簡易画像エディタを作成しよう

立木秀樹

November 10, 2015

1 DrawGraphics.java

ボタンで、長方形、楕円、線分をえらび、マウスで始点を指定し、ドラッグして離すことにより終点を指定する。

- ボタンの配置は、ボタンを置くパネルを別に用意し、それと描画機能を拡張したパネル (MyCanvas クラスという名前にした) を BoxLayout で縦に並べている。
- type という変数に、現在選ばれている種類を 1, 2, 3 として記憶。
- 1,2,3 は、RECT, OVAL, LINE という名前の final なクラス変数として、プログラムの中ではこの名前でアクセスするようにする。
- マウスが押された座標を ex, ey, 離された座標を x, y という変数で持っている。
- 描画領域のパネル (MyCanvas) は、JPanel のサブクラスとし、DrawGraphics の内部クラスとして定義している。
- 内部クラスのオブジェクトは、外のクラスのメソッドやコンストラクタからしか new できない。よって、各内部クラスのオブジェクトには、その外側のオブジェクトが唯一存在している。この外側のオブジェクトのローカル変数に名前だけでアクセスできる。これにより、DrawGraphics が持っている変数 x, y, ex, ey に、MyCanvas からアクセスできる。

2 DrawGraphics1.java

DrawGraphics.java を、匿名クラスを用いて書き換えよう。クラスによっては、MyCanvas のように、１つしかオブジェクトが作られないクラスがある。その場合、オブジェクトを作成する場所でクラスの定義を行うことにより、名前をつけずにクラスを定義することができる。これを匿名クラスという。匿名クラスは内部クラスであり、そのオブジェクト作成は、

```
new スーパークラス名 () { クラスの定義}
```

で行われる。匿名クラスではコンストラクターを定義できない。MyCanvas の部分を匿名クラスにしてみよう。MyCanvas 型はもはやないので、このクラスのオブジェクトを格納する変数 (area) の型は、もはや、MyCanvas にできない。しかし、スーパークラス JPanel 型の変数に代入できる。

DrawGraphicvs.java では Mouse やボタンのイベントのリスナーを自分自身にしていたが、自分とは別に内部クラスを定義して、それをリスナークラスにすることができる。MouseClicked などの、MouseListener の５つのリスナーメソッドを（何もしないメソッドとして）もつ、MouseAdapter クラスがある。そのサブクラスとしてリスナークラスを定義することにより、使わないメソッドの定義を省略できる。さらに、匿名クラスにすると、名前をつける必要もなくなる。MouseAdapter, MouseMotionAdapter を利用してみよう。

ActionListener は、１つしかメソッドを持たないので、対応するアダプタークラスはない。しかし、これを匿名クラスとして定義することにより、複数のボタンがある時に、ボタンごとに異なるリスナークラスを定義することができる。匿名クラスは、インターフェースを指定して作ることもできる。

```
new インターフェース名 () { クラスの定義}
```

匿名クラスは便利ではあるが、慣れないとかえって分かりにくいプログラムになってしまうので、以下では、DrawGraphics.java を拡張して考える。

3 DrawGraphics2.java

DrawGraphics では、次の図形を描くと前の図形が消えてしまった。複数の図形を合わせた絵を描くことのできるものにしよう。この機能は、オフスクリーンイメージを用いて実現できる。

```
BufferedImage bi = new BufferedImage(X, Y,BufferedImage.TYPE_INT_RGB);
Graphics bg = bi.createGraphics();
```

により、イメージオブジェクト bi と、その上に描画するための Graphics オブジェクト bg を作れる。 bg をもちいて bi に絵を描き、paintComponent では、bi を JPanel に drawImage によりコピーすればよい。

描画途中の線は、bi には描かずに、paintComponent の段階で JPanel に赤で描くようにする。

描かれた絵をファイルに画像ファイルとして保存する方法を追加しよう。147 ページ参照。色を変えるなどのことができるようにしよう。

4 DrawGraphics3.java

すでに描いた図形を、図形ごとに消去したり、編集しなおしたりできるようにしよう。undo により、最後に描いた図形を消すことができる。すでに描かれた図形を指定して、編集し直すには、線分は描き終わった時のマウスの位置を、Ctrl キーを押しながら指定する。この方法で図形を指定して undo すると、図形を消すことができる。

それには、絵全体ではなく、どういう図形が描かれているかという図形のリストの情報を記憶しておく必要がある。それに LinkedList を用いる。(もちろん、ArrayList でもよい。) LinkedList には、図形を表す Figure クラスのオブジェクトが格納される。Figure オブジェクトには、インスタンス変数として x, y, ex, ey, type の値を記憶しておくことにする。

イベント e が起きた時に CTRL が押されていたかどうかは、

```
if((e.getModifiers() & InputEvent.CTRL_MASK) > 0
```

で判断できる。`e.getModifiers()` は CTRL などのキーの状態をビット列で表現する。`InputEvent.CTRL_MASK` は、CTRL に対応するビットだけの立ったビット列の表現する数、`&` はビットごとの and である。

5 DrawGraphics4.java

ここまでの図形は、四角形、円、線分という、左上と右下の２つの頂点の座標で指定できるものばかりであった。それ以外の図形も描画できるようにしよう。例えば、折れ線を表現できるようにしよう。折れ線は、ドラッグを終了する時にシフトを押していると、折れ線の頂点となって折れ線の描画が続き、シフトを押していないと最後の頂点となって折れ線の描画が終わるようにしよう。ついでに、右上から左下だけではなく、様々な方向に四角形や円を描けるようにしよう。イベント発生時にシフトを押されているかは、`InputEvent.SHIFT_MASK` と `&` をとればよい。

折れ線は、x, y, ex, ey という４つの座標だけでは表現できない。よって、表現する図形に応じて、クラスを変える必要がある。Figure クラスは abstract class とし、そのサブクラスとして、四角形、円、線分、折れ線を表現するクラス、Rectangele, Oval, Line, Poly をそれぞれ定義しよう。後でマウスでクリックされた点から、編集する図形を探すためには、全てが共通して、マウスでクリックされた点に対応する x, y という変数を持つ必要がある。x, y は Figure クラスで定義するようにしよう。そして、描画するメソッドは、

```
abstract void drawit(Graphics g)
```

という抽象メソッドとし、Rectangle などのそれぞれのクラスで実現することにしよう。また、これまでは、描きかけの図形の頂点などの情報を DrawGraphics のインスタンス変数で持っていたが、描きかけの図形がどれかによって持つ情報が異なることになる。そのために、描きかけの図形も Figure とし、DrawGraphics の Figure 型の変数 cf に代入しておくことにしよう。そして、マウスが動くことにより x, y が変化した時に Figure の中の情報を変えるための

```
void update(int x1, int y1)
```

というメソッドを Figure クラスに用意しよう。

Poly は、x, y 以外に、(x, y) 以外の座標の情報も持つ必要がある。そのために、Point クラスを用意し、

```
class Point{int x, y; Point(int x, int y){this.x=x; this.y=y;}}
```

Point のリスト (LinkedList か ArrayList) points を Poly は持つことにしよう。Poly の drawit は、x, y と points をもとに、x 座標、y 座標のリストを作り drawPolyLine で折れ線を描くことになる。さらに、Poly は、新たに点が加わった時に、それを points に加えるメソッド

```
void newpoint(int x, int y){
```

が必要である。この図形エディタを本格的にするための拡張を考え、実現しよう。