
System.out.println("gh"+"¥n"+"ij");//文字列abを描画し改行コード　次の列に"cd"を描画

System.out.println("14----------------------");
String s="this is a pen";
System.out.println(s);//文字列変数sを描画し改行コード
```

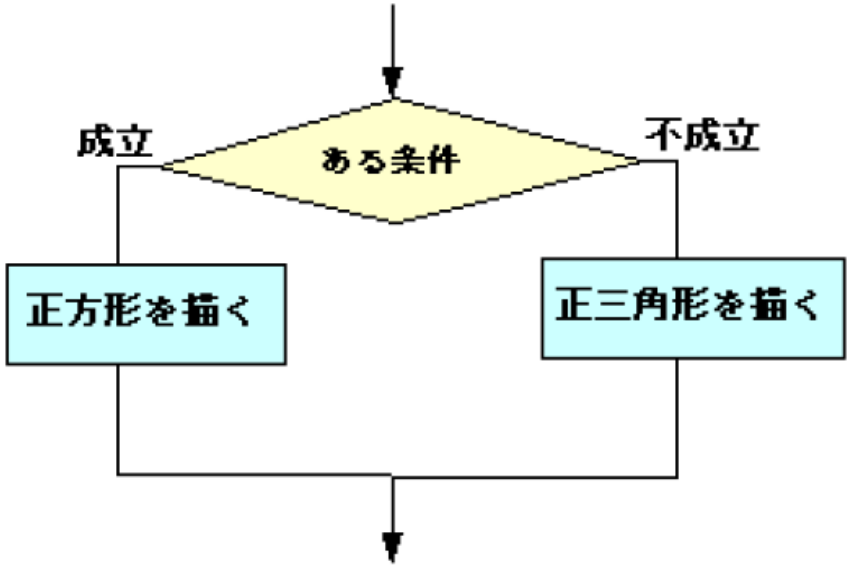
HowOldAreYou

```
int age = Integer.parseInt(line);
System.out.println("あなたは" + age + "歳ですね。");
System.out.println("あなたは10年後、" + (age + 10) + "歳ですね。");
```

宿題の解答例
HowOldAreYou2.java

条件分岐



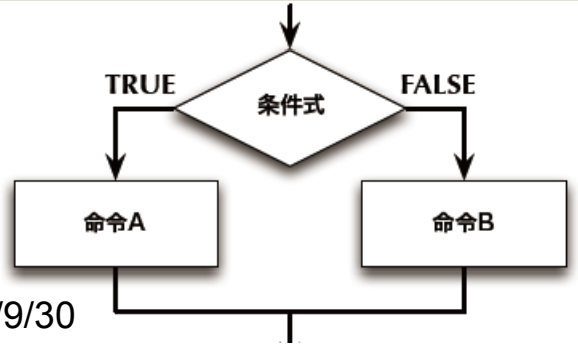


```
if(条件式){
    条件が成立した場合に行う処理
}
else{
    それ以外（条件が不成立）の場合に行う処理
}
```

```
if(条件式1){
    条件1が成立した場合に行う処理
}
else if(条件式2) {
    それ以外で条件2が成立の場合に行う処理
}
else{
    それ以外（条件が不成立）の場合に行う処理
}
```

条件分岐 (if文)

```
if(rain==1) {
   goByBus();
}
else {
   goByBicycle();
}
```



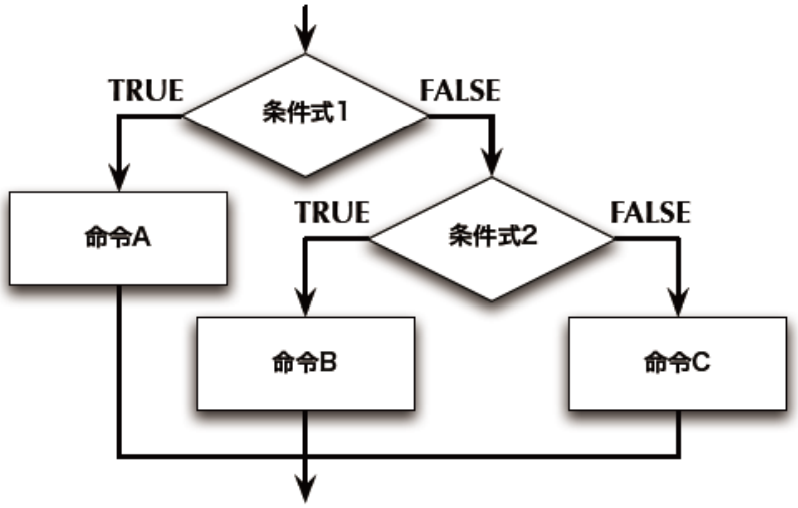

```
if(条件式) {
   条件式が成り立つ
   場合実行する処理
}
else {
   条件式が成り立たない
   場合実行する処理
}
```

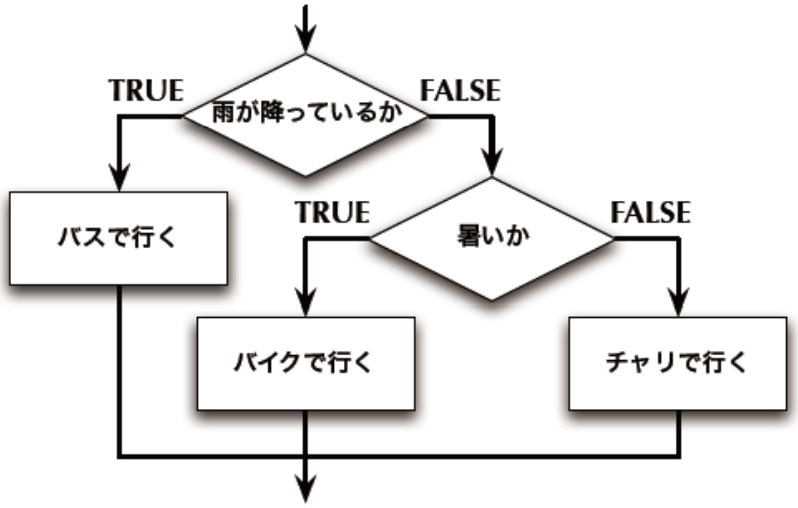
変数の比較

左辺 >= 右辺	≧	左辺の値が右辺の値以上
左辺 > 右辺	>	左辺の値が右辺の値より大きい
左辺 <= 右辺	≦	左辺の値が右辺の値以下
左辺 < 右辺	<	左辺の値が右辺の値より小さい
左辺 == 右辺	=	左辺の値と右辺の値が等しい
左辺 != 右辺	≠	左辺の値と右辺の値が等しくない

多重分岐

- 3つ以上分岐するには？
  - 雨だったらバス、雨でなくても暑かったらバイク、それ以外ならチャリ





多重分岐 (else if)

```
if(rain==1) {
  goByBus();
}
else if(hot==1) {
  goByBike();
}
else {
  goByBicycle();
}
```

```
if(条件式1) {
 条件式1が成り立つ場合
 実行する処理
}
else if (条件式2) {
 条件式1が成り立たないが
 条件式2が成り立つ場合
 実行する処理
}
else {
 条件式1も条件式2も
 成り立たない場合実行する処理
}
```

if文

```
if (条件式) {
    条件が成り立つときの処理
}
```

条件式は boolean型

その値は、trueかfalse

比較演算子

例　if (flag == true)　flagはboolean型変数

```
if ( p == 50)
if ( p != 50)
if (p >= 50)
if ( p < 50)
```

if … else

```
if (条件) {
        条件が成り立つときの処理
    }
else {
        条件が成り立たないときの処理
}
```

booleanの演算(かつ)

かつ　&&

条件式1 &&　条件式2

true&&true -> true

true&&false->false

false&&true->false

false&&false->false

booleanの演算(または)

または ||

条件式1 || 条件式2

true||true -> true

true || false->true

false || true-> true

false || false->false

条件式

```
if(          )    //iが3なら
if(          )    //iが10以上なら
if(          )    //iが10未満なら
if(          )    //iが偶数なら
if(          )    //iが奇数なら
if(                ) // iが6以上12以下
if(                ) // iが6未満12より大
```

IfTest0

- を実行して真偽値の演算を確かめてみよう。

IfTest1

- 引数5以上でbignumberと表示

IfTest2

- //xが7,8,9ながbignumber
- //xが4,5,6ならmiddle number
- //xが1,2,3ならsmall numberを表示するプログラムを書け

IfTest3

- //xが7,8,9ながbignumber
- //xが4,5,6ならmiddle number
- //xが1,2,3ならsmall numberを表示するプログラムを書け
- //xがそれ以外(10や-1なら)ならout of scopeと表示するプログラムをかけ

問題１

- 平均が80点以上なら「よくできました」 ８０点以下なら「がんばりましょう」 と表示するプログラムを作ろう

if … elseの連鎖

```
if (条件式1) {
        条件1が成り立つときの処理
    }
else  if  (条件式2) {
        条件1が成り立たず、
        条件2が成り立つときの処理
}
else  {
       条件1も、条件2も成り立たないときの処理
}
```

問題２

- 平均が１００点以上なら「点数が間違っています」、平均が80点以上なら「よくできました」８０点以下なら「がんばりましょう」 と表示するプログラムを作ろう

- else if を使う

または、かつ

または ||

条件式1 || 条件式2

かつ &&

条件式1 && 条件式2

例

((p > q) && (r < s)) || ((p < q) && (r < s))

問題３

- 平均が１００点以上または ０点未満なら「点数が間違っています」、平均が80点以上なら「よくできました」 ８０点以下なら「がんばりましょう」 と表示するプログラムを作ろう

- || をつかう

switch … case

```
switch (式) {
  case 定数式1:
          処理1
          break;
  case 定数式2:
          処理2
         break;
  case 定数式3:
          処理3
          break;
  default:
        処理
}
```

問題3.5

１～６を入力する
１がでたら、「一の目です」
２がでたら、「二の目です」
３がでたら、「三の目です」
４以上がでたら、「それ以外の目です」
と表示する

Switch 文を使う

問題４

さいころプログラムを作ろう

１がでたら、「一の目です」

２がでたら、「二の目です」

３がでたら、「三の目です」

４以上がでたら、「それ以外の目です」

と表示する

- Math.random()　をつかう
  - 0.0以上1.0未満の値の中からランダムな値を算出します。

問題５

- 引数に入れた2つの数字を加算するプログラム
  - Calc2
- ２つの引数以外を入れているとエラーメッセージがでるようなプログラムをつくる

引数に関して復習

```
public class Hikisu2 {
    public static void main(String[] args) {
        System.out.println(“1番目の引数は” + args[0] + "です。");
        System.out.println(“1番目の引数は” + args[1] + "です。");
        }
    }
}
```

繰り返し (for文)

```
int i;
for(i=1; i<=4; i=i+1) {
   内容
}
```

```
変数の宣言;
for(初期化式; 条件式; 増加式) {
   内容
}
```

繰り返し (for文)

- 変数の宣言
  - 繰り返しの回数をメモしておく変数を用意する
- 初期化式
  - 変数の最初の数字は何か
- 条件式
  - 変数がどうなっている間、続けるか
- 増加式
  - 一回繰り返すごとに、変数をどうするか

```
変数の宣言;
for(初期化式; 条件式; 増加式) {
   内容
}
```

繰り返し (for文)

- 変数の宣言
  - 整数型の名前がiという変数を用意
- 初期化式
  - 変数iの最初の数字は1
- 条件式
  - 変数iが4以下の間、続ける
- 増加式
  - 一回繰り返すごとに、変数iに1を足したものを、変数iに入れる

```
int i;
for(i=1; i<=4; i=i+1) {
   内容
}
```

繰り返し (while文)

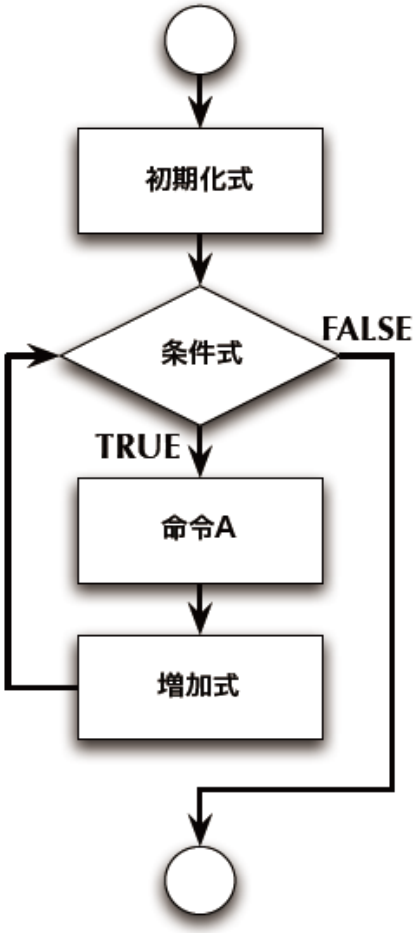
```
int i;
i=1;
while(i<=4) {
   内容
   i=i+1;
}
```

```
変数の宣言;
初期化式;
while(条件式) {
   内容
   増加式;
}
```

フローチャート（繰り返し）

```
int i;
for(i=1; i<=4; i=i+1) {
   命令A
}
```

```
変数の宣言;
for(初期化式; 条件式; 増加式) {
   命令A
}
```



省略演算

- 足し算
  - i=i+1;
  - i+=1;
  - i++;
- 引き算
  - i=i-1;
  - i-=1;
  - i--;
- 掛け算
  - i=i*2;
  - i*=2;
- 割り算(切捨て)
  - i=i/10;
- あまりの計算
  - p=i%2;
  - もしiが偶数ならp==0;
  - 奇数ならp==1

for文

```
for ( 初期化; 条件式; 次の一歩 ) {
        繰り返す処理
}
```

```
for (int i = 0; i < 3; i++) {
   System.out.println(i);
}
```

により、

0

1

2

が表示される。

変数の有効範囲(スコープ)

```
for (int i = 0; i < 3; i++) {
    System.out.println(i);
  }
  System.out.println("i = " + i):
```

← iは有効範囲外なので
コンパイルエラー。

解決策

```
int i;
for (i = 0; i < 3; i++) {
    System.out.println(i);
}
System.out.println("i = " + i):
```

while 文

```
while (条件式) {
   繰り返す処理
}
```

null (ナル、ヌル)の意味

```
while (line != null) {
```

null　入力の終わりに達したときの特殊なオブジェクト

繰り返し文の練習問題

1 から 100 までの整数を足し合わせる

(1) その1:　for文を使う

(2) その2:　while文を使う

CountTest.java

WhileTest.java

次週以降

CountTenRunnable.java

CountTesterTweThreads.java

Class

クラスは、物の設計図。
　　中に変数やメソッドが定義される。

オブジェクトは、クラス定義に基づく実際の物。プログラム上は、変数。

例：Automobile というクラスを定義する。
　　Automobileクラスで、volkesWagen, audi, volvo というオブジェクトを作る。

method

車というクラスを定義する。メソッドとして、

　　乗る、止める

を用意する。

ポルシェというオブジェクトを車というクラスで生成すると、

　ポルシェ. 乗る、

　ポルシェ. 止める

というメソッドが使える。

定義する場所

クラス

```
返り値  メソッド1 {
  XXXXXXXXXX
}
```

```
返り値  メソッド2 {
  XXXXXXXXXX
}
```

```
返り値  メソッド3 {
  XXXXXXXXXX
}
```

```
返り値  メソッド4 {
  XXXXXXXXXX
}
```

いくつでも
作ってよい。

public ?

クラス

```
public 返り値  メソッド1 {
  XXXXXXXXXX
}
```

```
public  返り値  メソッド2 {
  XXXXXXXXXX
}
```

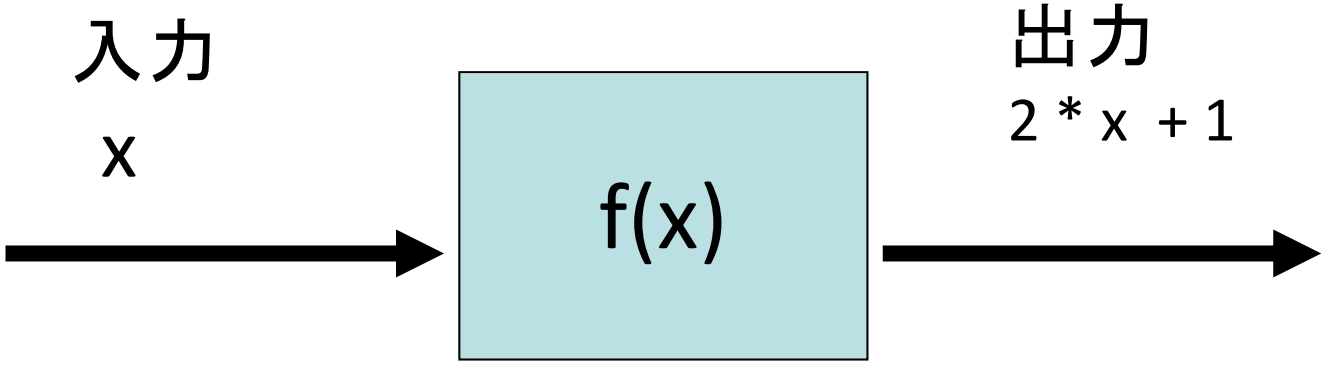
```
返り値  メソッド3 {
  XXXXXXXXXX
}
```

```
返り値  メソッド4 {
  XXXXXXXXXX
}
```

関数

- 関数

f(x) = 2 * x + 1

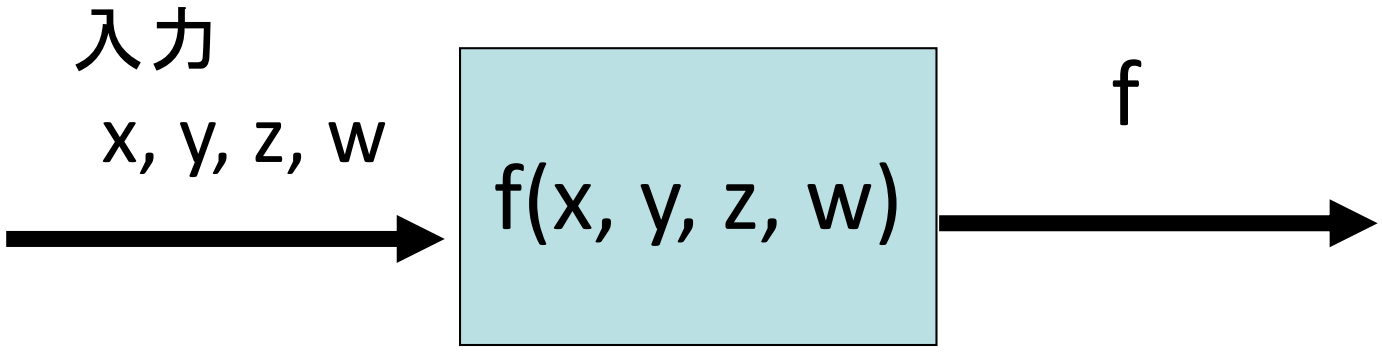


関数の定義部分

しかし、関数の入力はたくさん あってよい。

- 関数

f(x, y, z, w) = 2 * x + y - z
+ ∟w」



関数の定義部分

メソッドで引数がたくさんあるとき

```
int   calcComplex(int x, int y, int z,
                  float w) {
      if ( x > y ) {
        return z;
      } else {
        return (int)w;
}
```

メソッド分け

- 合成関数

  f(x) = 2 * x + 1

  h(x) = 3 * (2 * x + 1) + 5

のとき、 h(x) = (g ○ f) (x)

```
int  h(int x) {
  return 3 * (2 * x + 1) + 5;
}
```

```
int  h(int x) {
  return 3 * g(x) + 5;
}

int  g(int x) {
  return 2 * x + 1;
}
```

Javaプログラミングも同じ。メソッドとして
独立させた方がよいかどうか、よく考える。

メソッドの形式

公開するか否か

クラスメソッドとする

戻り値の型

```
public static int メソッド名(引数宣言) {

        メソッドの中身

        return (戻り値);
}
```

void

関数によっては、返り値がいらないものもある。そのときには、返り値なし (void) を指定する。
前回作成した、drawBar に返り値は必要なかった。

引数がない場合もある。

型

int　整数
float　浮動小数点数(実数)
char　文字型

等

メソッドの引数

```
戻り値　メソッド名(型　変数名1,
                  型　変数名2,
                  型　変数名3,
                  型　変数名4
                    ..............)　{

    メソッドの本体


}
```

Javaのメソッドの引数

戻り値　メソッド名(型　変数名1,
型　変数名2,
型　変数名3,
型　変数名4
.....)　{

メソッドの本体

}

クラス（既に定められたものでも、
自分で定めたものでも) の名前でもよい

メソッド呼び出し

本来は、

g.drawString(XXXXXXXXXXXXX);

のように、

オブジェクト.メソッド名(引数…);

と書く。

メソッド呼び出し(2)

しかし、自分で定義したクラスの中の
メソッドを呼び出すときは、

　　オブジェクト.

なしに、

　　メソッド名(引数...);

でよい。

例:

　　drawBar(XXXXXXXXXXX);

methodとクラス

- Heikin.java と Kamoku.java

- Heikin と Kamoku クラスを作る
  - public class Heikin
  - class Kamoku
- Heikin クラス
  - Kamokuクラスのインスタンスとして、englishとmath を作る
  - english の name に "英語" を設定する
  - english の score に 80 を設定する
  - math　も english と同様に (name→数学, score→70）
  - 英語と数学のscore を読み出して、平均値を表示する
- Kamoku クラス
  - String name
  - setScore というメソッドを定義する。score に値を設定する。
  - getScore というメソッドを定義する。scoreを返す。

定数の宣言

C++/C では、#define 文を使用した。
(例)

```
#define   WIDTH     80
```

Javaでは、final static で修飾する。
(例)

```
public final static  int  WIDTH  = 80;
public final static  String  school = "dendai";
```

プロトコル
TCP/IP

ドメイン名

- IPアドレスのかわりに用いる識別名

www.im.dendai.ac.jp

ーサーバの種類

ー組織名称

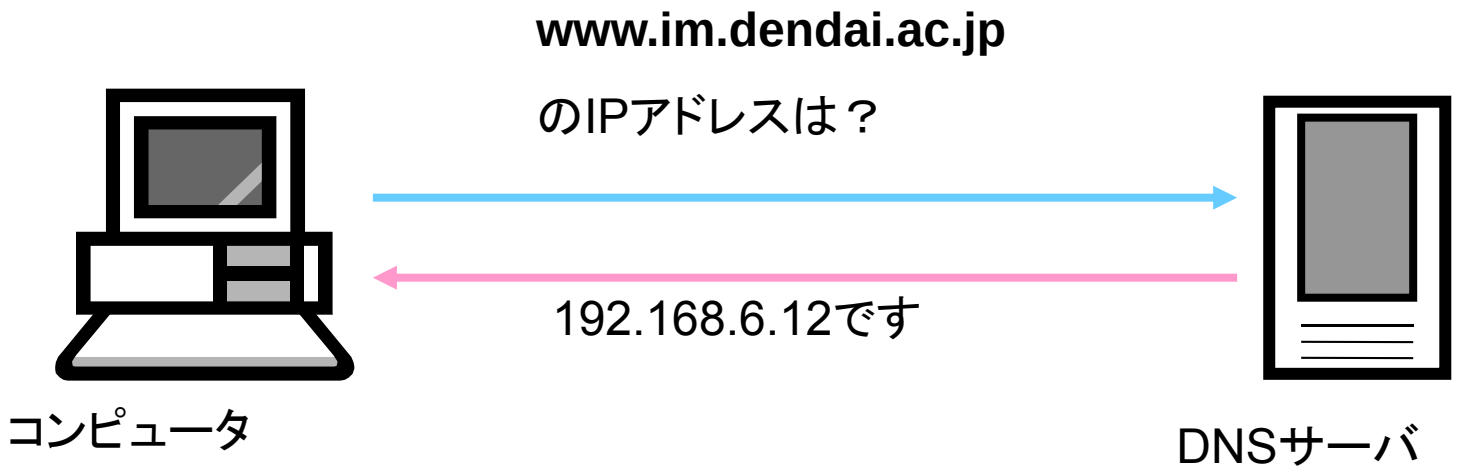
ー組織種別

ー国

ピリオドで分かれている

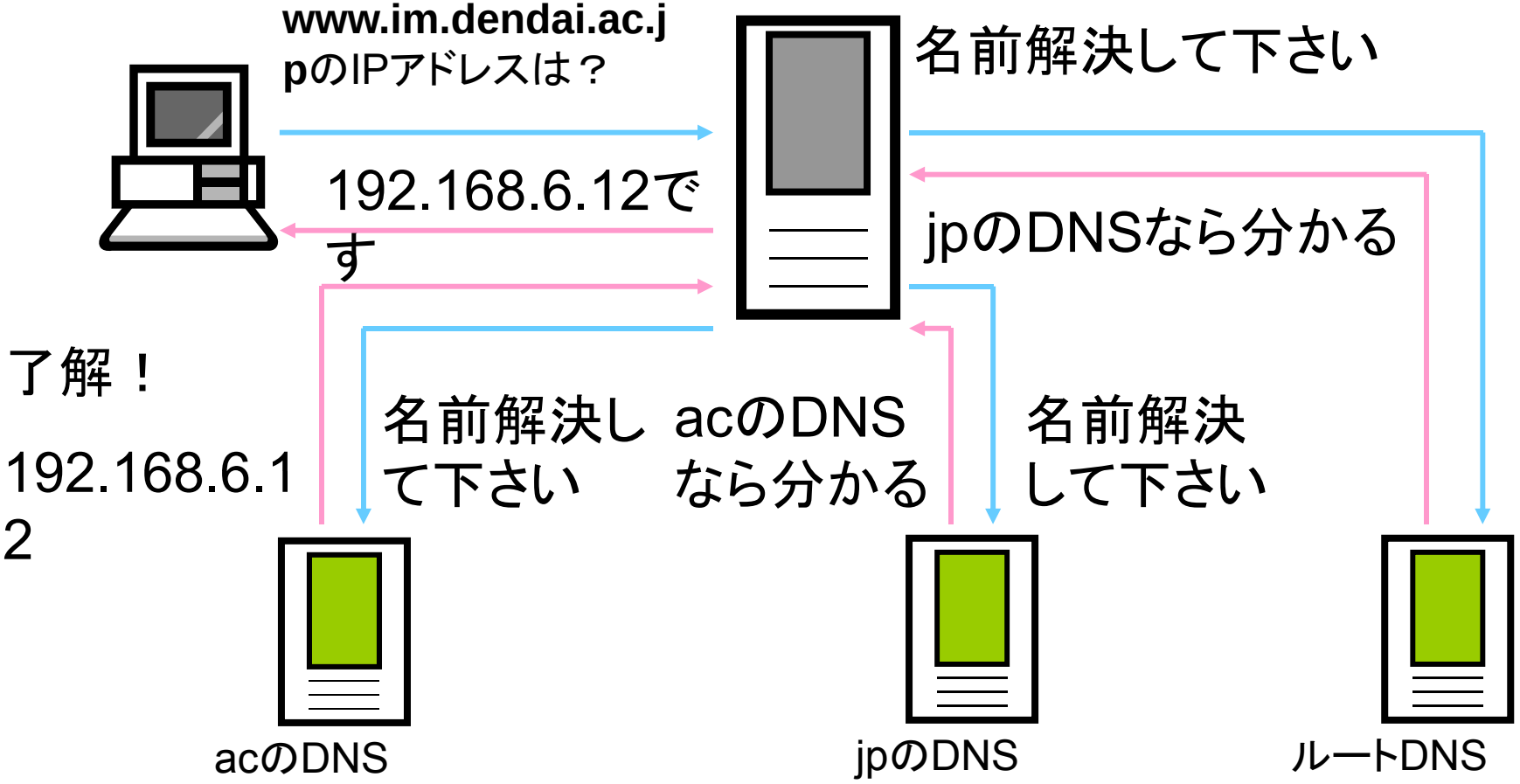
DNS

- ドメイン名とIPアドレスを対応付ける



DNS②

- DNSの仕組み



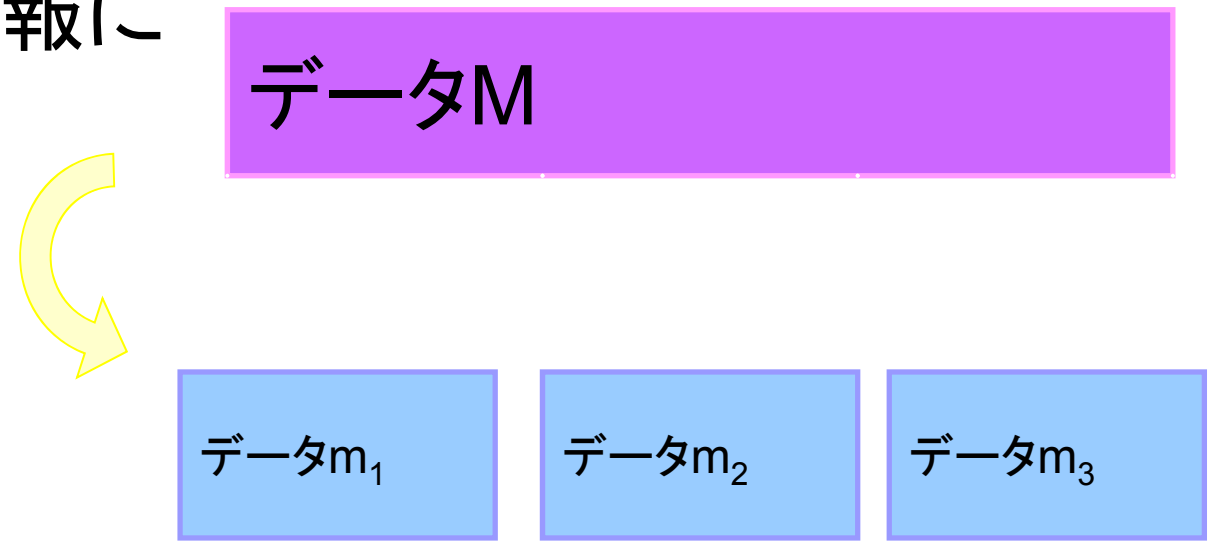
トランスポートプロトコル

- 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

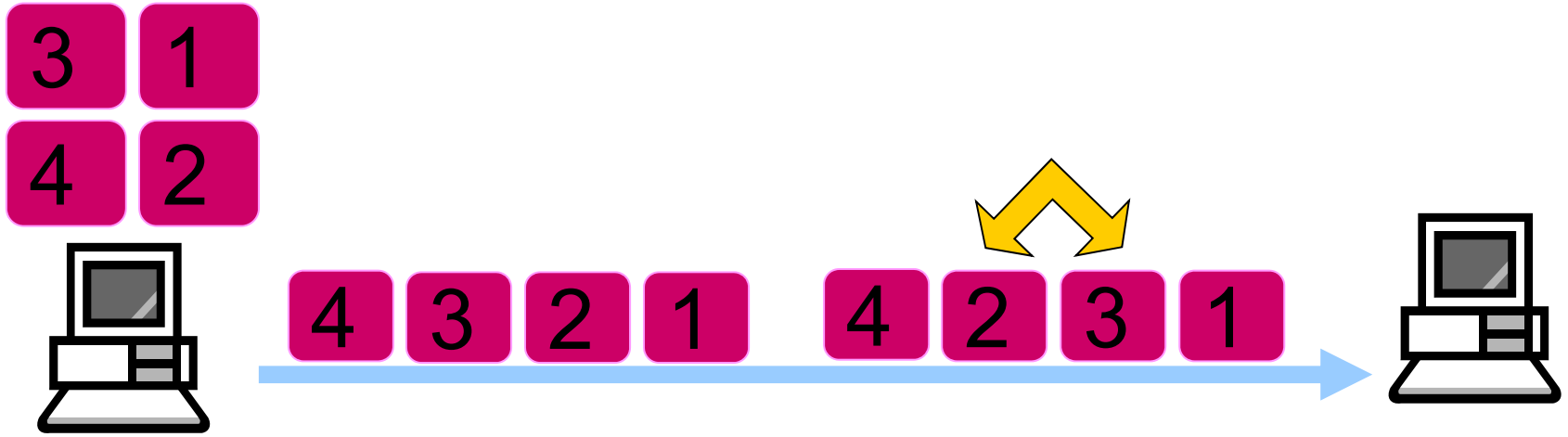
長いデータは通信できない

小さな複数の情報に分割

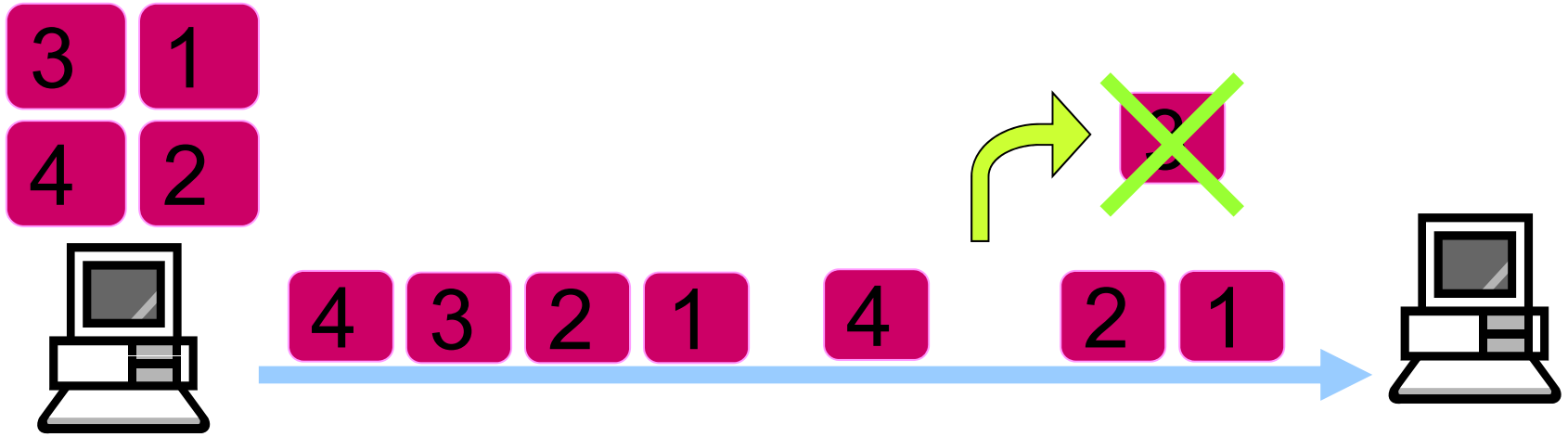


パケット通信の問題点

- パケットの順番



- パケットの欠落



TCP

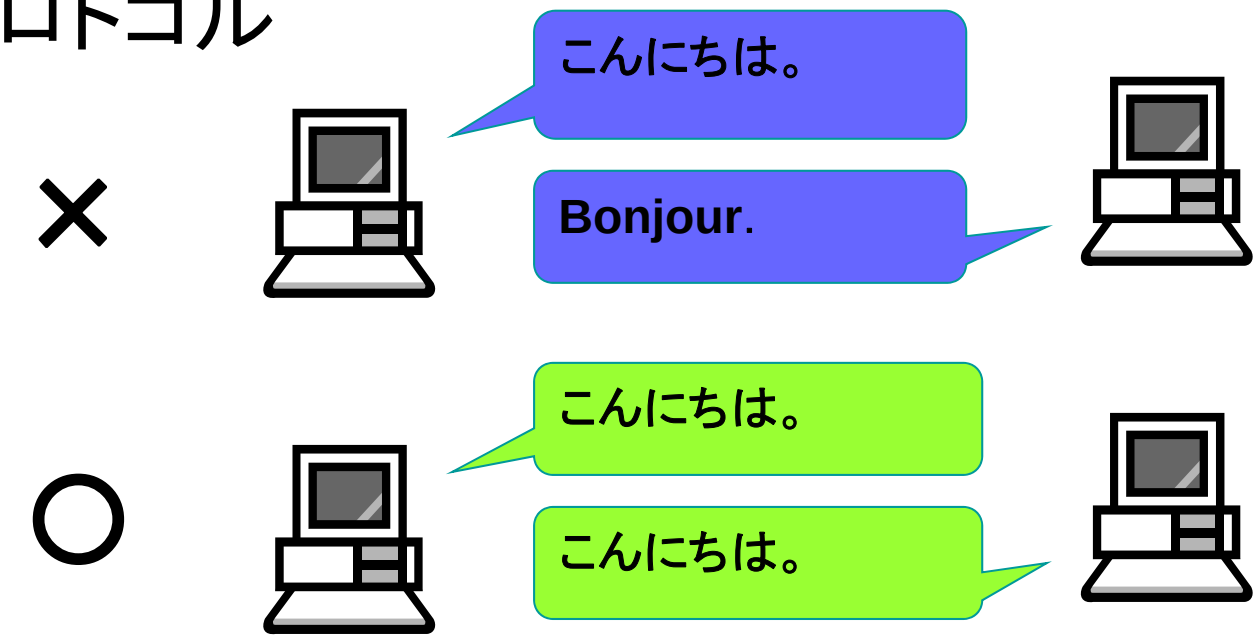
- TCPの主な特徴を以下に記す。
- パケットの破壊、重複、喪失、順序の入れ替わりをチェックする。また応答確認、再送信を行い、通信の信頼性を高める。
- ホスト間で通信する前に仮想的な通信路を作る（コネクション指向）。
- 始点と終点間で多重化を行うことで、同時に複数の通信が行うことができる。
- 送信のフローコントロール制御を行う。

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



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

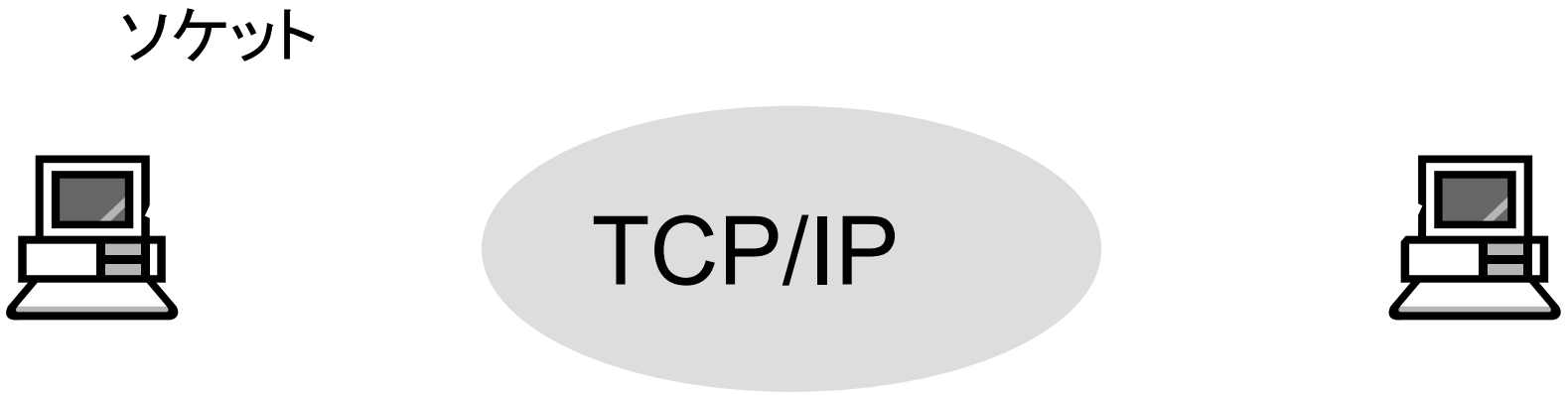
- ソケットを使ってデータの送受信
- ソケットを使った通信

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

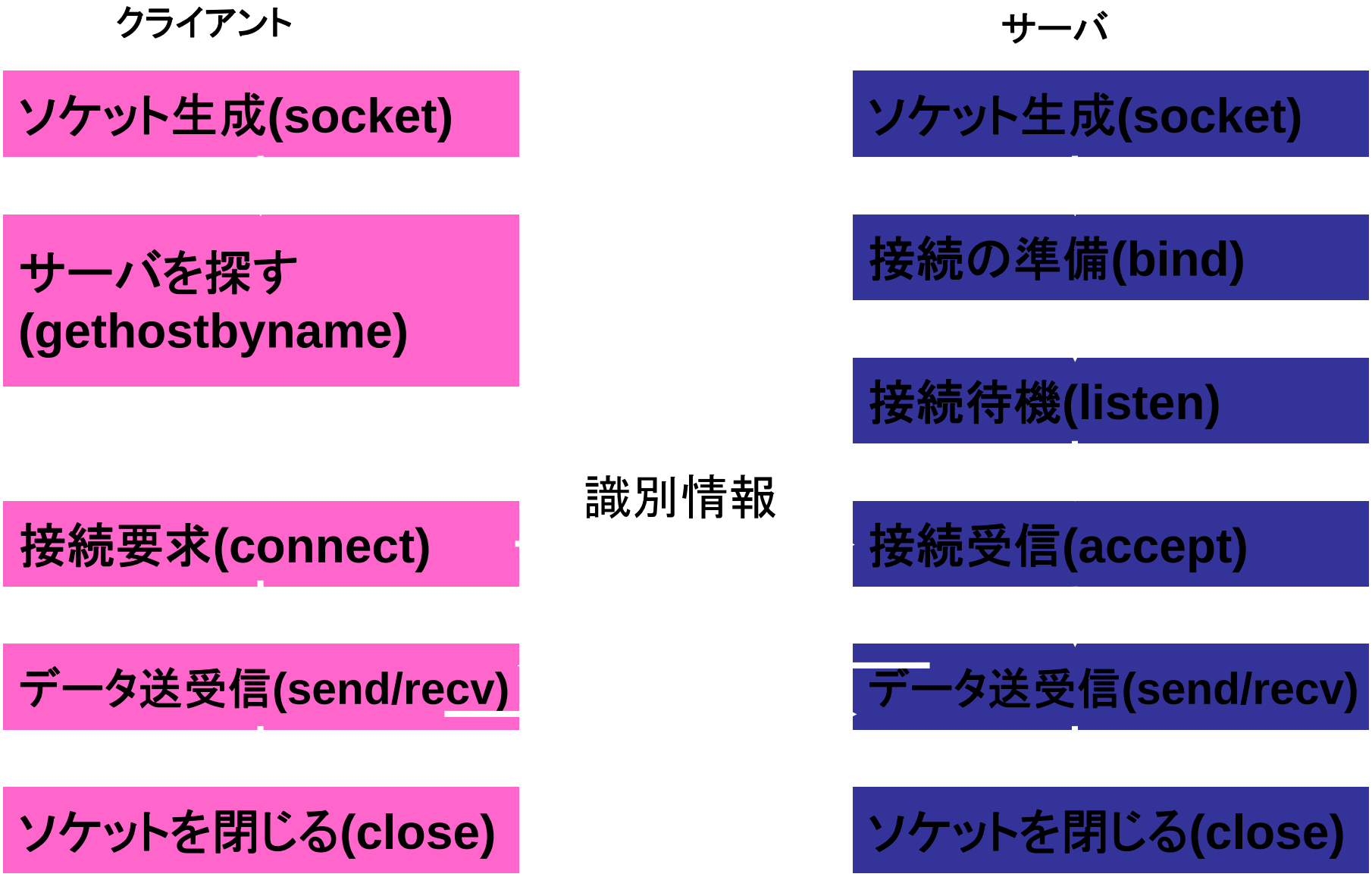
クライアントプログラム

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();
```

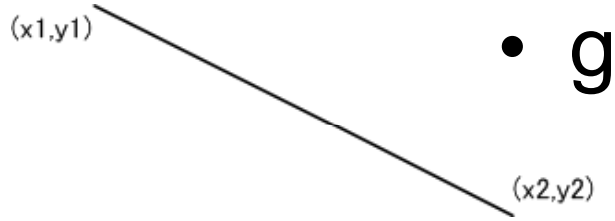
Graphics

Graphics というクラスには、
　drawString, drawCircle 等の
メソッドが定義されている。

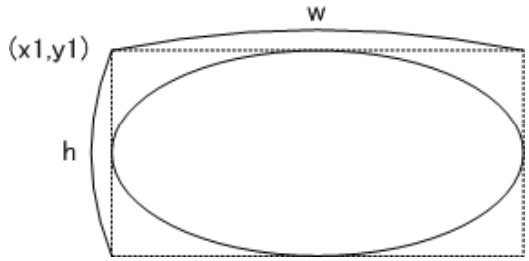
Graphics クラスである g という
オブジェクトに対して、
　g.drawString、
　g.drawCircle
という形でメソッドを呼び出せる。

Graphicsのmethod

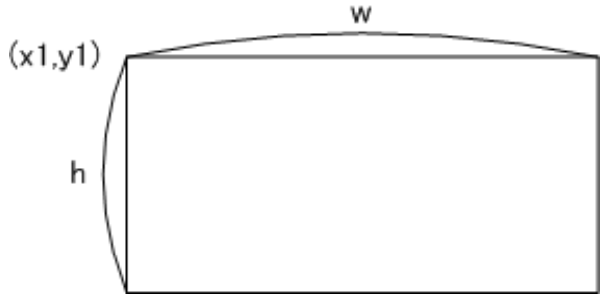
- g.drawLine(x1,y1,x2,y2);



- g.drawOval(x,y,w,h);

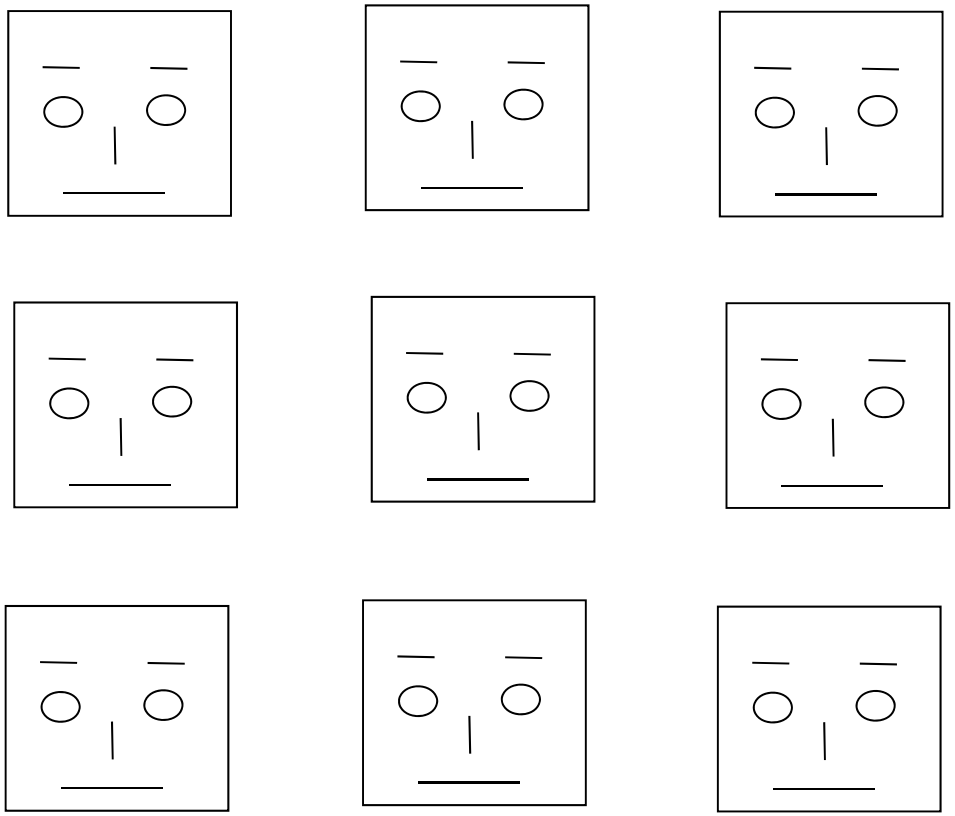


- g.drawRect(x,y,w,h);



今日の課題

課題faceを沢山つくってみよう
Face.javaを改造してFaces.java
を提出してください。
課題faceを沢山つくってみようFace.javaを改造して
Faces.javaを提出してください。二重の繰り返し文を
用いること。例



Face ヒント1

```
void   drawFace(Graphics g,
                int  xStart,
                int  yStart)  {
```

　左隅の座標を (xStart, yStart) として、一つの顔を描くメソッドを記述する。

```
}
```

Faceヒント2

paint メソッドの中には、

```
   for(int i = 0; i < 3; i++) {
      for(int j=0; j < 3; j++) {
         drawFace(g, 20 + 80 *i,
                              20 + 80 *j);
      }
 }
```

80 という数字は仮。顔の大きさを考えて計算する。

しかし、数字決め打ちは避けたい

```
  for(int i = 0; i < 3; i++) {
     for(int j=0; j < 3; j++) {
        drawFace(g, 20 + step *i,
                          20 + step *j);
     }
}
```

眉毛や鼻の形を自由に変えたい

```
void   drawFace(Graphics g, int  xStart,  int  yStart)  {
      drawFrame(g, xStart, yStart);
      drawEyeBrow(g, xStart, yStart);
      drawEye(g, xStart, yStart);
      drawNose(g, xStart, yStart);
      drawMouth(g, xStart, yStart);
}

void drawFrame(Graphics g, int xStart, int yStart) {
   記述
}
void drawEyeBrow(Graphics g, int xStart, int yStart) {
   記述
}
void drawEye(Graphics g, int xStart, int yStart) {
   記述
}
void drawNose(Graphics g, int xStart, int yStart) {
   記述
}
void drawMouth(Graphics g, int xStart, int yStart) {
   記述
}
```

さらに内部で
メソッドに分ける。

MovingBall
Thread,Runnable
RandSwitch
配列

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};