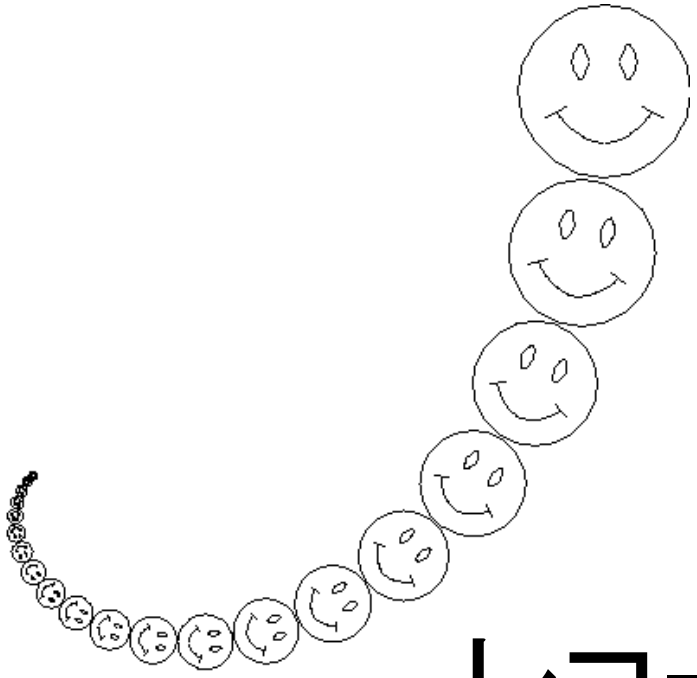
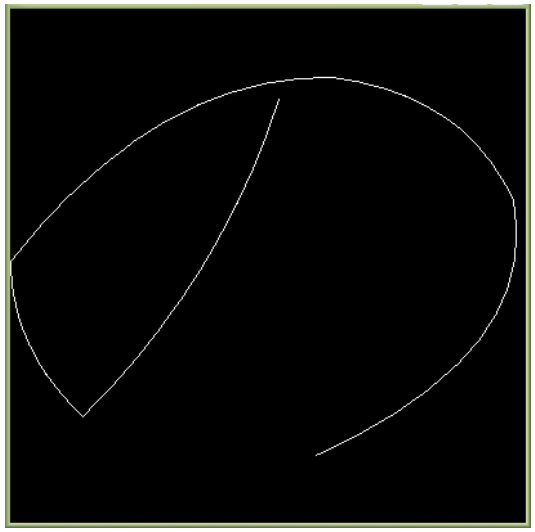
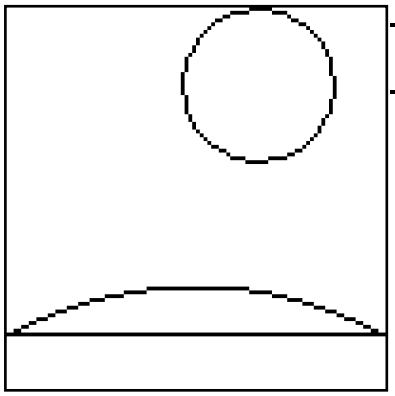


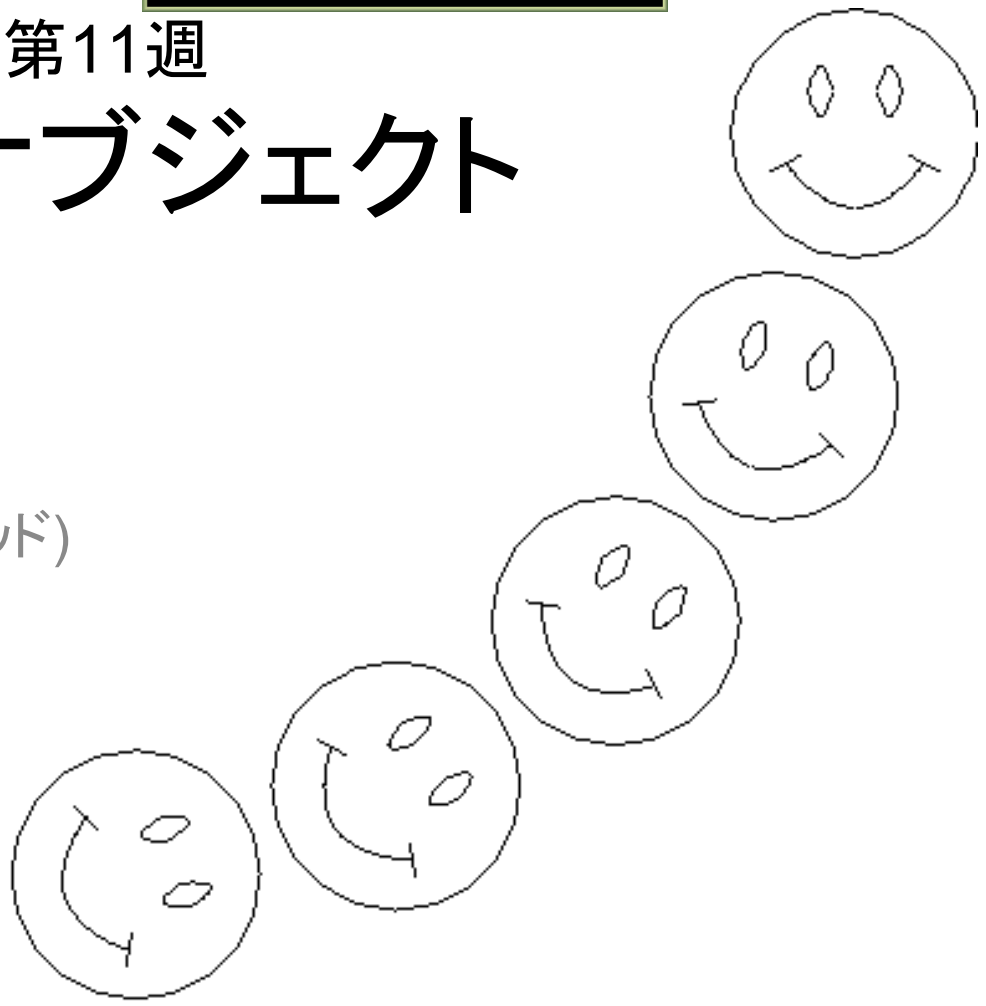


情報科学第11週

レコードとオブジェクト

- レコード
- オブジェクト
  - 点や図形 (クラス、メソッド)
  - 継承
  - 多相性



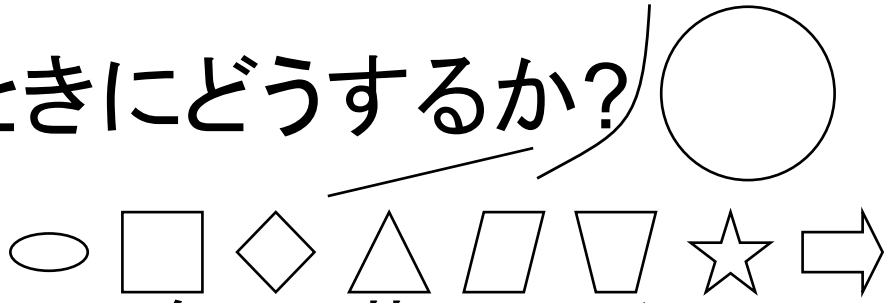


復習: オブジェクト指向の特長

- 各オブジェクトが「何をすればよいか」を知っている (メソッド)
  - 円や曲線は「自分を画面に描く」方法を知っている
- オブジェクト使う側はどんなオブジェクトかを気にせずに指示を出せる (多相性)
  - 図形に対して「画面に描け」とメッセージを送るだけ

継承: データの種類が増えたとき

- データの種類が増えたときにどうするか?
  - 直線・2次Bezier曲線・円
  - 正方形・長方形・三角形・正五角形・菱形・平行四辺形・台形・星形・楕円・矢印・折れ線・・・
- それぞれ別のクラスとして定義する?
  - 似たような定義を何回もしなければいけない



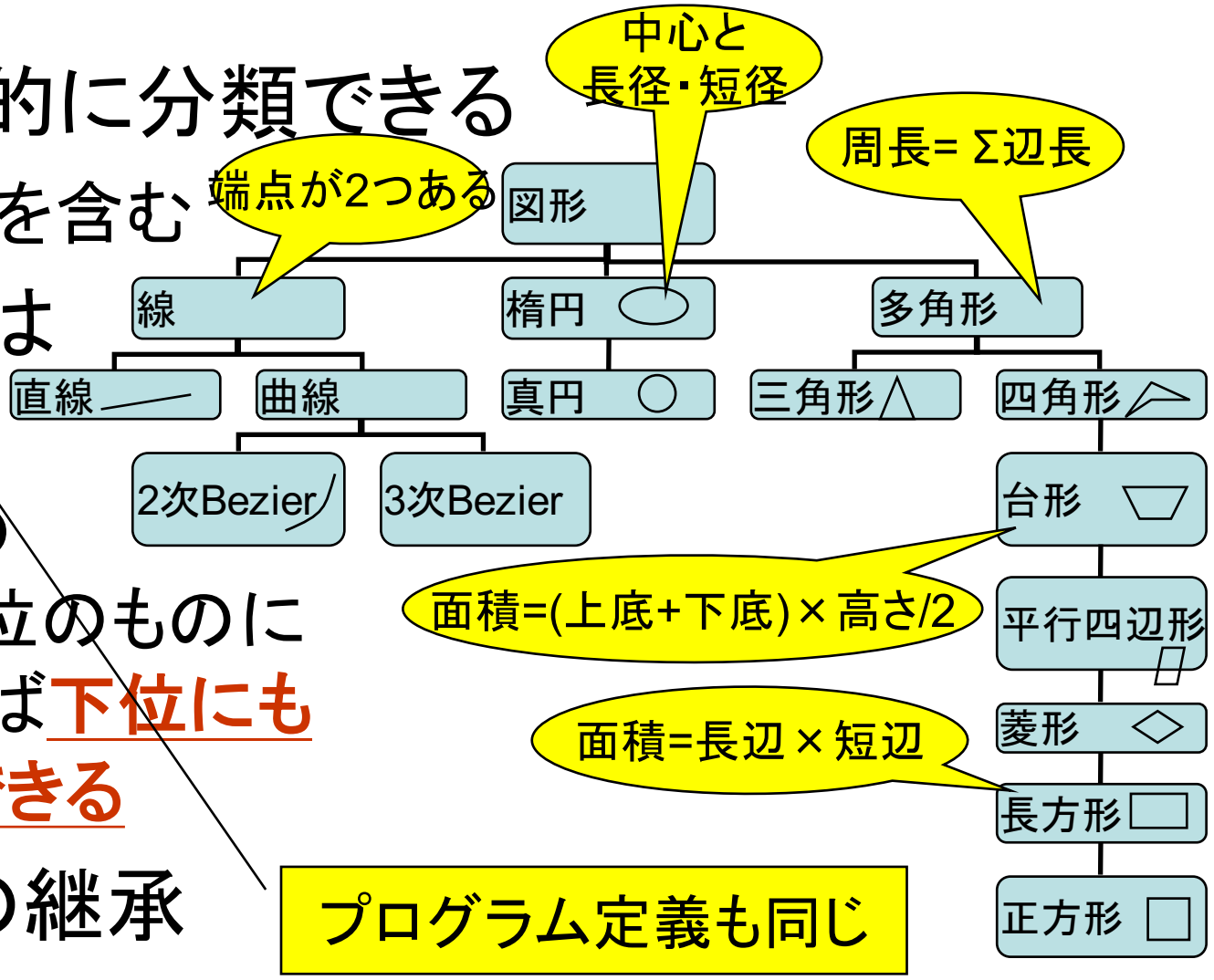
→分類して共通部分をとり出す

継承: 図形の分類と性質

- 図形は階層的に分類できる
  - 上位が下位を含む
- 上位の性質は下位も持つ

  例: 面積・周の計算は、上位のものに定義されれば**下位にも使うことができる**

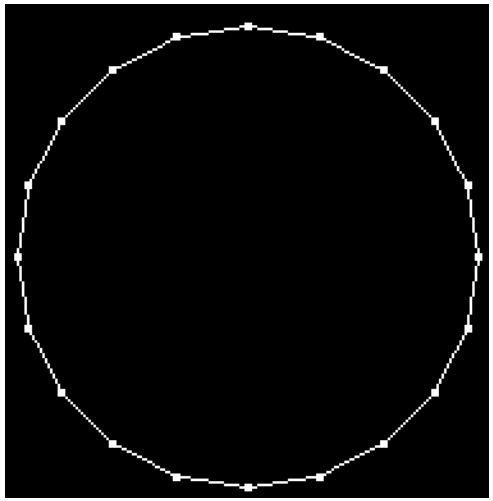
→プログラムの継承



継承の例: 端点のある直線

- 直線:

- 点付き直線:

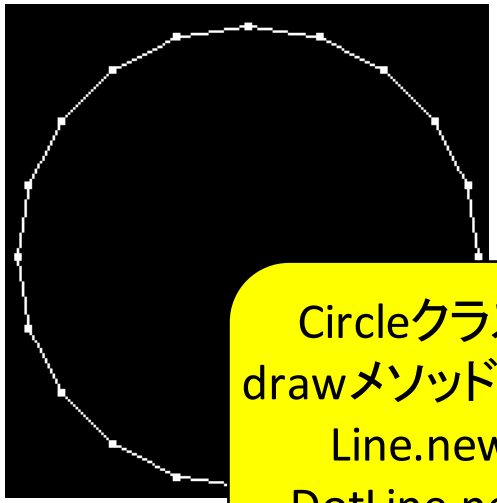


- インスタンス変数
  - 両端点の位置
- メソッド
  - 初期化: 与えられた2点を両端点として記憶
  - 描画: 両端点を結ぶ直線上に点を打ち、さらに両端点のまわりに点を打つ
  - 回転: 両端点を回転させる
  - 定数倍: 両端点を定数倍

直線と違うのはほんの一部

継承の例

- 直線:
- 点付き直線:



Circleクラスの
drawメソッドの中で
Line.newを
DotLine.newに
置きかえる

```
def dot(p,a)
  for x in -1..1
    for y in -1..1
      p.add(Point.new(x,y)).draw(a)
    end
  end
end
class DotLine < Line
  def draw(a)
    super(a)
    dot(p0,a)
    dot(p1,a)
  end
end
```

pの八方に
点を打つ関数

DotLineクラスは
Lineクラスを継承

DotLineクラスは
drawメソッドを上書き

親クラスの
draw(a)を実行

p0とp1に点を打つ

継承

```
class Bezier < Line
  attr_accessor("p0", "c", "p1")

  def initialize(q,r,s)
    super(q,s)
    self.c  = r
  end
  …
  def turn(theta)
    super(theta)
    self.c = self.c.rotate(theta)
  end
end
```

Lineクラスを継承

Lineクラスの初期化メソッドを呼び出す

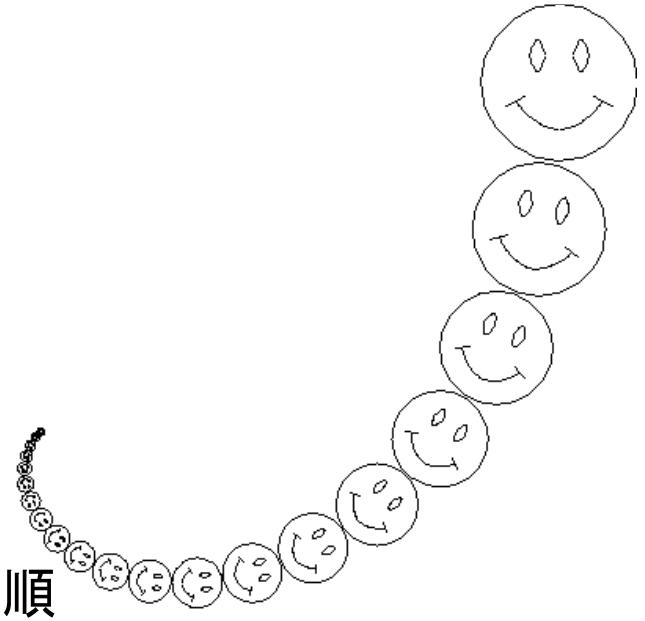
Lineクラスのturnメソッドを呼び出す

oo-bezier.rb

配布プログラムの先頭に load("./必要なファイル")を追加しておくと便利

練習

1. Pointクラス(oo-point.rb)にdraw(a)とinterpolate(q,t)を追加 (練習8.5 b,c)
2. drawall.rbのCircle.newの行を消してdrawmoon()
3. Circleクラスを定義(練習8.6) (必要なメソッドはinitializeとdraw)
4. drawall.rbのCircle.newを復活させてdrawmoon()
5. Pointクラスにsub,rotateを追加 (練習8.5a,d)
6. oo-face.rbを読込、a=make2d(400,400) drawall(face(),a) ; show(a)



練習8.11の手順

- Line, Circle, Bezierに以下のメソッドを定義
  - zoom(s): 自身を原点からs倍の位置に移動させる
  - turn(t): 自身を角度tだけ回転させる
  - move(p): 自身をpの座標ぶんだけ平行移動 (練習8.7a)
- oo-line.rb, oo-circle.rb, oo-bezier.rb, oo-face.rb, oo-whirl.rbを読込、whirl()を実行

Pointクラスのdrawとinterpolate

```
class Point
  …
  def interpolate(q,t)
    
  end
  def draw(a)
    
  end
end
```

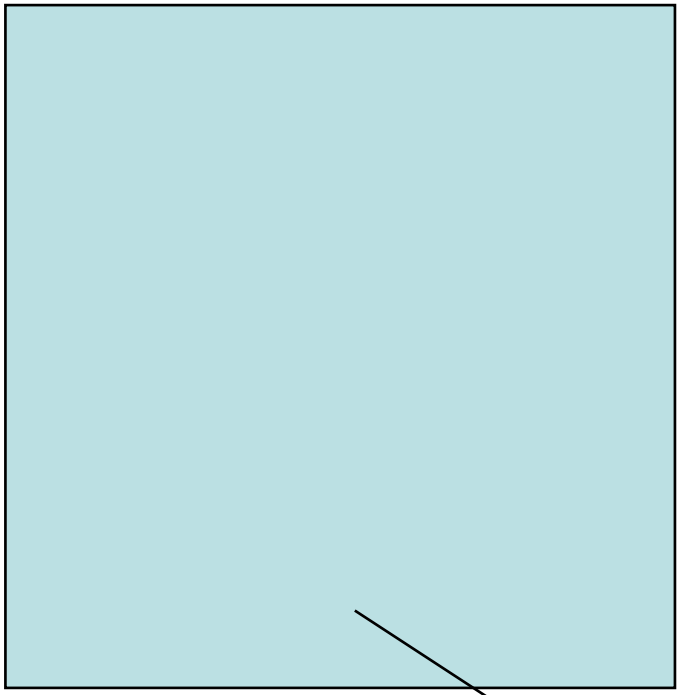
```
def point_interpolate(p,q,t)
  point_add(point_scale(p,1-t),
            point_scale(q,t))
end

def point_draw(p,a)
  if 0 <= p.y+0.5 && p.y+0.5 < a.   length() &&
     0 <= p.x+0.5 && p.x+0.5 < a[0].length()
    a[p.y+0.5][p.x+0.5]=1
  end
end
```

point.rb

- pのかわりにself(自分)を使う
- point_add(x,y) は x.add(y) になる。point_scaleも同様

Circleクラスの定義

```
class Circle
```



```
end
```

参考:

```
class Point
  attr_accessor("x", "y")
  def initialize(u,v)
    self.x = u
    self.y = v
  end
end
```

- (r cosθ+c.x, r sinθ+c.y)が円上の点
- $\theta_i=2\pi i/n$としてθ_iの点とθ_i+1の点を線で結ぶ

オブジェクト指向のまとめ

- カプセル化
  - 複雑な値を1つにまとめられる (レコードも同様)

  例: X座標とY座標 点オブジェクト
- 多相性 (polymorphism)
  - 異なる種類の値を同じように扱える
  - オブジェクト(のクラス)ごとにメソッド定義が違ってよい

  例: 色々な種類の図形が入った配列を表示
  ある点まわりの回転 = 平行移動+回転
- 継承
  - 異種の値に共通する性質をまとめられる
  - ちょっと異なる種類の値を簡単に定義できる
  - クラスの親子関係を作り、子は親の定義を引き継ぐ

  例: 曲線は直線に制御点を追加したもの
  両端に点のある直線は、表示のときに直線の表示 + 端点の表示

クラスの定義: クラス定義は次のような形をしている。

```
class 名前1 < 名前2
  attr_accessor("変数名1","変数名2",...)
  メソッド定義1
  メソッド定義2
  ⋮
end
```

名前$_1$は新しく定義されるクラスの名前である。名前$_2$はすでであるクラスの名前で**親クラス**と呼ばれる。「< 名前$_2$」の部分は省略することができる。

インスタンス変数: １つ１つのオブジェクトの中にある変数をインスタンス変数 という。クラス定義に

attr_accessor("変数名$_1$","変数名$_2$",...)

と書くとそのクラスのオブジェクトは 変数名$_i$ という名前のインスタンス変数 を持つようになる。attr_accessor に書く変数名は " " で囲まなければいけないことに注意せよ。

初期化メソッド: あるクラスのオブジェクトが作られるときに呼び出されるメソッドを初期化メソッドという。Ruby では initialize という名前のメソッドを定義すると、それが初期化メソッドとなる。

自分自身を表わす変数 self: メソッド定義の中では、現在そのメソッドを実行しているオブジェクト自身を self という名前の変数で表わす。

親クラスのメソッドを呼び出す `super`: メソッド中で `super(`式$_1$`,`式$_2$`,...)` という式が計算されると、現在実行中のメソッドと同じ名前の親クラスに定義されたメソッドが実行される。

オブジェクトの作成: あるクラスのオブジェクトを作る式は

クラス名`.new(`式$_1$`,`式$_2$`,...)`

という形をしている。この式が計算されると、クラス名のオブジェクト が 1 つでき、そのオブジェクトの `initialize` メソッドが自動的に実行される。

インスタンス変数の読み書き: 式 . 変数名 という式は、 式 が表わすオブジェクトの 変数名 というインスタンス変数を参照する。

式$_1$. 変数名 = 式$_2$ という命令は、 式$_1$ が表わすオブジェクトの 変数名 というインスタンス変数に 式$_2$ の値を代入する。

メソッドの実行: 式$_0$. メソッド名 (式$_1$, 式$_2$, . . .) という式は、 式$_0$ が表わすオブジェクトのメソッドを呼び出す。 式$_0$ が表わすオブジェクトのクラスに メソッド名 を持つメソッドが定義されている場合は、そのメソッドが実行される。定義されていない場合は、親クラスを順に辿ってゆき最初に見つけた メソッド名 を持つメソッド を実行する。