ヒューマンインタフェース'99 ＃4

久野　靖*

1999.5.7

はじめに

前回の tcl/tk は面倒なことを tcl/tk の中でやってくれてしまうので、「生の」ウィンドウシステムの機能がどのように使われているのかが分からない。また、tcl/tk がやってくれないこと (たとえば「アイコン投げ」のようなヘンなこと) はできない。

今回は相当ハードだが、ウィンドウシステムを直接使ってプログラミングするとどうなるか、ということをまず見物し、続いてこのようなプログラミングにおいて「オブジェクト指向」がどのように役立つかを C++ 言語を題材として体験する。

1　ウィンドウプログラミング

1.1　復習: イベントドリブン

まず、計算機科学基礎で取り上げた X-Window の例題プログラムを再掲する (ちょっとだけ改良してあるが)。

```
/* t61.c --- very simple X client */

#include <X11/Xlib.h>

main() {
  Display *disp = XOpenDisplay(NULL);
  Screen *scr = DefaultScreenOfDisplay(disp);
  Window root = DefaultRootWindow(disp);
  unsigned long black = BlackPixelOfScreen(scr);
  unsigned long white = WhitePixelOfScreen(scr);
  Window mw = XCreateSimpleWindow(disp,root,100,100,400,200,2,black,white);
  XSelectInput(disp, mw, ButtonPressMask|KeyPressMask|ExposureMask);
  XMapWindow(disp, mw);
  while(1) {
    XEvent ev;
    XNextEvent(disp, &ev);
    if(ev.type == KeyPress)
      exit(0);
```

*筑波大学大学院経営システム科学専攻

```
    else if(ev.type == Expose)
      draw(disp, mw, 20, 20);
    else if(ev.type == ButtonPress)
      draw(disp, mw, ev.xbutton.x, ev.xbutton.y);
  }
}

draw(Display *disp, Window mw, int x, int y) {
  GC dgc = DefaultGC(disp, 0);
  XFillArc(disp, mw, dgc, x-20, y-20, 40, 40, 0, 360*64);
}
```

なお、これを動かすには次のようにする。

```
% export LD_LIBRARY_PATH=/usr/local/X11R6/lib  ← 1 回やればよい
% gcc t61.c -I/usr/local/X11R6/include -lX11 -lnsl -lsocket
                                          ↑ -lnsl 以降は Soraris のみ
```

で、計算機科学基礎の練習問題では「上に別の窓を重ねても黒丸が復活できるようにせよ」ということだったが…やりましたか?

演習 1 t61.c をコピーしてきて動かせ。

演習 2 同じ練習問題を再度やってみよ。

1.2 黒丸を移動可能にする

次に、画面上の「もの」を直接操作 (direct manipulation) するための基本技術を見てみよう。基本的な要件は次の通り。

1. ボタンを押した状態でのマウスの移動と、ボタン離しとを通知してもらうようにする。
2. ボタンを押したときに、「どの黒丸の上で押したか」を調べる (picking)。
3. マウス移動イベントが来るごとに、画面をマウスに追従して更新。
4. ボタンを離したらそこでおしまい。

1 は、単に XSelectInput でマスクを書き足せばよい。2 は、順番に探せばよい。見つかったらそれを変数に覚える。一番むずかしいのは 3 の画面更新 (毎回クリアして描き直しているとちらちらして汚い)。ここではラバーバンドと呼ばれる技法を使う。基本的なアイデアは、反転演算で画面に描くこと。反転だと 2 回同じ操作をすると元に戻るので、描いたものを消して新しい位置に描くのが簡単。これを実現するには、GC(graphic context、描画操作の特性をまとめた「ペン先」) としてラバーバンド用のものをあらかじめ用意しておく。4 の「おしまい」では、本当に元の黒丸を消して移動するので、画面をクリアして再描画している。

```
/* t63.c --- make the circle movable */

#include <X11/Xlib.h>

struct obj {
  int x, y; } objs[1000];
int ouse = 0;
```

```
GC rubbergc;

main() {
  int moving = -1;
  Display *disp = XOpenDisplay(NULL);
  Screen *scr = DefaultScreenOfDisplay(disp);
  Window root = DefaultRootWindow(disp);
  unsigned long black = BlackPixelOfScreen(scr);
  unsigned long white = WhitePixelOfScreen(scr);
  Window mw = XCreateSimpleWindow(disp,root,100,100,400,200,2,black,white);
  { XGCValues gcv; gcv.function = GXinvert; gcv.line_width = 3;
    rubbergc = XCreateGC(disp, mw, GCFunction|GCLineWidth, &gcv); }
  XSelectInput(disp, mw, ButtonPressMask|KeyPressMask|ExposureMask
    |ButtonMotionMask|ButtonReleaseMask);
  XMapWindow(disp, mw);
  while(1) {
    XEvent ev;
    XNextEvent(disp, &ev);
    if(ev.type == KeyPress)
      exit(0);
    else if(ev.type == Expose) {
      drawall(disp, mw);
    }
    else if(ev.type == ButtonPress) {
      if(ev.xbutton.button == Button1) {
        objs[ouse].x = ev.xbutton.x; objs[ouse].y = ev.xbutton.y; ++ouse;
        draw(disp, mw, ev.xbutton.x, ev.xbutton.y); }
      else if(ev.xbutton.button == Button2) {
        int i = moving = lookup(ev.xbutton.x, ev.xbutton.y);
        if(i >= 0) drawrubber(disp, mw, objs[i].x, objs[i].y); }
    }
    else if(ev.type == MotionNotify) {
      if(moving >= 0) {
        int i = moving;
        drawrubber(disp, mw, objs[i].x, objs[i].y);
        objs[i].x = ev.xbutton.x; objs[i].y = ev.xbutton.y;
        drawrubber(disp, mw, objs[i].x, objs[i].y); }
    }
    else if(ev.type == ButtonRelease) {
      if(moving >= 0) {
        XClearWindow(disp, mw); drawall(disp, mw); moving = -1; }
    }
  }
}

lookup(int x, int y) {
  int i;
  for(i = 0; i < ouse; ++i)
    if(near(objs[i].x, objs[i].y, x, y, 20)) return i;
  return -1;
}
```

```
near(int x0, int y0, int x1, int y1, int d) {
  return (x0-x1)*(x0-x1) + (y0-y1)*(y0-y1) <= d*d;
}

drawall(Display *disp, Window mw) {
  int i;
  for(i = 0; i < ouse; ++i)
    draw(disp, mw, objs[i].x, objs[i].y);
}

draw(Display *disp, Window mw, int x, int y) {
  GC dgc = DefaultGC(disp, 0);
  XFillArc(disp, mw, dgc, x-20, y-20, 40, 40, 0, 360*64);
}

drawrubber(Display *disp, Window mw, int x, int y) {
  XDrawArc(disp, mw, rubbergc, x-20, y-20, 40, 40, 0, 360*64);
}
```

演習 3 このまま動かせ。

演習 4 動かし初めのところが少し変だがわかるか? これを「なめらかに」するにはどうしたらいいか? 直してみよ。

演習 5 ラバーバンドをやめて毎回クリア+描画にしてみよ。

演習 6 データ構造に「半径」を増やして、新しくできるものほど 1 ドットずつ半径が増すようにしてみよ。

演習 7 右ボタンを押して動かした場合、黒丸の半径が変化させられるようにしてみよ。もちろん、ラバーバンドを使う。

1.3 受動的なデータ構造から抽象データ型へ

まず、練習 7 の回答例から見てみる。

```
/* t64.c --- make the circle movable and resizable */

#include <X11/Xlib.h>
#include <math.h>

struct obj {
  int x, y, r; } objs[1000];
int ouse = 0;

GC rubbergc;

main() {
  int moving = -1;
  int resizing = -1;
  int radius = 20;
```

```
  Display *disp = XOpenDisplay(NULL);
  Screen *scr = DefaultScreenOfDisplay(disp);
  Window root = DefaultRootWindow(disp);
  unsigned long black = BlackPixelOfScreen(scr);
  unsigned long white = WhitePixelOfScreen(scr);
  Window mw = XCreateSimpleWindow(disp,root,100,100,400,200,2,black,white);
  { XGCValues gcv; gcv.function = GXinvert; gcv.line_width = 3;
    rubbergc = XCreateGC(disp, mw, GCFunction|GCLineWidth, &gcv); }
  XSelectInput(disp, mw, ButtonPressMask|KeyPressMask|ExposureMask
    |ButtonMotionMask|ButtonReleaseMask);
  XMapWindow(disp, mw);
  while(1) {
    XEvent ev;
    XNextEvent(disp, &ev);
    if(ev.type == KeyPress)
      exit(0);
    else if(ev.type == Expose) {
      drawall(disp, mw);
    }
    else if(ev.type == ButtonPress) {
      if(ev.xbutton.button == Button1) {
        objs[ouse].x = ev.xbutton.x; objs[ouse].y = ev.xbutton.y;
        objs[ouse].r = ++radius; ++ouse;
        draw(disp, mw, ev.xbutton.x, ev.xbutton.y, radius); }
      else if(ev.xbutton.button == Button2) {
        int i = moving = lookup(ev.xbutton.x, ev.xbutton.y);
        if(i >= 0) drawrubber(disp, mw, objs[i].x, objs[i].y, objs[i].r); }
      else if(ev.xbutton.button == Button3) {
        int i = resizing = lookup(ev.xbutton.x, ev.xbutton.y);
        if(i >= 0) {
          objs[i].r=distance(objs[i].x,objs[i].y,ev.xbutton.x,ev.xbutton.y);
          drawrubber(disp, mw, objs[i].x, objs[i].y, objs[i].r); }
      }
    }
    else if(ev.type == MotionNotify) {
      if(moving >= 0) {
        int i = moving;
        drawrubber(disp, mw, objs[i].x, objs[i].y, objs[i].r);
        objs[i].x = ev.xbutton.x; objs[i].y = ev.xbutton.y;
        drawrubber(disp, mw, objs[i].x, objs[i].y, objs[i].r); }
      if(resizing >= 0) {
        int i = resizing;
        drawrubber(disp, mw, objs[i].x, objs[i].y, objs[i].r);
        objs[i].r=distance(objs[i].x, objs[i].y, ev.xbutton.x, ev.xbutton.y);
        drawrubber(disp, mw, objs[i].x, objs[i].y, objs[i].r); }
    }
    else if(ev.type == ButtonRelease) {
      if(moving >= 0 || resizing >= 0) {
        XClearWindow(disp, mw); drawall(disp, mw); moving = resizing = -1; }
    }
  }
}
```

```
lookup(int x, int y) {
  int i;
  for(i = 0; i < ouse; ++i)
    if(near(objs[i].x, objs[i].y, x, y, objs[i].r)) return i;
  return -1;
}

near(int x0, int y0, int x1, int y1, int d) {
  return (x0-x1)*(x0-x1) + (y0-y1)*(y0-y1) <= d*d;
}

distance(int x0, int y0, int x1, int y1) {
  return (int)sqrt((x0-x1)*(x0-x1) + (y0-y1)*(y0-y1));
}

drawall(Display *disp, Window mw) {
  int i;
  for(i = 0; i < ouse; ++i)
    draw(disp, mw, objs[i].x, objs[i].y, objs[i].r);
}

draw(Display *disp, Window mw, int x, int y, int r) {
  GC dgc = DefaultGC(disp, 0);
  XFillArc(disp, mw, dgc, x-r, y-r, 2*r, 2*r, 0, 360*64);
}

drawrubber(Display *disp, Window mw, int x, int y, int r) {
  XDrawArc(disp, mw, rubbergc, x-r, y-r, 2*r, 2*r, 0, 360*64);
}
```

かなり「限界」という感じがしませんか? このように、受動的なデータ構造を使っている場合、機能を増やしていくとデータ構造に関する「約束ごと」がどんどん増え、それをアクセスする手続き側で約束を破らないようにするのがたいへんになってくる。

このような問題に対する回答として、「抽象データ型」(Abstract Data Types、ADT) がある。ADT は前にやったモジュールによく似ているが、モジュールはあくまでもデータが 1 式だったのに対し、ADT ではデータ (インスタンスとか実体とも呼ぶ) が N 個あってよい。ここでは「1 つの黒丸」を 1 つの ADT としてみる。

```
/* t65.c --- ADT circle */

#include <X11/Xlib.h>
#include <math.h>

typedef struct obj {
  Display *d; Window w; int x, y, r; } *obj_t;
obj_t objs[1000], obj_create(), lookup();
int ouse = 0;

GC rubbergc;

main() {
  obj_t moving = 0, resizing = 0;
```

```
  int radius = 20;
  Display *disp = XOpenDisplay(NULL);
  Screen *scr = DefaultScreenOfDisplay(disp);
  Window root = DefaultRootWindow(disp);
  unsigned long black = BlackPixelOfScreen(scr);
  unsigned long white = WhitePixelOfScreen(scr);
  Window mw = XCreateSimpleWindow(disp,root,100,100,400,200,2,black,white);
  { XGCValues gcv; gcv.function = GXinvert; gcv.line_width = 3;
    rubbergc = XCreateGC(disp, mw, GCFunction|GCLineWidth, &gcv); }
  XSelectInput(disp, mw, ButtonPressMask|KeyPressMask|ExposureMask
    |ButtonMotionMask|ButtonReleaseMask);
  XMapWindow(disp, mw);
  while(1) {
    XEvent ev;
    XNextEvent(disp, &ev);
    if(ev.type == KeyPress)
      exit(0);
    else if(ev.type == Expose) {
      drawall();
    }
    else if(ev.type == ButtonPress) {
      if(ev.xbutton.button == Button1) {
        objs[ouse]=obj_create(disp, mw, ev.xbutton.x, ev.xbutton.y, ++radius);
        obj_draw(objs[ouse]); ++ouse; }
      else if(ev.xbutton.button == Button2) {
        moving = lookup(ev.xbutton.x, ev.xbutton.y);
        if(moving) obj_drawrubber(moving); }
      else if(ev.xbutton.button == Button3) {
        resizing = lookup(ev.xbutton.x, ev.xbutton.y);
        if(resizing) {
          obj_resize(resizing, ev.xbutton.x, ev.xbutton.y);
          obj_drawrubber(resizing); }
      }
    }
    else if(ev.type == MotionNotify) {
      if(moving) {
        obj_drawrubber(moving);
        obj_move(moving, ev.xbutton.x, ev.xbutton.y);
        obj_drawrubber(moving); }
      if(resizing) {
        obj_drawrubber(resizing);
        obj_resize(resizing, ev.xbutton.x, ev.xbutton.y);
        obj_drawrubber(resizing); }
    }
    else if(ev.type == ButtonRelease) {
      if(moving || resizing) {
        XClearWindow(disp, mw); drawall(); moving = resizing = 0; }
    }
  }
}

obj_t lookup(int x, int y) {
```

```
  int i;
  for(i = 0; i < ouse; ++i)
    if(obj_near(objs[i], x, y)) return objs[i];
  return 0;
}

near(int x0, int y0, int x1, int y1, int d) {
  return (x0-x1)*(x0-x1) + (y0-y1)*(y0-y1) <= d*d;
}

distance(int x0, int y0, int x1, int y1) {
  return (int)sqrt((x0-x1)*(x0-x1) + (y0-y1)*(y0-y1));
}

drawall() {
  int i;
  for(i = 0; i < ouse; ++i) obj_draw(objs[i]);
}

obj_t obj_create(Display *disp, Window mw, int x, int y, int r) {
  obj_t o = (struct obj*)malloc(sizeof(struct obj));
  o->d = disp; o->w = mw; o->x = x; o->y = y; o->r = r; return o;
}

obj_draw(obj_t o) {
  int x = o->x, y = o->y, r = o->r;
  GC dgc = DefaultGC(o->d, 0);
  XFillArc(o->d, o->w, dgc, x-r, y-r, 2*r, 2*r, 0, 360*64);
}

obj_drawrubber(obj_t o) {
  int x = o->x, y = o->y, r = o->r;
  XDrawArc(o->d, o->w, rubbergc, x-r, y-r, 2*r, 2*r, 0, 360*64);
}

obj_near(obj_t o, int x, int y) {
  return near(o->x, o->y, x, y, o->r);
}

obj_move(obj_t o, int x, int y) {
  o->x = x; o->y = y;
}

obj_resize(obj_t o, int x, int y) {
  o->r = distance(o->x, o->y, x, y);
}
```

演習 8 そのまま動かせ。

演習 9 動かし初めのずれをなくすには、どういうADTインタフェースがあればいいか。設計し、実装してみよ。

演習 10 形を黒丸でなく黒四角に変更してみよ。

演習 11 黒丸と黒四角の両方が使えるようにしてみよ。たとえばシフトキーを押しながらクリックすると四角ができる、というふうに。

なお、四角の場合には次の描画ルーチンを使えばよい。

```
XFillRectangle(disp, win, GC, x, y, width, height);
XDrawRectangle(disp, win, GC, x, y, width, height);
```

また、練習 11 のためには、キーを押したらすぐ終わりでは困るので、キーイベントの処理部分を次のようにする。

```
char buf[20];
if(XLookupString(&ev, buf, 20, 0, 0) == 1 && buf[0] == 'q') exit(0);
```

つまり、XLookupString でイベントを文字列に変換して、長さ 1 の「q」の場合だけおしまいにする。あとは、マウスボタンイベントのところで Shift が押されているかどうか調べるには

```
ev.xbutton.state&ShiftMask
```

で見ることができる。

2 C++言語入門

2.1 再掲: 抽象データ型

先に取り上げように、抽象データ型 (Abstract Data Type、ADT) とは

- 外部から保護されたデータと、それをアクセスする関数群という点ではパッケージと同様。
- ただし、普通パッケージでは実体が 1 つだが、抽象データ型では任意個数の実体 (インスタンス) を生成できる。
- 利点としては、インタフェースを変更しなければ内部実現は自由に変化させられ、プログラム部品間の独立性が高まること。

しかし、C 言語で ADT をやるには「ある一定の規則に従ってプログラムを構成する」わけで、その規則が言語によって強制されていない以上様々な問題が派生する (具体的には何か?)

そこで、「よりよい言語」を使用するのがまっとうなアプローチ。ここでは C++言語を使用する。C++はオブジェクト指向言語 (って何?) と言われるが、オブジェクト指向の利点の 1 つが ADT なので、これはこれでよい (それ以外の利点については次回以降で取り上げる)。

C++では「クラス」によって ADT が作成できる。その構文は次の通り。

```
class クラス名 {
private:
  変数定義…
public:
  関数宣言…
};            ←この「;」を忘れないように!!!
```

private:部分はモジュール内のデータ (実体変数) で外部からは隠蔽される (厳密にはモジュール内ローカル関数もあってよい)。public:部分は外部インタフェースをなす関数 (メンバ関数) の宣言 (厳密にはデータを公開してもよいが ADT の主旨からいうと薦められない)。

メンバ関数の定義は

```
[型指定] クラス名::関数名(引数部…) { 文… }
```

つまり普通の関数定義とおなじ。そして、メンバ関数の中ではクラス宣言で定義した変数がローカル変数として使える。

特例として、クラス名と同じ名前の関数があり、これを「コンストラクタ」という。コンストラクタはクラスの実体が生成される時に呼ばれ、実体変数を初期設定するのがおもな役割。クラスを生成するには、

```
new クラス名(引数…)
```

により記憶場所を割り当てクラス実体へのポインタを得る。引数はコンストラクタ関数の引数。そしてメンバ関数を呼び出すには

```
p->メンバ関数名(引数…)
```

による。なお、メンバ関数やコンストラクタ以外に、C と同様の普通の関数もあってよい(main などがそう)。

2.2 ディスプレイ型の ADT

では早速、先の「円が出るだけ」のプログラムで、「ディスプレイ」関係のデータを ADT にしてみる。

```
/* t71.c --- Display ADT in C++ */

#include <X11/Xlib.h>  ←ヘッダファイルは C と兼用

class Disp {   ←ディスプレイの ADT
private:
  Display *dsp1;          ← X の Display
  Screen *scr1;           ← X の Screen
  Window root1;           ← X のルート窓
  unsigned long fg1, bg1; ←白と黒
  GC gc1, rgc1;           ← 2 つの GC
public:
  Disp::Disp();           ←コンストラクタ
  Display *dsp();         ←以下は各種の値を返す
  Screen *scr();
  Window root();
  unsigned long fg();
  unsigned long bg();
  GC gc();
  GC rgc();
};

Disp::Disp() {        ← Disp は最初に 1 個だけ作るつもり。
  dsp1 = XOpenDisplay(NULL);  ←各種初期設定
  scr1 = DefaultScreenOfDisplay(dsp1);
  root1 = DefaultRootWindow(dsp1);
  fg1 = BlackPixelOfScreen(scr1);
  bg1 = WhitePixelOfScreen(scr1);
  gc1 = DefaultGC(dsp1, 0);
```

```
  XGCValues gcv; gcv.function = GXinvert; gcv.line_width = 3;
  rgc1 = XCreateGC(dsp1, root1, GCFunction|GCLineWidth, &gcv);
}

Display *Disp::dsp() { return dsp1; }  ←アクセス関数群
Screen *Disp::scr() { return scr1; }
Window Disp::root() { return root1; }
unsigned long Disp::fg() { return fg1; }
unsigned long Disp::bg() { return bg1; }
GC Disp::gc() { return gc1; }
GC Disp::rgc() { return rgc1; }

int draw(Disp *d, Window w, int x, int y);  ← C++では宣言必須

main() {
  Disp *d = new Disp();  ←ここで Disp を作る
  Window mw = XCreateSimpleWindow(d->dsp(), d->root(),100,100,400,200,2,
                                  d->fg(), d->bg());
  XSelectInput(d->dsp(), mw, ButtonPressMask|KeyPressMask|ExposureMask);
  XMapWindow(d->dsp(), mw);
  while(1) {
    XEvent ev;
    XNextEvent(d->dsp(), &ev);
    if(ev.type == KeyPress)
      exit(0);
    else if(ev.type == Expose)
      draw(d, mw, 20, 20);
    else if(ev.type == ButtonPress)
      draw(d, mw, ev.xbutton.x, ev.xbutton.y);
  }
}

draw(Disp *d, Window w, int x, int y) {
  XFillArc(d->dsp(), w, d->gc(), x-20, y-20, 40, 40, 0, 360*64);
}
```

いかがですか? Disp という ADT ができただけで、main はあんまり変わっていない。それでもとにかく、「X の各種資源をアクセスするには Disp のインスタンスのアクセス関数で取ってくる」という統一ができただけで結構「美しい」と思うがどうでしょう。

2.3 Window の ADT

ADT は複数の実態が作れるが、上の Disp は 1 個だけしか作らないのであまりらしくなかった。そこで次に「窓」を ADT にして、その後複数の窓を開くことを考えてみよう。Disp クラスはそのままにして、その後を Win クラスと main の組み合わせに置き換える。まず class 定義から。

```
class Win {
private:
  Disp *dsp1;     ← Disp は内部にとっておく
  Window win1;    ←窓の本体
  int exit1;      ←窓を閉じてしまったかどうか?
```

```
public:
  Win(Disp *d, int x, int y, int w, int h);  ←コンストラクタ
  void destroy();           ←窓を閉じる
  int handle(XEvent *ev);  ←イベントの処理
private:
  void draw(int x, int y); ←ローカル関数。丸を描く。
};
```

まず窓を作るのはこれまで通り。

```
Win::Win(Disp *d, int x, int y, int w, int h) {
  exit1 = False;
  dsp1 = d;
  win1 = XCreateSimpleWindow(d->dsp(), d->root(),x,y,w,h,2,d->fg(), d->bg());
  XSelectInput(d->dsp(), win1, ButtonPressMask|KeyPressMask|ExposureMask);
  XMapWindow(d->dsp(), win1);
}
```

消すときはXDestroyWindowを使う。また、消したという旗を立てる。

```
void Win::destroy() {
  XDestroyWindow(dsp1->dsp(), win1); exit1 = True;
}
```

おもしろいのはhandle。これは、イベントを渡されて「処理しようとする」。ここでイベントが自分の受け持ち範囲なら処理した上でTrueを返すが、受け持ちでないなら何もせずFalseを返す。

```
int Win::handle(XEvent *ev) {
  if(exit1 || ev->xany.window != win1) return False;
  if(ev->type == KeyPress)
    exit(0);
  else if(ev->type == Expose)
    draw(20, 20);
  else if(ev->type == ButtonPress)
    draw(ev->xbutton.x, ev->xbutton.y);
  return True;
}
```

drawは前と同じだが、DispやWindowは中で持っているので渡さなくてよい。

```
void Win::draw(int x, int y) {
  XFillArc(dsp1->dsp(), win1, dsp1->gc(), x-20, y-20, 40, 40, 0, 360*64);
}
```

ではmain。

```
main() {
  Disp *d = new Disp();
  Win *w1 = new Win(d, 100, 100, 400, 200);
  Win *w2 = new Win(d, 110, 110, 400, 200);
  while(1) {
    XEvent ev;
    XNextEvent(d->dsp(), &ev);
    if(w1->handle(&ev))
```

```
      ;
    else if(w2->handle(&ev))
      ;
    else
      ;
  }
}
```

おもしろいでしょう?

演習 12 これを改造して、窓を 5 個開くようにしてみよ (できるだけ配列を利用する)。

演習 13 それぞれの窓で、丸が復活できるように改造せよ。

2.4 簡単なクラスを自作する

上のプログラムではどれかの窓でキーを押したとたんにすべての窓が終わってしまう。これではつまらないので、キーを押したらその窓だけ終わるようにしよう。そのため、「個数を数える」次のようなクラスを設計した。

```
Counter *c = new Counter(5);  ←初期値 5 のカウンタを作る。
c->add(-1);                   ←カウンタに値を足し込む。
while(c->value() > 0) ...     ←カウンタの現在値を参照。
```

演習 14 Counter のクラス定義を書け。

演習 15 Counter のメンバ関数定義を書け。

演習 16 Counter を使うように先のプログラムを直せ。具体的には

- main の先頭で窓の個数を初期値としたカウンタを用意。
- Win の実体変数とコンストラクタの引数にカウンタを追加。
- Win でキーが押されたら窓を消してカウンタを-1 する。
- main の無限ループを「カウンタが 0 以上ならループ」に変更。

2.5 表示対象物もオブジェクトに

次に、表示対象 (この場合は丸) もオブジェクトにする。

```
Circle *c = new Circle(d, w, r, x, y);  ←コンストラクタ
c->draw();  ←描く
```

カウンタよりずっと簡単でしょう?

演習 17 Circle クラス (定義、メンバ関数) を作成せよ。

演習 18 Win の実現を Circle を使うように変更せよ。

2.6 ここまでで完成したプログラムの全リスト

```
/* t74.c --- make circle an object */

#include <X11/Xlib.h>

class Counter {
private:
  int value1;
public:
  Counter(int n);
  int add(int n);
  int value();
};

Counter::Counter(int n) { value1 = n; }
int Counter::add(int n) { return value1 += n; }
int Counter::value() { return value1; }

class Disp {
private:
  Display *dsp1;
  Screen *scr1;
  Window root1;
  unsigned long fg1, bg1;
  GC gc1, rgc1;
public:
  Disp::Disp();
  Display *dsp();
  Screen *scr();
  Window root();
  unsigned long fg();
  unsigned long bg();
  GC gc();
  GC rgc();
};

Disp::Disp() {
  dsp1 = XOpenDisplay(NULL);
  scr1 = DefaultScreenOfDisplay(dsp1);
  root1 = DefaultRootWindow(dsp1);
  fg1 = BlackPixelOfScreen(scr1);
  bg1 = WhitePixelOfScreen(scr1);
  gc1 = DefaultGC(dsp1, 0);
  XGCValues gcv; gcv.function = GXinvert; gcv.line_width = 3;
  rgc1 = XCreateGC(dsp1, root1, GCFunction|GCLineWidth, &gcv);
}

Display *Disp::dsp() { return dsp1; }
Screen *Disp::scr() { return scr1; }
Window Disp::root() { return root1; }
unsigned long Disp::fg() { return fg1; }
```

```
unsigned long Disp::bg() { return bg1; }
GC Disp::gc() { return gc1; }
GC Disp::rgc() { return rgc1; }

class Circle {
private:
  Disp *dsp1;
  Window win1;
  int radius1, xpos1, ypos1;
public:
  Circle(Disp *d, Window w, int r, int x, int y);
  void draw();
};

Circle::Circle(Disp *d, Window w, int r, int x, int y) {
  dsp1 = d; win1 = w; radius1 = r; xpos1 = x; ypos1 = y;
}

void Circle::draw() {
  XFillArc(dsp1->dsp(), win1, dsp1->gc(),
           xpos1-radius1, ypos1-radius1, radius1*2, radius1*2, 0, 360*64);
}

class Win {
private:
  Disp *dsp1;
  Window win1;
  Counter *cnt1;
  int exit1;
  Circle *objs1[100];
  int objuse1;
public:
  Win(Disp *d, int x, int y, int w, int h, Counter *c);
  void destroy();
  int handle(XEvent *ev);
private:
  void drawall();
};

Win::Win(Disp *d, int x, int y, int w, int h, Counter *c) {
  exit1 = False;
  cnt1 = c;
  dsp1 = d;
  objuse1 = 0;
  win1 = XCreateSimpleWindow(d->dsp(), d->root(),x,y,w,h,2,d->fg(), d->bg());
  XSelectInput(d->dsp(), win1, ButtonPressMask|KeyPressMask|ExposureMask);
  XMapWindow(d->dsp(), win1);
}

void Win::destroy() {
  XDestroyWindow(dsp1->dsp(), win1); exit1 = True;
}
```

```
int Win::handle(XEvent *ev) {
  if(exit1 || ev->xany.window != win1) return False;
  if(ev->type == KeyPress) {
    cnt1->add(-1); destroy(); }
  else if(ev->type == Expose)
    drawall();
  else if(ev->type == ButtonPress) {
    objs1[objuse1] = new Circle(dsp1, win1, 20, ev->xbutton.x, ev->xbutton.y);
    objs1[objuse1]->draw();
    ++objuse1; }
  return True;
}

void Win::drawall() {
  int i;
  for(i = 0; i < objuse1; ++i) objs1[i]->draw();
}

main() {
  Counter *c = new Counter(5);
  Disp *d = new Disp();
  Win *a[5];
  int i;
  for(i = 0; i < 5; ++i) a[i] = new Win(d, 100, 100, 400, 200, c);
  while(c->value() > 0) {
    XEvent ev;
    XNextEvent(d->dsp(), &ev);
    for(i = 0; i < 5; ++i)
      if(a[i]->handle(&ev)) break;
  }
}
```

2.7 別の種類の窓をつくる

さて、ここまでは「丸」を描く窓しかなかったが、もっと別の種類の窓を用意してみよう。たとえば、丸と四角を切り替えたりプログラムを終わらせるのに「スイッチ」のように使える窓を考える。

```
class SwWin {
private:
  Disp *dsp1;
  Window win1;
  char **lines1;
  int lnum1, sel1, exit1, width1, height1;
public:
  SwWin(Disp *d, int x, int y, int w, int h, char **l, int n);
  void destroy();
  int handle(XEvent *ev);
  int sel();
  void drawall();
};
```

つまり、作成するときにメニューとして使う「文字列の配列」を用意し、これを表示する窓を作る。そして、マウスでクリックすることで「選択肢」を変更できる。メンバ関数は次の通り。

```
SwWin::SwWin(Disp *d, int x, int y, int w, int h, char **l, int n) {
  exit1 = False; dsp1 = d; lines1 = l;
  lnum1 = n; sel1 = 0; width1 = w; height1 = h;
  win1 = XCreateSimpleWindow(d->dsp(), d->root(),x,y,w,h,2,d->fg(), d->bg());
  XSelectInput(d->dsp(), win1, ButtonPressMask|KeyPressMask|ExposureMask);
  XMapWindow(d->dsp(), win1);
}

void SwWin::destroy() {
  XDestroyWindow(dsp1->dsp(), win1); exit1 = True;
}

int SwWin::handle(XEvent *ev) {
  if(exit1 || ev->xany.window != win1) return False;
  if(ev->type == KeyPress) {
    if(++sel1 >= lnum1) sel1 = 0; }
  else if(ev->type == Expose)
    drawall();
  else if(ev->type == ButtonPress) {
    sel1 = ev->xbutton.y * lnum1 / height1;
    if(sel1 >= lnum1) sel1 = lnum1 - 1;
    if(sel1 < 0) sel1 = 0;
    drawall(); }
  return True;
}

int SwWin::sel() {
  return sel1;
}

void SwWin::drawall() {
  int i;
  int d = height1 / lnum1;
  XClearWindow(dsp1->dsp(), win1);
  for(i = 0; i < lnum1; ++i) {
    XDrawImageString(dsp1->dsp(), win1, dsp1->gc(), 4, d*(i+1)-5,
                     lines1[i], strlen(lines1[i])); }
  XFillRectangle(dsp1->dsp(), win1, dsp1->rgc(), 0, d*sel1, width1, d);
}
```

これを使って、丸と四角と「終了」を選択する窓を増やして見た。

```
main() {
  static char *menu[] = { "Circle", "Rectangle", "Quit" };
  Counter *c = new Counter(5);
  Disp *d = new Disp();
  Win *a[5];
  int i;
  SwWin *sw = new SwWin(d, 100, 100, 100, 60, menu, 3);
  sw->drawall();
```

```
  for(i = 0; i < 5; ++i) a[i] = new Win(d, 100, 100, 400, 200, c);
  while(sw->sel() != 2 && c->value() > 0) {
    XEvent ev;
    XNextEvent(d->dsp(), &ev);
    if(sw->handle(&ev))
      ;
    else
      for(i = 0; i < 5; ++i)
        if(a[i]->handle(&ev)) break;
  }
}
```

演習 19 これを直して、四角も打てるようにするにはどうしたらいいか。

2.8 動的配分 (dynamic dispatch)

さて、オブジェクト指向の利点の一つとしては、上で述べたように ADT が実現されるというのがあるのですが、次に残りの利点つまり

- 動的配分による汎用コード
- 継承によるコードの共有

を取り上げます。(継承は次回にしようと思っていたのですが、皆様の様子から見るとじっくりやっても苦しいだけみたいですから、今日さっさと済ませて次回はおまけのテーマにします。)

まず、動的配分とは「オブジェクトの種類に関わらず似たようなオブジェクトには似たようなメッセージが送れる」「その結果の動作はオブジェクトの種類に応じたものになる」ということです。(また要演説ですね。)

問題は、C++のように強い型の言語では「オブジェクトの種類が違うと型も違うため、それらを一緒の変数には入れられない」という制約があることです。そこで、次の方法を取ります。

- クラスに親子関係を導入して、「類似したもの」を共通の親の子供 (ないし子孫) としてまとめる。
- 親クラスのポインタ型には、子孫のクラスのポインタ値が入れられる。

具体的には、前回「Circle」をオブジェクト (クラス) にしていましたが、これと独立に「Rect」を作る変わりに「DrawObj」(描けるもの) という親クラスを作り、Circle や Rect をその子孫にします。

```
class DrawObj {
public:
  virtual void draw() { }
};
```

この DrawObj というクラスは実体変数は全くなく、draw というメソッドがあるだけです。しかもこのメソッドは何もしません! ({ }のところ。)

ではこれは何のためにあるのでしょう。draw というメソッドは、子供のクラスでそれぞれ固有のメソッドに置き換えます。そして、

```
  DrawObj *o = ...
  ...
  o->draw();
```

というのがあると、**現在 o に入っている『もの』に応じてその draw が呼ばれる**わけです (動的配分)。では子供のクラス。

```
class Circle : public DrawObj { ←ここで親を指定
private:
  Disp *dsp1;
  Window win1;
  int x1, y1, w1, h1;
public:
  Circle(Disp *d, Window m, int x, int y, int r);
  void draw();
};

Circle::Circle(Disp *d, Window m, int x, int y, int r) {
  dsp1 = d; win1 = m; x1 = x-r; y1 = y-r; w1 = h1 = r*2;
}

void Circle::draw() {
  XFillArc(dsp1->dsp(), win1, dsp1->gc(), x1, y1, w1, h1, 0, 360*64);
}
```

Rect も同様。

```
class Rect : public DrawObj {
private:
  Disp *dsp1;
  Window win1;
  int x1, y1, w1, h1;
public:
  Rect(Disp *d, Window m, int x, int y, int w, int h);
  void draw();
};

Rect::Rect(Disp *d, Window m, int x, int y, int w, int h) {
  dsp1 = d; win1 = m; x1 = x; y1 = y; w1 = w; h1 = h;
}

void Rect::draw() {
  XFillRectangle(dsp1->dsp(), win1, dsp1->gc(), x1, y1, w1, h1);
}
```

では本体を示します。

```
/* t81.c --- dynamic dispatch */

#include <X11/Xlib.h>
#include <X11/Xutil.h>

class Counter {
private:
```

```
  int value1;
public:
  Counter(int n);
  int add(int n);
  int value();
};

Counter::Counter(int n) { value1 = n; }
int Counter::add(int n) { return value1 += n; }
int Counter::value() { return value1; }

class Disp {
private:
  Display *dsp1;
  Screen *scr1;
  Window root1;
  unsigned long fg1, bg1;
  GC gc1, rgc1;
public:
  Disp::Disp();
  Display *dsp();
  Screen *scr();
  Window root();
  unsigned long fg();
  unsigned long bg();
  GC gc();
  GC rgc();
};

Disp::Disp() {
  dsp1 = XOpenDisplay(NULL);
  scr1 = DefaultScreenOfDisplay(dsp1);
  root1 = DefaultRootWindow(dsp1);
  fg1 = BlackPixelOfScreen(scr1);
  bg1 = WhitePixelOfScreen(scr1);
  gc1 = DefaultGC(dsp1, 0);
  XGCValues gcv;
  gcv.function = GXinvert; gcv.line_width = 3;
  rgc1 = XCreateGC(dsp1, root1, GCFunction|GCLineWidth, &gcv);
}

Display *Disp::dsp() { return dsp1; }
Screen *Disp::scr() { return scr1; }
Window Disp::root() { return root1; }
unsigned long Disp::fg() { return fg1; }
unsigned long Disp::bg() { return bg1; }
GC Disp::gc() { return gc1; }
GC Disp::rgc() { return rgc1; }

// DrawObj 類はここに入る。

class DrawWin {
```

```
private:
  Disp *dsp1;
  Window win1;
  Counter *cnt1;
  int exit1;
  DrawObj *objs1[100];
  int objuse1;
public:
  DrawWin(Disp *d, int x, int y, int w, int h, Counter *c);
  void destroy();
  int handle(XEvent *ev);
  void drawall();
};

DrawWin::DrawWin(Disp *d, int x, int y, int w, int h, Counter *c) {
  exit1 = False;
  cnt1 = c;
  dsp1 = d;
  objuse1 = 0;
  win1 = XCreateSimpleWindow(d->dsp(), d->root(),x,y,w,h,2,d->fg(), d->bg());
  XSelectInput(d->dsp(), win1, ButtonPressMask|KeyPressMask|ExposureMask);
  XMapWindow(d->dsp(), win1);
}

void DrawWin::destroy() {
  cnt1->add(-1); XDestroyWindow(dsp1->dsp(), win1); exit1 = True;
}

int DrawWin::handle(XEvent *ev) {
  if(exit1 || ev->xany.window != win1) return False;
  if(ev->type == KeyPress) {
    char buf[20];
    if(XLookupString(&ev->xkey,buf,20,0,0) == 1 && buf[0] == 'q') destroy(); }
  else if(ev->type == Expose)
    drawall();
  else if(ev->type == ButtonPress) {
    DrawObj *o;
    if(ev->xbutton.state&ShiftMask)
      o = new Rect(dsp1, win1, ev->xbutton.x-20, ev->xbutton.y-20, 40, 40);
    else
      o = new Circle(dsp1, win1, ev->xbutton.x, ev->xbutton.y, 20);
    o->draw(); objs1[objuse1] = o; ++objuse1; }
  return True;
}

void DrawWin::drawall() {
  XClearWindow(dsp1->dsp(), win1);
  for(int i = 0; i < objuse1; ++i) objs1[i]->draw();
}

main() {
  Counter *c = new Counter(3);
```

```
  Disp *d = new Disp();
  DrawWin *a[3];
  for(int i = 0; i < 3; ++i) a[i] = new DrawWin(d, 100, 100, 400, 200, c);
  while(c->value() > 0) {
    XEvent ev;
    XNextEvent(d->dsp(), &ev);
    for(i = 0; i < 3; ++i)
      if(a[i]->handle(&ev)) break;
  }
}
```

演習 20 このまま動かせ。

演習 21 三角形オブジェクトを作成し追加せよ。(塗らなくてよい。)

2.9 実現の継承

上のではインタフェースのみを継承していたが、dsp1 や win1 などの実体変数は共通していましたね? そこで、これらを親クラスにまとめることを考える。

```
class DrawObj {
protected:
  Disp *dsp1;
  Window win1;
  int x1, y1, w1, h1;
public:
  DrawObj(Disp *d, Window m);
  virtual void draw() { }
};

DrawObj::DrawObj(Disp *d, Window m) {
  dsp1 = d; win1 = m;
}
```

protected: というのは、外からはアクセスできないが、子孫のクラスからはアクセスできるという意味。子孫のクラスにおける実体変数群は親クラスの実体変数をすべて集めたものになる。そして、実体変数があると普通はコンストラクタも必要。で、子クラスのコンストラクタからは親クラスのコンストラクタを呼び出すのが普通。

```
class Circle : public DrawObj {
public:
  Circle(Disp *d, Window m, int x, int y, int r);
  void draw();
};

Circle::Circle(Disp *d, Window m, int x, int y, int r) : DrawObj(d, m) {
  x1 = x-r; y1 = y-r; w1 = h1 = r*2;
}

void Circle::draw() {
  XFillArc(dsp1->dsp(), win1, dsp1->gc(), x1, y1, w1, h1, 0, 360*64);
}
```

```
class Rect : public DrawObj {
public:
  Rect(Disp *d, Window m, int x, int y, int w, int h);
  void draw();
};

Rect::Rect(Disp *d, Window m, int x, int y, int w, int h) : DrawObj(d, m) {
  x1 = x; y1 = y; w1 = w; h1 = h;
}

void Rect::draw() {
  XFillRectangle(dsp1->dsp(), win1, dsp1->gc(), x1, y1, w1, h1);
}
```

演習 22 さっきのクラスをこの形に直せ。

2.10 親クラスにおけるメソッド

ところで、データ構造が共通ということは、それに関する操作にも共通部分が存在してよいということ。たとえば、「移動」や「リサイズ」という操作をできるようにするには、各オブジェクトの位置や大きさをアクセスしたり変更したりできるようにしたい。そういうメソッドは DrawObj につけておけばよい。

```
class DrawObj {
protected:
  Disp *dsp1;
  Window win1;
  int x1, y1, w1, h1;
public:
  DrawObj(Disp *d, Window m);
  virtual int getx();
  virtual int gety();
  virtual int getw();
  virtual int geth();
  virtual void move(int x, int y);
  virtual void resize(int w, int h);
  virtual int hit(int x, int y);
  virtual void draw() { }
  virtual void drawrubber();
};

DrawObj::DrawObj(Disp *d, Window m) {
  dsp1 = d; win1 = m;
}

int DrawObj::getx() { return x1; }
int DrawObj::gety() { return y1; }
int DrawObj::getw() { return w1; }
int DrawObj::geth() { return h1; }
void DrawObj::move(int x, int y) { x1 = x; y1 = y; }
void DrawObj::resize(int w, int h) { w1 = w; h1 = h; }
```

```
int DrawObj::hit(int x, int y) {
  return x1 <= x && x <= x1+w1 && y1 <= y && y <= y1+h1;
}

void DrawObj::drawrubber() {
  XDrawRectangle(dsp1->dsp(), win1, dsp1->rgc(), x1, y1, w1, h1);
}
```

これらを利用してオブジェクトの移動を行うように直したDrawWinを示す。

```
class DrawWin {
private:
  Disp *dsp1;
  Window win1;
  Counter *cnt1;
  int exit1;
  DrawObj *objs1[100], *moving1;
  int objuse1;
  int dx1, dy1;
public:
  DrawWin(Disp *d, int x, int y, int w, int h, Counter *c);
  void destroy();
  int handle(XEvent *ev);
  void drawall();
};

DrawWin::DrawWin(Disp *d, int x, int y, int w, int h, Counter *c) {
  exit1 = False;
  cnt1 = c;
  dsp1 = d;
  objuse1 = 0;
  moving1 = NULL;
  win1 = XCreateSimpleWindow(d->dsp(), d->root(),x,y,w,h,2,d->fg(), d->bg());
  XSelectInput(d->dsp(), win1, ButtonMotionMask|ButtonReleaseMask
    |ButtonPressMask|KeyPressMask|ExposureMask);
  XMapWindow(d->dsp(), win1);
}

void DrawWin::destroy() {
  cnt1->add(-1); XDestroyWindow(dsp1->dsp(), win1); exit1 = True;
}

int DrawWin::handle(XEvent *ev) {
  if(exit1 || ev->xany.window != win1) return False;
  if(ev->type == KeyPress) {
    char buf[20];
    if(XLookupString(&ev->xkey,buf,20,0,0) == 1 && buf[0] == 'q') destroy(); }
  else if(ev->type == Expose)
    drawall();
  else if(ev->type == ButtonPress && ev->xbutton.button == Button1) {
    DrawObj *o;
    if(ev->xbutton.state&ShiftMask)
      o = new Rect(dsp1, win1, ev->xbutton.x-20, ev->xbutton.y-20, 40, 40);
```

```
    else
      o = new Circle(dsp1, win1, ev->xbutton.x, ev->xbutton.y, 20);
    o->draw(); objs1[objuse1] = o; ++objuse1; }
  else if(ev->type == ButtonPress && ev->xbutton.button == Button2) {
    for(int i = 0; i < objuse1; ++i)
      if(objs1[i]->hit(ev->xbutton.x, ev->xbutton.y)) {
        moving1 = objs1[i]; moving1->drawrubber();
        dx1 = moving1->getx() - ev->xbutton.x;
        dy1 = moving1->gety() - ev->xbutton.y;
        break; }
  }
  else if(ev->type == MotionNotify) {
    if(moving1) {
      moving1->drawrubber();
      moving1->move(ev->xbutton.x+dx1, ev->xbutton.y+dy1);
      moving1->drawrubber(); }
  }
  else if(ev->type == ButtonRelease) {
    if(moving1) {
      moving1->drawrubber(); drawall(); moving1 = NULL; }
  }
  return True;
}

void DrawWin::drawall() {
  XClearWindow(dsp1->dsp(), win1);
  for(int i = 0; i < objuse1; ++i) objs1[i]->draw();
}
```

ところで、移動中のラバーバンドが丸くないと気持ちが悪いですね? それにはCircleでdrawrubberを用意すればよい。このように、子クラスで親クラスにあるメソッドを再定義することを「オーバライド」とよぶ。オブジェクト指向だとこのように子供クラスを追加してはそこにメソッドを増やして行くことで徐々にプログラムを成長させられる。

演習 23 これらの変更を混ぜて動かせ。

演習 24 Circleのラバーバンドを丸くしてみよ。

演習 25 自分の作ったもののラバーバンドも作れ。

演習 26 大きさ変更もできるようにしてみよ。

2.11 複合オブジェクト

ところで、丸と四角が接した新しいオブジェクトを作りたくなったとしよう。その場合、またぞろXDrawなんとかを使わなくてもいい。なぜか? それは、**既存のオブジェクトを組み合わせて**使えばよいから。

演習 27 丸四角オブジェクトを作成してみよ。

2.12 オブジェクトのたどり

今度は、たとえばプリントアウトを考えてみよう。つまり、図形の各要素を PostScript に変換してファイルに出力する (PostScript は覚えていますね?)。それには、DrawWin で「p」というコマンドを打った時に PS ファイルをオープンして用意し、各オブジェクトに「自分を書き出せ」といい、最後に後始末すればよい。完成版のプログラム一式を最後に示す。

```
/* t84.c --- print out */

#include <stdio.h>
#include <X11/Xlib.h>
#include <X11/Xutil.h>

class Counter {
private:
  int value1;
public:
  Counter(int n);
  int add(int n);
  int value();
};

Counter::Counter(int n) { value1 = n; }
int Counter::add(int n) { return value1 += n; }
int Counter::value() { return value1; }

class Disp {
private:
  Display *dsp1;
  Screen *scr1;
  Window root1;
  unsigned long fg1, bg1;
  GC gc1, rgc1;
public:
  Disp::Disp();
  Display *dsp();
  Screen *scr();
  Window root();
  unsigned long fg();
  unsigned long bg();
  GC gc();
  GC rgc();
};

Disp::Disp() {
  dsp1 = XOpenDisplay(NULL);
  scr1 = DefaultScreenOfDisplay(dsp1);
  root1 = DefaultRootWindow(dsp1);
  fg1 = BlackPixelOfScreen(scr1);
  bg1 = WhitePixelOfScreen(scr1);
  gc1 = DefaultGC(dsp1, 0);
  XGCValues gcv;
  gcv.function = GXinvert; gcv.line_width = 3;
```

```
  rgc1 = XCreateGC(dsp1, root1, GCFunction|GCLineWidth, &gcv);
}

Display *Disp::dsp() { return dsp1; }
Screen *Disp::scr() { return scr1; }
Window Disp::root() { return root1; }
unsigned long Disp::fg() { return fg1; }
unsigned long Disp::bg() { return bg1; }
GC Disp::gc() { return gc1; }
GC Disp::rgc() { return rgc1; }

class DrawObj {
protected:
  Disp *dsp1;
  Window win1;
  int x1, y1, w1, h1;
public:
  DrawObj(Disp *d, Window m);
  virtual int getx();
  virtual int gety();
  virtual int getw();
  virtual int geth();
  virtual void move(int x, int y);
  virtual void resize(int w, int h);
  virtual int hit(int x, int y);
  virtual void draw() { }
  virtual void drawrubber();
  virtual void postscript(FILE *f);
};

DrawObj::DrawObj(Disp *d, Window m) {
  dsp1 = d; win1 = m;
}

int DrawObj::getx() { return x1; }
int DrawObj::gety() { return y1; }
int DrawObj::getw() { return w1; }
int DrawObj::geth() { return h1; }
void DrawObj::move(int x, int y) { x1 = x; y1 = y; }
void DrawObj::resize(int w, int h) { w1 = w; h1 = h; }

int DrawObj::hit(int x, int y) {
  return x1 <= x && x <= x1+w1 && y1 <= y && y <= y1+h1;
}

void DrawObj::drawrubber() {
  XDrawRectangle(dsp1->dsp(), win1, dsp1->rgc(), x1, y1, w1, h1);
}

void DrawObj::postscript(FILE *f) {
  fprintf(f, "newpath %d %d moveto %d 0 rlineto\n", x1, y1, w1);
  fprintf(f, "0 %d rlineto -%d 0 rlineto closepath fill\n", h1, w1);
```

```
}

class Circle : public DrawObj {
public:
  Circle(Disp *d, Window m, int x, int y, int r);
  void draw();
  void postscript(FILE *f);
};

Circle::Circle(Disp *d, Window m, int x, int y, int r) : DrawObj(d, m) {
  x1 = x-r; y1 = y-r; w1 = h1 = r*2;
}

void Circle::draw() {
  XFillArc(dsp1->dsp(), win1, dsp1->gc(), x1, y1, w1, h1, 0, 360*64);
}

void Circle::postscript(FILE *f) {
  int cx = x1 + w1 / 2;
  int cy = y1 + h1 / 2;
  fprintf(f, "gsave %d %d translate %d %d scale\n", cx, cy, w1/2, h1/2);
  fprintf(f, "newpath 0 0 1 0 360 arc closepath fill grestore\n");
}

class Rect : public DrawObj {
public:
  Rect(Disp *d, Window m, int x, int y, int w, int h);
  void draw();
};


Rect::Rect(Disp *d, Window m, int x, int y, int w, int h) : DrawObj(d, m) {
  x1 = x; y1 = y; w1 = w; h1 = h;
}

void Rect::draw() {
  XFillRectangle(dsp1->dsp(), win1, dsp1->gc(), x1, y1, w1, h1);
}

class DrawWin {
private:
  Disp *dsp1;
  Window win1;
  Counter *cnt1;
  int exit1;
  DrawObj *objs1[100], *moving1, *res1;
  int objuse1;
  int dx1, dy1;
public:
  DrawWin(Disp *d, int x, int y, int w, int h, Counter *c);
  void destroy();
  int handle(XEvent *ev);
```

```
  void drawall();
};

DrawWin::DrawWin(Disp *d, int x, int y, int w, int h, Counter *c) {
  exit1 = False;
  cnt1 = c;
  dsp1 = d;
  objuse1 = 0;
  moving1 = NULL;
  res1 = NULL;
  win1 = XCreateSimpleWindow(d->dsp(), d->root(),x,y,w,h,2,d->fg(), d->bg());
  XSelectInput(d->dsp(), win1, ButtonMotionMask|ButtonReleaseMask
    |ButtonPressMask|KeyPressMask|ExposureMask);
  XMapWindow(d->dsp(), win1);
}

void DrawWin::destroy() {
  cnt1->add(-1); XDestroyWindow(dsp1->dsp(), win1); exit1 = True;
}

int DrawWin::handle(XEvent *ev) {
  if(exit1 || ev->xany.window != win1) return False;
  if(ev->type == KeyPress) {
    char buf[20];
    if(XLookupString(&ev->xkey, buf, 20, 0, 0) != 1)
      ;
    else if(buf[0] == 'p') {
      FILE *f = fopen("out.ps", "w");
      fprintf(f, "20 500 translate 1 -1 scale 0.5 setgray\n");
      for(int i = 0; i < objuse1; ++i) objs1[i]->postscript(f);
      fprintf(f, "showpage\n"); fclose(f); }
    else if(buf[0] == 'q')
      destroy();
  }
  else if(ev->type == Expose)
    drawall();
  else if(ev->type == ButtonPress && ev->xbutton.button == Button1) {
    DrawObj *o;
    if(ev->xbutton.state&ShiftMask)
      o = new Rect(dsp1, win1, ev->xbutton.x-20, ev->xbutton.y-20, 40, 40);
    else
      o = new Circle(dsp1, win1, ev->xbutton.x, ev->xbutton.y, 20);
    o->draw(); objs1[objuse1] = o; ++objuse1; }
  else if(ev->type == ButtonPress && ev->xbutton.button == Button2) {
    for(int i = 0; i < objuse1; ++i)
      if(objs1[i]->hit(ev->xbutton.x, ev->xbutton.y)) {
        moving1 = objs1[i]; moving1->drawrubber();
        dx1 = moving1->getx() - ev->xbutton.x;
        dy1 = moving1->gety() - ev->xbutton.y;
        break; }
  }
  else if(ev->type == ButtonPress && ev->xbutton.button == Button3) {
```

```
    for(int i = 0; i < objuse1; ++i)
      if(objs1[i]->hit(ev->xbutton.x, ev->xbutton.y)) {
        res1 = objs1[i]; res1->drawrubber();
        dx1 = res1->getx()+res1->getw() - ev->xbutton.x;
        dy1 = res1->gety()+res1->geth() - ev->xbutton.y;
        break; }
  }
  else if(ev->type == MotionNotify) {
    if(moving1) {
      moving1->drawrubber();
      moving1->move(ev->xbutton.x+dx1, ev->xbutton.y+dy1);
      moving1->drawrubber(); }
    if(res1) {
      res1->drawrubber();
      int w1 = ev->xbutton.x + dx1 - res1->getx(); if(w1 < 10) w1 = 10;
      int h1 = ev->xbutton.y + dy1 - res1->gety(); if(h1 < 10) h1 = 10;
      res1->resize(w1, h1); res1->drawrubber(); }
  }
  else if(ev->type == ButtonRelease) {
    if(moving1) {
      moving1->drawrubber(); drawall(); moving1 = NULL; }
    if(res1) {
      res1->drawrubber(); drawall(); res1 = NULL; }
  }
  return True;
}

void DrawWin::drawall() {
  XClearWindow(dsp1->dsp(), win1);
  for(int i = 0; i < objuse1; ++i) objs1[i]->draw();
}

main() {
  Counter *c = new Counter(3);
  Disp *d = new Disp();
  DrawWin *a[3];
  for(int i = 0; i < 3; ++i) a[i] = new DrawWin(d, 100, 100, 400, 200, c);
  while(c->value() > 0) {
    XEvent ev;
    XNextEvent(d->dsp(), &ev);
    for(int i = 0; i < 3; ++i)
      if(a[i]->handle(&ev)) break;
  }
}
```

演習 28 自分のオブジェクトもプリントアウトできるようにせよ。