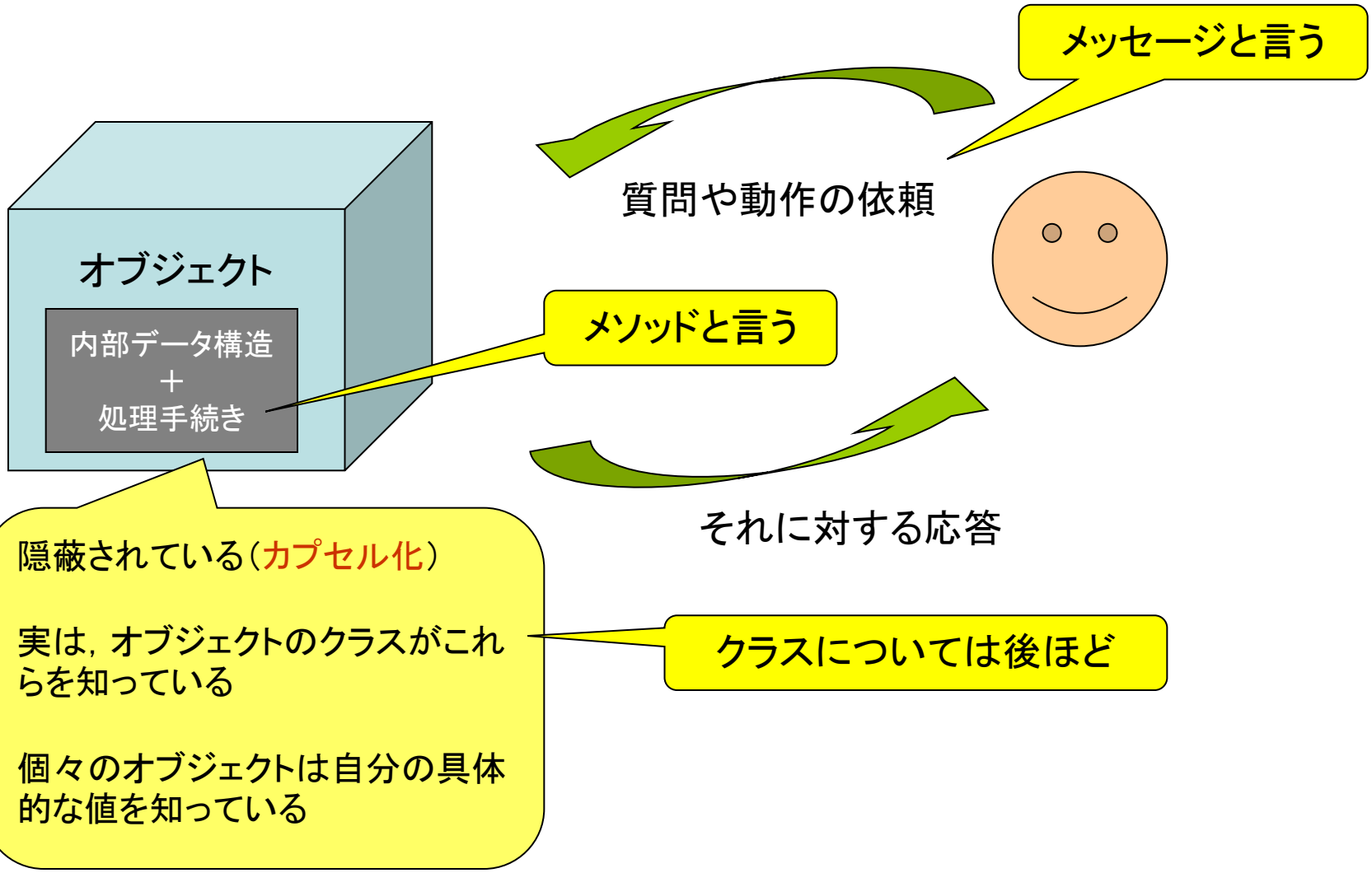
オブジェクト指向

簡単なネットワーク・プログラミング

```
irb(main):005:0> a = [1,2,3]
=> [1, 2, 3]
irb(main):006:0> a.length()
=> 3
irb(main):007:0> a.at(2)
=> 3
irb(main):008:0> a.concat([4,5])
=> [1, 2, 3, 4, 5]
irb(main):009:0> a
=> [1, 2, 3, 4, 5]
```

オブジェクト


メッセージと言う
質問や動作の依頼
オブジェクト
内部データ構造
＋
処理手続き
メソッドと言う
それに対する応答
隠蔽されている（カプセル化）
実は，オブジェクトのクラスがこれらを知っている
個々のオブジェクトは自分の具体的な値を知っている
クラスについては後ほど

```
irb(main):005:0> a = [1,2,3]
=> [1, 2, 3]
irb(main):006:0> a.length()
=> 3
irb(main):007:0> a.at(2)
=> 3
irb(main):008:0> a.concat([4,5])
=> [1, 2, 3, 4, 5]
irb(main):009:0> a
=> [1, 2, 3, 4, 5]
```

配列もオブジェクト

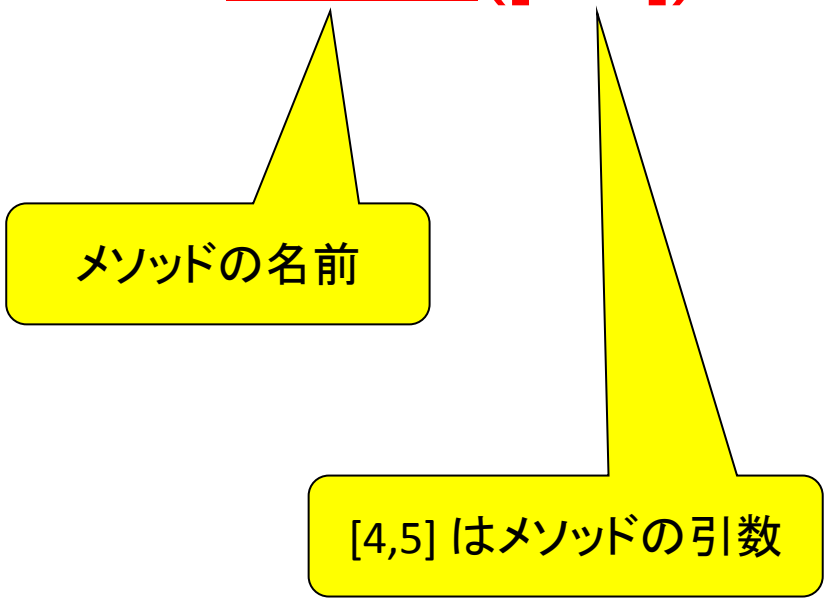
あなたの長さは何？

あなたの2番目の要素は何？

[4,5]をくっつけなさい！

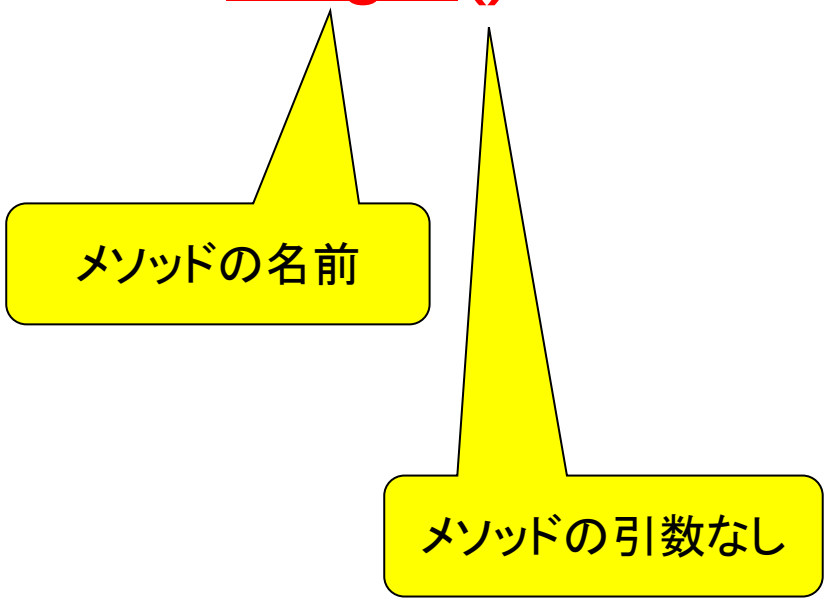
メッセージ

irb(main):008:0> a.concat([4,5])



メッセージ

irb(main):006:0> a.length()



ハッシュ（テーブル）

```
irb(main):011:0> h = Hash.new()
=> {}
irb(main):012:0> h.store("hagiya", 3)
=> 3
irb(main):013:0> h.store("suzuki", 5)
=> 5
irb(main):014:0> h.length()
=> 2
irb(main):015:0> h.fetch("hagiya")
=> 3
```

Array

- 配列
- 作成

  Array.new(長さ)
- メソッド

  length()　　長さを返す

  at(位置)　　位置の要素を返す

  concat(配列)　配列の要素を追加する
- 実は...

Hash

- ハッシュ（テーブル）
- 作成

  Hash.new()
- メソッド

  store(キー, 値)　　キーの値を登録する

  fetch(キー)　　キーの値を取り出す

  fetch(キー, デフォルト)　　キーの値を取り出す
  登録がなければデフォルト

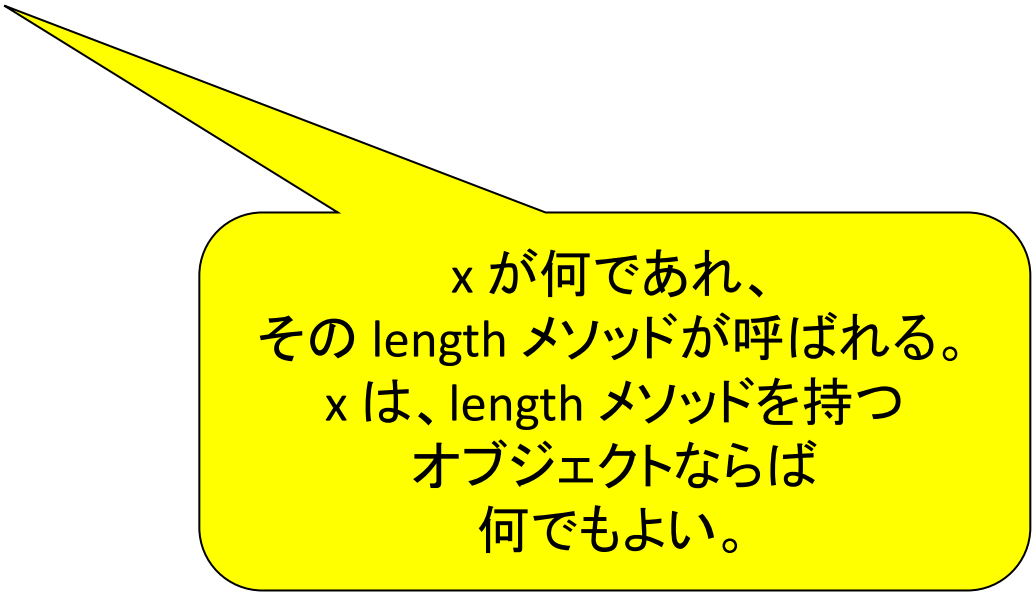
  length()　　登録数を返す

やってみよう

```
h = Hash.new()
h.store("hagiya", 3)
h.store("suzuki", 5)
h.fetch("sato", 0)
```

多相性

```
def empty(x)
  x.length() == 0
end
```



cc.rb

```
require 'socket'

def vote(sel, addr)
  s = TCPSocket.new(addr, 12345)
  s.puts(sel)
  r = s.gets()
  s.close()
  r
end
```

addr をアドレス、
12345 をポートとする
通信路（ソケット）を作って
変数 s に返す。

sel の後ろに
改行を付けて、
s に送信する。

s から1行受け取って、
変数 r に返す。
改行が付いているはず。

s を閉じる。

r を返す。

TCPSocket

- TCP/IP通信のための通信路（ソケット）
- 作成

  TCPSocket.new(アドレス, ポート番号)
- メソッド

  puts(文字列)　文字列を（改行を付けて）送信する

  gets()　一行を受信して文字列として返す

  close()　通信路を閉じる

TCPServer

- TCP通信の受け手（サーバソケット）
- 作成

  TCPServer.new(ポート番号)
- メソッド

  accept()　　クライアントからの
  コネクションを受けて、
  TCPSoket のオブジェクトを返す

やってみよう

- ウィンドウを二つ開け、それぞれ irb を起動。
- 両方で require('socket') とする。
- 片方で

  gs = TCPServer.new(12345)

  s = gs.accept()

  とする。待っているはず。
- もう片方で

  s = TCPSocket.new("localhost", 12345)

  とする。

やってみよう

- 片方で

  s.gets()
- もう片方で

  s.puts("hello")
- 繰り返してもよい。

やってみよう

- どちらでもよいので

  s.peeraddr[3]

  を評価してみよう。

- 終わったら両方とも irb を終了させる。

ss.rb

```
def collect()
  log = Hash.new()
  result = make1d(10)
  gs = TCPServer.new(12345)
  while true
    s = gs.accept()
    serve(s, log, result)
  end
end
```

12345 をポートとする
受け手（サーバソケット）を作って
変数 gs に返す。

gs を通して通信を一つ
受け付ける。
その通信のための
通信路（ソケット）を s に返す。

s との通信の面倒を
見る下請けの手続き

やってみよう

- 一つのウィンドウで irb を起動し

  ss.rb をロード

  collect() を実行

  （終了するにはコントロールC）
- 別のウィンドウで irb を起動し

  cc.rb をロード

  たとえば vote("3", "localhost") を実行

  もう一度 vote("5", "localhost")

さらに

- 隣の人の端末のIPアドレスを調べる。
  ifconfig というコマンドを用いる。
- そこに投票してみよう。
  たとえば vote("3", "133.11.70.129")
- IPアドレスの代わりに、端末の名前でもよいかもしれない。

サーバとクライアント

- インターネット上でサービスを提供する典型的な方法
- サービスを提供コンピュータ上でサーバと呼ばれるプログラムを走らせておく。
- サーバに仕事を依頼するプログラムをクライアントという。
- クライアントはサーバへの問い合わせや仕事の依頼を行う。
- サーバは所定の処理をして返答する。

進捗の確認

1. 他人が自分のサーバに投票できた。自分も他人のサーバに投票できた。
2. 自分のサーバに投票できた。
3. プログラムが動かない。
4. プログラムの動かし方がわからない。
5. プログラムがダウンロードできない。

ss.rb（続き）

```
def serve(s, log, result)
  addr = s.peeraddr[3]
  sel = s.gets()
  i = sel.to_i()
  if i < 1 || i > 9
    s.puts('error: illegal selection')
  else
    if log.fetch(addr, 0) != 0
      s.puts('updated')
      update(result, i, log.fetch(addr))
    else
      s.puts('ok')
      update(result, i, 0)
    end
    log.store(addr, i)
  end
  s.close()
end
```

クライアントの IP アドレス

s から1行受け取って、
変数 sel に返す。
改行が付いているはず。

sel を整数に変換。改行は無視される。

log（ハッシュ）から addr をキーとして探す。
addr がなければ 0（デフォールト）を返す。

```
if log.fetch(addr, 0) != 0
  s.puts('updated')
  update(result, i, log.fetch(addr))
else
  s.puts('ok')
  update(result, i, 0)
end
log.store(addr, i)
```

log から addr をキーとして探す。
（必ずあるはず。）

log の addr に i を登録する。

```
def update(result, n, o)
  result[n] = result[n] + 1
  if o != 0
    result[o] = result[o] - 1
  end
  show_result(result)
end
```

n の投票数を 1 増やし、
o の投票数を1 減らす。

o が 0 の場合は、
古い投票はない。

```
def show_result(result)
  puts("¥033[H¥033[J")
  puts('=====================================')
  for i in 1..9
    if result[i]>0
      puts(i.to_s() + ' ' + ('*' * result[i]))
    end
  end
  puts('=====================================')
end
```

puts は端末に文字列を改行を付けて出力する。

この文字列は、ウィンドウをクリアする。

'*' を result[i] 回繰り返す。

i を文字列に変換。

やってみよう

- 次の sss.rb は ss.rb から通信部分だけを抜き出したもの
- 一つのウィンドウで irb を起動し

  sss.rb をロード

  collect()
- 別のウィンドウで irb を起動し

  cc.rb をロード

  vote("3", アドレス)

  vote("la la lan", アドレス)

sss.rb

```
require 'socket'

def collect()
  gs = TCPServer.new(12345)
  while true
    s = gs.accept()
    serve(s)
  end
end
```

```
def serve(s)
  addr = s.peeraddr[3]
  p(addr)
  sel = s.gets()
  p(sel)
  s.puts('ok')
  s.close()
end
```

進捗の確認

1. 他人が自分のサーバに送信できた。自分も他人のサーバに送信できた。
2. 自分のサーバに送信できた。
3. プログラムが動かない。
4. プログラムの動かし方がわからない。
5. プログラムがダウンロードできない。

注意

- collect() はコントロールCで止めます。
- 再び collect() を呼ぶとエラーになることがあります。
- この場合は、irb をいったん終了して、もう一度、irb を起動し直してください。

実は

- Rubyではすべてのデータはオブジェクト
- たとえば数もオブジェクト
- 数を文字列に変換するには

  10.to_s()
- 逆に文字列を数（整数）にするには

  "13".to_i()

Fixnum

- 小さい整数
- メソッド

  to_s()　文字列に変換

  to_f()　浮動小数点数に変換
- 実は...

String

- 文字列
- 作成

  String.new()

  String.new(文字列)
- メソッド

length()	長さを返す
to_i()	対応する整数を返す
split(/¥s+/)	空白文字で区切って、 文字列の配列を返す

練習

- "Fri 13 Jan" という文字列から 13 を整数として取り出せ。

1. できた。
2. エラーが出てしまう。
3. どのメソッドを使えばよいか、わからない。
4. 何を期待されているのか、わからない。

去年のレポート

- 課題
  - cc.rb と ss.rb を拡張しよう。
  - cc.rb に次の関数を追加したい。
  - myvote(アドレス)
    - 自分が投票した内容を返してくれる。
    - もちろん vote は前のまま動くように。
  - count(選択肢, アドレス)
    - 選択肢の投票数を返す。
  - これらをサービスを提供するように ss.rb を拡張する。
- ヒント
  - サービスの「種類」も送ってやる。

ヒント（続き）

- 三種類のサービス
  - vote
  - myvote
  - count
- クライアントはサービスの「種類」も送る。
  - たとえば、送信する行の頭に何かを付ける。
  - たとえば、種類を指定する行を先に送信する。
- サーバはサービスの種類を判断する。
  - そして、種類に従った処理。

ヒント（続き）

- サーバが受け取った文字列を処理するため、文字列の処理が必要になるかもしれない。
- たとえば、変数 s に文字列が入っているとき

  s.to_i()

  s.length()

  s[i..j]

  s.split(/¥s+/)

  などを用いる。

今年のレポート

- 簡単なオークションサイトを作ろう。
- クライアントは次の操作を行えるとする。
  - start(初期値)
    - 指定された初期値でオークションを開始する。
  - close()
    - オークションを終了する。
    - start を実行したクライアントのみ実行可能とする。
    - 最高値を付けたクライアントの addr を返すとする。

今年のレポート（続き）

- クライアントの操作（続き）
  - current()
    - 現在の値を返す。
  - bid(値)
    - 指定された値でビッドする。
    - 現在の値より大きければ true が返る。
    - 現在の値以下ならば false が返る。

  各関数は、特に指定がない場合は、適当な値を返すものとする。

レポート

- 提出先

  is-komaba@lyon.is.s.u-tokyo.ac.jp
- Subject:欄は、

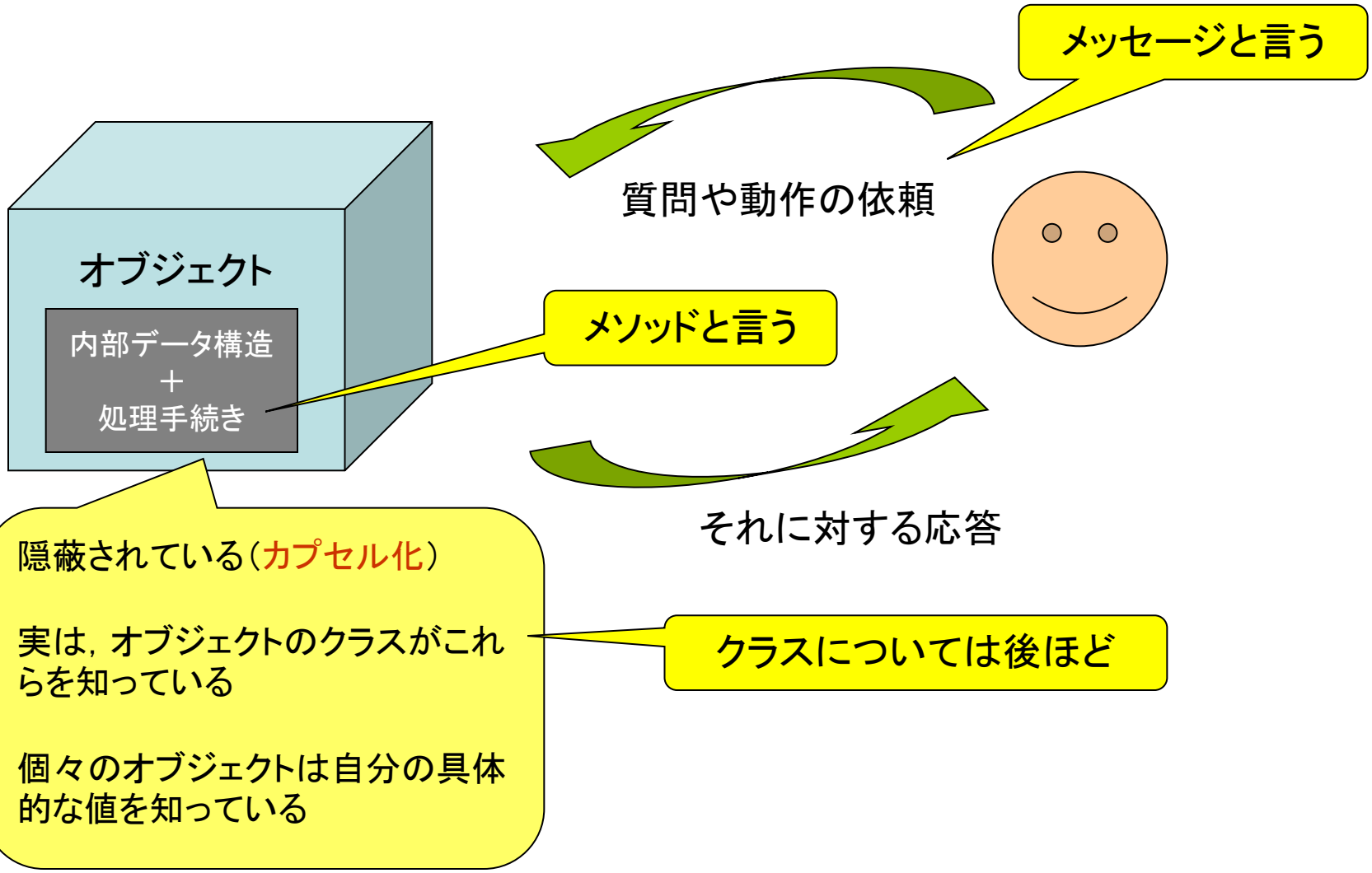
  is-komaba report 2

  としてください。
- 提出内容
  - プログラム
  - 簡単な説明
- 〆切: 1月10日（過ぎても受け付けます）

レポート補足

- シナリオ
  - サーバを起動
  - クライアントA: start(1000)
  - クライアントB: bid(1050) → true
  - クライアントC: bid(1010) → false
  - クライアントC: current() → 1050
  - クライアントC: bid(1100) → true
  - クライアントD: close() → 未定（エラー）
  - クライアントA: close() → Cのアドレス

オブジェクト（再掲）


メッセージと言う
質問や動作の依頼
オブジェクト
内部データ構造
＋
処理手続き
メソッドと言う
それに対する応答
隠蔽されている（カプセル化）
実は，オブジェクトのクラスがこれらを知っている
個々のオブジェクトは自分の具体的な値を知っている
クラスについては後ほど

クラス

- クラスとは、オブジェクトの種類
  - たとえば Hash はクラス
  - クラスの名前は大文字で始まる
- オブジェクト.class() とすると、オブジェクトのクラスが得られる。
- クラス.new() とすると、
  - そのクラスに属するオブジェクト（インスタンス）が新しく作られて返される。
  - たとえば Hash.new()

クラス（続き）

- クラスは属性とメソッドを持つ。
  - 属性 --- オブジェクトの内部データ構造
  - メソッド --- 処理手続き
- 属性
  - そのクラスのインスタンスごとに異なる値が設定される。
  - インスタンス変数、フィールドともいう。
  - Rubyでは @ で始まる。
  - たとえば @x

クラス（続き）

- メソッド
  - そのクラスのインスタンスに共有される。
  - メソッドの定義は def で始まる。
  - メソッドの中でインスタンス変数を参照できる。
  - 特にインスタンスメソッドという。
- 属性とメソッドを定義することにより、新しいクラスを定義することができる。
  - Rubyでは属性を明示的に指定する必要はない。
  - 単に @ で始まる変数を使えばよい。

クラス定義

```
class クラス名
  def メソッド(仮引数)
    ...
  end
  ...
end
```

初期化メソッド

- initialize という名前が付いている。
- クラス.new(...) とすると、そのクラスのインスタンスが作られ、initialize が引数 (...) とともに呼び出される。

self

- オブジェクト自分自身を表す。
- self.メソッド(…) は単に メソッド(…) でよい。

so.rb

```
class Poll

  def initialize()
    @log = Hash.new()
    @result = make1d(10)
  end

  def update(n, o)
    @result[n] = @result[n] + 1
    if o != 0
      @result[o] = @result[o] - 1
    end
    self.show_result()
  end

  def vote(addr, i)
    if @log.fetch(addr, 0) != 0
      x = true
      self.update(i, @log.fetch(addr))
    else
      x = false
      self.update(i, 0)
    end
    @log.store(addr, i)
    x
  end
```

```
  def show_result()
    puts("¥033[H¥033[J")
    puts('==========================================')
    for i in 1..9
      if @result[i]>0
        puts(i.to_s() + ' ' + ('*' * @result[i]))
      end
    end
    puts('==========================================')
  end


end
```

Poll

- 集計表
- 作成

  Poll.new()
- メソッド

vote(addr, i)	addr が i を投票する
update(n, o)	新投票 n で旧投票 o を置き換える
show_result()	集計表を表示する

やってみよう

- sss.rb をロードして

```
p = Poll.new()
p.vote("hagiya", 9)
p.vote("nakamura", 7)
p.vote("hagiya", 6)
p.vote("suzuki", 7)
p.show_result()
```

進捗の確認

1. 動いた。
2. Poll の使い方はわかるが、動かない。
3. Poll の使い方がわからない。
4. プログラムがダウンロードできない。

so.rb

- ss.rb をオブジェクト指向に書き直したもの
- Collector というクラスが定義されている。
- 次のようにして使う。

```
c = Collector.new(12345)
c.collect()
```

```
class Collector


  def initialize(port)
    @poll = Poll.new()
    @gs = TCPServer.new(port)
  end


  def collect()
    while true
      s = @gs.accept()
      self.serve(s)
    end
  end
```

```
  def serve(s)
    addr = s.peeraddr[3]
    sel = s.gets()
    i = sel.to_i()
    if i < 1 || i > 9
      s.puts('error: illegal selection')
    else
      if @poll.vote(addr, i)
        s.puts('updated')
      else
        s.puts('ok')
      end
    end
    s.close()
  end
```

so.rb

```
end
```

Collector

- 集計者
- 作成

  Collector.new(ポート番号)
- メソッド

  collect()　　集計手続きを起動する

  serve(ソケット)　　クライアントの相手をする

やってみよう

- ss.rb の代わりに so.rb を用いてサーバを起動する。
- cc.rb から同様に投票できるか確認する。

進捗の確認

1. 動いた。
2. Collector の使い方はわかるが、動かない。
3. Collector の使い方がわからない。
4. プログラムがダウンロードできない。

オブジェクト，クラス，インスタンス変数，メソッド

- オブジェクト
  - コンピュータのなかで「もの」に対応する"もの"
- クラス
  - 同じようなオブジェクトをまとめたもの
  - プログラミングでは先にクラスを定義しておき，必要に応じてクラスに属するオブジェクトを生成する
  - 生成したオブジェクトを，クラスのインスタンスという
- インスタンス変数
  - オブジェクトなかでデータを表す要素
  - レコードではフィールド，「もの」の視点では属性に対応
- インスタンスメソッド
  - オブジェクトの操作を行うためのメソッド

メッセージのやり取りはメソッドの呼び出しで代用　メッセージの内容は引数に対応

カプセル化，継承，多相性

- カプセル化
  - 「もの」を表すデータとメソッドをまとめ，決められたインスタンスメソッドを通してのみインスタンス変数にアクセスをさせ，インスタンス変数に直接触らせないこと
- 継承
  - 他のクラスのインスタンス変数やメソッドを受け継いで新しいクラスを作ること
  - 新しいクラスをサブクラス，元のクラスをスーパークラスと呼ぶ
- 多相性
  - 同じ名前のインスタンスメソッドを呼び出しても，適用されたオブジェクトによって異なるメソッドを呼び出すこと
  - 実行時にメソッドを探す動的結合で実現する

参考

- 去年のレポート課題は、Collector クラスを継承するサブクラスにより実現することができる。

```
class NewCollector < Collector
  def serve(s)
    ...
```

- serve メソッドを書き換える。
- 新しいサービスのためのメソッドを追加。

参考

- サーバが多数のクライアントの相手をする場合、一つのクライアントの処理に手間がかかっていると、別のクライアントの処理が遅れてしまう。
- 通常は、クライアントごとに、スレッド（計算主体）を生成して、その相手をさせる。
- 複数のスレッドが同時に走る場合、その間の競合を避ける仕組みが必要になる。
- もとの s.rb はそのようなプログラムであった。