計算機ソフトウェア＃5

久野　靖*

1992.5.27

本日の内容

今日は最初に、デモを改造された方の「成果」を拝見したいですね。

本日の論文紹介は原田さんの番ですのでおねがいします。なお OOPSLA の Proceeding は「計算機ソフトウェア」と書かれたメールボックスに入っていますよ、豊島さん。あと、もしかしたら尾崎さんが 1 本紹介してくださるかも知れません...

後半のデモですが、今日は図形を増やすのと、あと＋αがちょっとあります。そろそろ種切れが近そうなので、来週あたり IB の資料を配布しようかと思っています。

本日もミニアンケートを (あとで) 出しますので、2・3 日中に答を投稿してください。

1　図形を増やす

さて、図形を増やす宿題ですが、ここでは簡単だから直線を増やしてみました。まず Line.h ですが。

```
#import <appkit/appkit.h>

@interface Line : Object {
    float x1, y1, x2, y2;
}
+ newStartX: (float)px1 Y: (float) py1 EndX: (float)px2 Y: (float)py2;
- drawIt;
@end
```

実現も簡単です。

```
#import "Line.h"
#import <dpsclient/psops.h>

@implementation Line
+ newStartX: (float)px1 Y: (float) py1 EndX: (float)px2 Y: (float)py2; {
      self = [super new];
      x1 = px1; y1 = py1; x2 = px2; y2 = py2;
      return self;
}
- drawIt {
      PSnewpath();
      PSmoveto(x1, y1);
      PSlineto(x2, y2);
      PSsetgray(0.0);
```

*筑波大学大学院経営システム科学専攻

```
        PSstroke();
    }
    @end
```

よろしいですね? main.m もほとんど変わりません。

```
    #import <appkit/appkit.h>
    #import "TestView.h"
    #import "Circle.h"
    #import "Line.h"

    main() {
        id myWin, myPanel, myMenu, myView;
        id cir1, cir2, lin1, lin2;
        NXRect aRect;
        [Application new];
        NXSetRect(&aRect, 100.0, 300.0, 300.0, 300.0);
        myWin = [Window newContent: &aRect
                    style: NX_TITLEDSTYLE
                    backing: NX_BUFFERED
                    buttonMask: NX_MINIATURIZEBUTTONMASK
                    defer: NO];
        [myWin setTitle: "CompSoftDemo 4.0"];
        NXSetRect(&aRect, 0.0, 0.0, 300.0, 300.0);
        myView = [TestView newFrame: &aRect];
        [[myWin contentView] addSubview: myView];
        cir1 = [[Circle newRadius: 80.0] setPosX: 100.0 Y: 100.0];
        [myView addOne: cir1];
        cir2 = [[Circle newRadius: 40.0] setPosX: 200.0 Y: 50.0];
        [myView addOne: cir2];
        lin1 = [Line newStartX: 40.0 Y: 10.0 EndX: 100.0 Y: 100.0];
        [myView addOne: lin1];
        lin2 = [Line newStartX: 80.0 Y: 100.0 EndX: 10.0 Y: 20.0];
        [myView addOne: lin2];
        NXSetRect(&aRect, 100.0, 700.0, 300.0, 40.0);
        myPanel = [Panel newContent: &aRect
                       style: NX_TITLEDSTYLE
                       backing: NX_BUFFERED
                       buttonMask: NX_CLOSEBUTTONMASK
                       defer: YES];
        [myPanel setTitle: "About CompSoftDemo"];
        [myPanel removeFromEventMask: (NX_KEYDOWNMASK|NX_KEYUPMASK)];
        myMenu = [Menu newTitle: "CS Demo"];
        [[myMenu addItem: "Info..."
              action: @selector(orderFront:)
              keyEquivalent: 'i'] setTarget: myPanel];
        [myMenu addItem: "Hide" action: @selector(hide:) keyEquivalent: 'h'];
        [myMenu addItem: "Quit" action: @selector(terminate:) keyEquivalent: 'q'];
        [myMenu sizeToFit];
        [NXApp setMainMenu: myMenu];
        [myWin display];
        [myWin orderFront: nil];
        [myWin makeKeyWindow];
        [NXApp run];
        [NXApp free];
    }
```

さて、これで実行させるのですが、だいぶファイルが増えてきたので、全部一度に翻訳していては遅くなってきました。そこで、次の方法をやってみてください。

いつものようにコピーしてきて、そのディレクトリに行く。

```
% cc -c main.m
% cc -c TestView.m
% cc -c Circle.m
% cc -c Line.m
% cc main.o TestView.o Circle.o Line.o -lNeXT_s -lsys_s
% a.out
```

なお、-c というオプションは「ソースファイルから対応するオブジェクトファイルを作ってくれ」という意味になります。つまりここでは、各ファイルをまずそれぞれオブジェクトにして、最後にそれらオブジェクトをまとめて実行形式 (a.out) を作っているわけですね。さて、こうするとどういういいことがあるでしょう?

それを体験するために、円と直線の位置/大きさを適宜変更して a.out を作りなおしてみてください。

2 make と構成管理

上のようにいちいち手で打ち込み、しかもどの.m は再翻訳が必要かを把握しておくのは面倒そのものである。そこで、それを自動化してくれるツールが make である。make 指令は現在位置にある Makefile というファイルを参照して動作する。実は上のサンプルには次の Makefile が付随している。

```
OBJS = main.o TestView.o Circle.o Line.o

SRCS = main.m TestView.m Circle.m Line.m

CFLAGS =

a.out:  ${OBJS}
        cc ${OBJS} -lNeXT_s -lsys_s

main.o: main.m
        cc ${CFLAGS} -c main.m

TestView.o:     TestView.m TestView.h
        cc ${CFLAGS} -c TestView.m

Circle.o:       Circle.m Circle.h
        cc ${CFLAGS} -c Circle.m

Line.o: Line.m Line.h
        cc ${CFLAGS} -c Line.m
```

これは例えば次のように読む。『a.out はなんとか.o ファイルすべてに依存していて、作るためにはこれらが存在しないといけない。これらがあれば、a.out は「cc なんとか.o のならび -lNeXT_s -lsys_s」を実行することでできる』『main.o は main.m に依存していて、作るためにはこれがないといけない。あれば、main.o は「cc -c main.m」を実行することでできる』以下同様。

make を実行するには単に

```
make
```

というだけでよい。

練習: また円や直線の位置を変更して、make を実行してみよ。

3 復号図形

さて、円と直線が使えるので、今度はこれで人の形を描いてみよう。たとえば...(白板に描く。)これを描くのに、あなたならどういう風に例題を直しますか?

もちろん、「人」もオブジェクトにする。じゃあその中はどうやって描く? いきなり PS の円とか直線をつらねてももちろんいい。でもせっかく円や直線のオブジェクトを作ったのだから、これを「張り合わせて」人を作ろう。まず Person.h。

```
#import <appkit/appkit.h>

@interface Person : Object {
    id part[6];
    float x, y;
}
+ newAtX: (float)px Y: (float) py Height: (float) ph;
- drawIt;
@end
```

つぎに Person.m。

```
#import "Person.h"
#import "Circle.h"
#import "Line.h"

@implementation Person
+ newAtX: (float)px Y: (float)py Height: (float)ph {
      float d = ph / 8.0;
      self = [super new];
      part[0] = [[Circle newRadius: d*2.0] setPosX: 0.0 Y: d*4.0];
      part[1] = [Line newStartX: 0.0 Y: d*2.0 EndX: 0.0 Y: -d*2.0];
      part[2] = [Line newStartX: 0.0 Y: 0.0 EndX: d*2.0 Y: d];
      part[3] = [Line newStartX: 0.0 Y: 0.0 EndX: -d*2.0 Y: d];
      part[4] = [Line newStartX: 0.0 Y: -d*2.0 EndX: d*2.0 Y: -d*4.0];
      part[5] = [Line newStartX: 0.0 Y: -d*2.0 EndX: -d*2.0 Y: -d*4.0];
      x = px; y = py;
      return self;
}
- drawIt {
      int i;
      PSgsave();
      PStranslate(x, y);
      for(i = 0; i < 6; ++i) [part[i] drawIt];
      PSgrestore();
}
@end
```

で、これをいくつでも張り付ければいいわけだ。

```
#import <appkit/appkit.h>
#import "TestView.h"
#import "Person.h"

main() {
    id myWin, myPanel, myMenu, myView;
    id per1, per2;
    NXRect aRect;
    [Application new];
    NXSetRect(&aRect, 100.0, 300.0, 300.0, 300.0);
    myWin = [Window newContent: &aRect
                style: NX_TITLEDSTYLE
```

```
                backing: NX_BUFFERED
                buttonMask: NX_MINIATURIZEBUTTONMASK
                defer: NO];
    [myWin setTitle: "CompSoftDemo 4.1"];
    NXSetRect(&aRect, 0.0, 0.0, 300.0, 300.0);
    myView = [TestView newFrame: &aRect];
    [[myWin contentView] addSubview: myView];
    per1 = [Person newAtX: 100.0 Y: 100.0 Height: 100.0];
    [myView addOne: per1];
    per2 = [Person newAtX: 200.0 Y: 100.0 Height: 100.0];
    [myView addOne: per2];
    NXSetRect(&aRect, 100.0, 700.0, 300.0, 40.0);
    myPanel = [Panel newContent: &aRect
                    style: NX_TITLEDSTYLE
                    backing: NX_BUFFERED
                    buttonMask: NX_CLOSEBUTTONMASK
                    defer: YES];
    [myPanel setTitle: "About CompSoftDemo"];
    [myPanel removeFromEventMask: (NX_KEYDOWNMASK|NX_KEYUPMASK)];
    myMenu = [Menu newTitle: "CS Demo"];
    [[myMenu addItem: "Info..."
          action: @selector(orderFront:)
          keyEquivalent: 'i'] setTarget: myPanel];
    [myMenu addItem: "Hide" action: @selector(hide:) keyEquivalent: 'h'];
    [myMenu addItem: "Quit" action: @selector(terminate:) keyEquivalent: 'q'];
    [myMenu sizeToFit];
    [NXApp setMainMenu: myMenu];
    [myWin display];
    [myWin orderFront: nil];
    [myWin makeKeyWindow];
    [NXApp run];
    [NXApp free];
}
```

4 じゃあ猫も描くには?

で、次の問題。同様に、猫も描きたいときは? もちろん、猫もオブジェクトにする。どうやって作る? Person.h と同様? それだとちょっと... 実は Person も Cat も最初に「張り付ける」ところが違うだけで、あとは同じになるはずでしょう? ということは... そう、「張り付けて作るオブジェクト」というクラスをまず用意し、Person と Cat はそのサブクラスにする、というのが正しい。まず Composite.h。

```
#import <appkit/appkit.h>

@interface Composite : Object {
    id part[20];
    int count;
    float x, y;
}
+ newAtX: (float)px Y: (float) py;
- addPart: p;
- drawIt;
@end
```

つづいて Composite.m。

```
#import "Composite.h"
```

```
@implementation Composite
+ newAtX: (float)px Y: (float)py {
      self = [super new];
      x = px; y = py; count = 0;
      return self;
}
- addPart: p {
      part[count++] = p;
}
- drawIt {
      int i;
      PSgsave();
      PStranslate(x, y);
      for(i = 0; i < count; ++i) [part[i] drawIt];
      PSgrestore();
}
@end
```

さて、これで準備ができた。Cat.h/m は次の通り。

```
#import <appkit/appkit.h>
#import "Composite.h"

@interface Cat : Composite {
}
+ newAtX: (float)px Y: (float) py Height: (float) ph;
@end


#import "Cat.h"
#import "Circle.h"
#import "Line.h"

@implementation Cat
+ newAtX: (float)px Y: (float)py Height: (float)ph {
      float d = ph / 4.0;
      self = [super newAtX: px Y: py];
      [self addPart: [[Circle newRadius: d*2.0] setPosX: -d*3.0 Y: 0.0]];
      [self addPart: [Line newStartX: -d*4.0 Y: 0.0 EndX: -d*3.0 Y: 0.0]];
      [self addPart: [Line newStartX: -d*4.0 Y: 0.0 EndX: -d*3.0 Y: -d/2.0]];
      [self addPart: [Line newStartX: -d*4.0 Y: 0.0 EndX: -d*3.0 Y: -d]];
      [self addPart: [Line newStartX: -d Y: 0.0 EndX: d*3.0 Y: 0.0]];
      [self addPart: [Line newStartX: 0.0 Y: 0.0 EndX: d Y: -d*3.0]];
      [self addPart: [Line newStartX: 0.0 Y: 0.0 EndX: -d Y: -d*3.0]];
      [self addPart: [Line newStartX: d*3.0 Y: 0.0 EndX: d*2.0 Y: -d*3.0]];
      [self addPart: [Line newStartX: d*3.0 Y: 0.0 EndX: d*4.0 Y: -d*3.0]];
      [self addPart: [Line newStartX: d*3.0 Y: 0.0 EndX: d*4.0 Y: d]];
      return self;
}
@end
```

なお、Person.h/m も同様に直してあるので注意。

```
#import <appkit/appkit.h>
#import "Composite.h"

@interface Person : Composite {
}
+ newAtX: (float)px Y: (float) py Height: (float) ph;
@end
```

```
#import "Person.h"
#import "Circle.h"
#import "Line.h"

@implementation Person
+ newAtX: (float)px Y: (float)py Height: (float)ph {
      float d = ph / 8.0;
      self = [super newAtX: px Y: py];
      [self addPart: [[Circle newRadius: d*2.0] setPosX: 0.0 Y: d*4.0]];
      [self addPart: [Line newStartX: 0.0 Y: d*2.0 EndX: 0.0 Y: -d*2.0]];
      [self addPart: [Line newStartX: 0.0 Y: 0.0 EndX: d*2.0 Y: d]];
      [self addPart: [Line newStartX: 0.0 Y: 0.0 EndX: -d*2.0 Y: d]];
      [self addPart: [Line newStartX: 0.0 Y: -d*2.0 EndX: d*2.0 Y: -d*4.0]];
      [self addPart: [Line newStartX: 0.0 Y: -d*2.0 EndX: -d*2.0 Y: -d*4.0]];
      return self;
}
@end
```

以上なわけです。ところで、TestView.m の中にも Composite.m と同じようなコードがありましたね? だから、TestView も Composite のサブクラスに... と思うと、こいつは既に View のサブクラスだからそうも行かない。多重継承だとそれができるけど、そうするのがいいかどうか... まあ難しいところだとは思います。

もう 1 つ。今回は Person や Cat に共通の親を与えたわけですが、Line や Circle にも共通の親があるといいと思いません? (なぜか?) そうやっていくと、結構それらしい階層構造ができると思いませんか。

さて、練習問題です。

練習 1 円や直線の共通の親を作り、ここに「大きくなる」「回転」などの機能を入れてみよ。

練習 2 もちろん、それをボタンやスライダーなどで動かしてみられるようにして欲しい。

とりあえずは TestView に 1 つしか張り付けないとか、いくつも張り付けても 1 つだけ動かすことにするとか、全部いっせいに動くとか、適当にやりやすいように工夫してやってみてください。そろそろ苦しいですか? 聞きにきてくださってもいいですよー。