

第六回 演習問題 (GUI2)

諸注意

- 今回、ファイルの提出は行わない。
- 課題ができた段階で教員もしくは TA を呼びその場でチェックを受ける。

課題 1

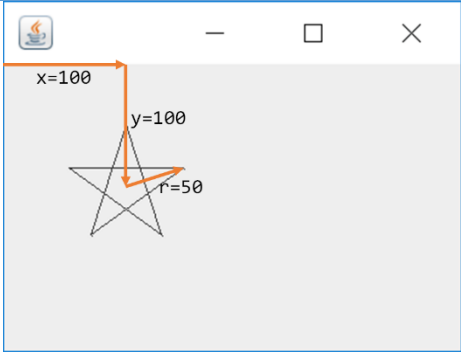
1	問題設定	座標 (x, y) を中心とし半径 r の五角形を描く際の頂点間を直線で結ぶことで星を描画することができる。これを drawStar(Graphics g, int x, int y, int r); で定義してほしい。図 2 にテスト例を示す。なお橙色矢印とそれに付随する文字列は説明用でありアプリ用で表示する必要はない。	
	テスト例	<pre>@Override public void paintComponent(Graphics g){ this.drawStar(g, 100, 100, 50); }</pre>	
	動作画面例		
	ヒント	練習問題 2 を参考にすると良い。もはや高校数学の話。drawPolygon() を応用して開発できるとカッコよい。	

図 2. 動作画面例

課題 1：解答例 (KadaiPanel1 内の必要箇所抜粋)

```
// 座標(cx, cy)を中心として、半径rの円に内接する星を描画するメソッド
// drawStar内でGraphics gは取得せず、引数のものを用いることで、
// paintComponent()から呼び出されることを想定している。
public void drawStar(Graphics g, int x, int y, int r){
    // 表示するPanel全体のサイズをDimension型で取得する。
    Dimension d = this.getSize();

    // 星の頂点5つを格納するための配列を準備する
    int[] xarr = new int[5];
    int[] yarr = new int[5];
    for(int i=0; i<5; i++){
        // 144度ずつずらすことで1点飛ばしで頂点を得る
        // これにより、その順序で線を結ぶと星になる。
        xarr[i] = x + (int)(r*Math.cos
            (Math.toRadians(90 + i*144)));
        yarr[i] = y - (int)(r*Math.sin
            (Math.toRadians(90 + i*144)));
    }
    // drawPolygon()は2点の配列を引数として渡すことで
    // 頂点を順にdrawLine()で結んでくれる
    g.drawPolygon(xarr, yarr, 5);
}
```

課題 2

2	問題設定	<p>お絵かきアプリを開発したい。マウスイベントをうまく用いることで、クリック時に黒線でお絵かきできるアプリを開発してほしい。ただし、授業中の例をシンプルに用いると、図3の様に途切れ途切れになってしまうことが懸念される。そこで、線の太さは 1px で構わないので、図4のように途切れないように改造して実装してほしい。</p> <p>本課題ではマウスイベント内の<code>this.getGraphics();</code>で取得した<code>Graphics</code>インスタンスに対して直接描画処理を実装してくれて良い（ダブルバッファリングしない）。</p> <p>ドラッグ中にマウスポインタがフレーム外に移動したケースは想定しなくて良い（発生しないものとする）。</p>
	動作画面例	<div data-bbox="564 757 858 981" data-label="Image"> </div> <div data-bbox="970 757 1264 981" data-label="Image"> </div> <div data-bbox="576 987 842 1021" data-label="Caption">図3. シンプルな例</div> <div data-bbox="991 987 1230 1021" data-label="Caption">図4. 正解画面例</div>
	ヒント	点と点を線でつなぐと. . .

課題 2：解答例（KadaiPanel2 内の必要箇所抜粋）

// 前回の座標(x,y)をフィールドとして保持する

`private int lastX=-1, lastY=-1;`

@Override

`public void mouseDragged(MouseEvent e) {`

// 描画用のGraphicsを取得する

`Graphics g = this.getGraphics();`

// 前回座標が、初期値である(-1, -1)以外の場合

`if(lastX != -1 && lastY != -1)`

// 前回座標→今回座標で直線を描画する

`g.drawLine(lastX, lastY, e.getX(), e.getY());`

// 前回座標を更新する

`lastX = e.getX();``lastY = e.getY();``}`

@Override

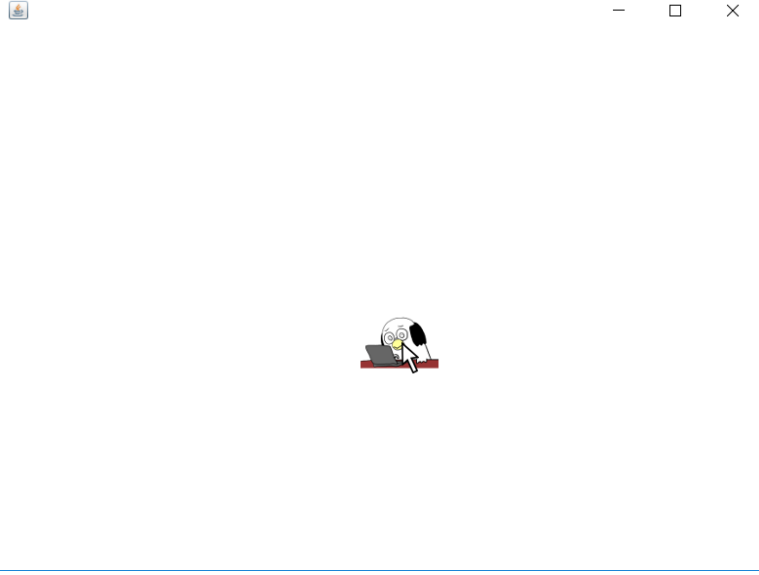
`public void mouseReleased(MouseEvent e) {`

// マウスを離したときに前回座標を初期化しないと、

// 再度ドラッグを開始したときに線がつながってしまう。

`this.lastX = -1;``this.lastY = -1;``}`

課題 3

3-1	問題設定	<p>一昔前（もっと前かもしれない）に流行った「マウスカーソルを変更する」に近い動作を実現したい。図3のように JPanel 上にマウスカーソルがある時に画像が追跡してくるような動作を実装せよ。ただし、マウスカーソルが画像の中心に来るように注意せよ。</p> <p>本課題ではマウスイベント内の <code>this.getGraphics();</code> で取得した <code>Graphics</code> インスタンスに対して直接描画処理を実装してくれて良い（ダブルバッファリングしない）。</p> <p>ドラッグ中にマウスポインタがフレーム外に移動したケースは想定しなくて良い（発生しないものとする）。</p> <p>画像は演習問題同様に Web から DL し使用すること。</p>
	動作画面例	 <p>図3. 動作画面例</p>
	ヒント	<p>参考までに、画像に関連するメソッドの使用例を下記に示す。</p> <pre>// Imageインスタンスの取得 Image img = (new ImageIcon("img.png")).getImage(); // Imageインスタンスのサイズの取得 int width = img.getWidth(this); int height = img.getHeight(this); // 座標(0, 0)にImageインスタンスを描画 g.drawImage(img, 0, 0, this);</pre>

課題 3-1：解答例 (KadaiPanel3 内の必要箇所抜粋)

```
@Override
public void mouseMoved(MouseEvent e) {
    // 描画用のGraphics gを取得する.
    Graphics g = this.getGraphics();
    // Panelの描画領域全体のサイズをDimension型で取得する.
    Dimension d = this.getSize();
    // Panelの描画領域全体を白で塗りつぶす.
    g.setColor(Color.WHITE);
    g.fillRect(0, 0, (int)d.getWidth(), (int)d.getHeight());

    // 画像のサイズを取得する.
    int imgw = img.getWidth(this);
    int imgh = img.getHeight(this);
    // 表示座標が画像の中心となるように各軸に対して、画像サイズの半分
    // を減じた座標から表示する。なおImage型インスタンスimgは
    // コンストラクタで画像取得を行うものとする.
    g.drawImage(img, this.lastX-imgw/2, this.lastY-imgh/2, this);
}
```

3-2	問題設定	<p>課題3-1をmouseMoved()内にて実装した場合、画像がたまにチラつく（点滅する）ことがある。これは白背景で塗りつぶし→画像の描画のタイムラグによって発生する現象である。ダブルバッファリングを用いることで本現象を解決できるが、使用するためには描画処理は全て</p> <pre>public void paintComponent(Graphics g){}</pre> <p>内に記述されなければならない。課題 3-1 を修正し、ダブルバッファリングを機能させ、チラつきを改善せよ。</p>
-----	------	---

課題 3-2：解答例（KadaiPanel3 内の必要箇所抜粋）

```
// 表示させたい座標をフィールドとして保持する
private int lastX = -1, lastY = -1;

@Override
public void mouseMoved(MouseEvent e) {
    // マウスイベントの際には座標の取得のみを行う。
    this.lastX = e.getX();
    this.lastY = e.getY();
    // 再描画を強制する。
    this.repaint();
}

@Override
// ダブルバッファリングを有効にするため描画処理はpaintComponent()で行う。
public void paintComponent(Graphics g){
    // Panelの描画領域全体のサイズをDimension型で取得する。
    Dimension d = this.getSize();
    // Panelの描画領域全体を白で塗りつぶす。
    g.setColor(Color.WHITE);
    g.fillRect(0, 0, (int)d.getWidth(), (int)d.getHeight());

    // 画像のサイズを取得する。
    int imgw = img.getWidth(this);
    int imgh = img.getHeight(this);
    // 表示座標が画像の中心となるように各軸に対して、画像サイズの半分
    // を減じた座標から表示する。なおImage型インスタンスimgは
    // コンストラクタで画像取得を行うものとする。
    g.drawImage(img, this.lastX-imgw/2, this.lastY-imgh/2, this);
}
```

	問題設定	<p>(発展問題：採点対象外)</p> <p>課題 3-2 をベースに、表示させる図を動的に変更できるように拡張したい。まず、フレーム内上部に現在開発している JPanel を配置し、下部に 3 つのラジオボタンを配置する (ボタンのテキストはそれぞれ●, ☆, 画)。表示される図はラジオボタンでチェックされているものが表示される。</p> <p>●の場合は、単純な塗りつぶし円をマウスカーソル中心で描画する。</p> <p>☆の場合は、課題 1 で開発した☆をマウスカーソル中心で描画する。</p> <p>画の場合は、課題 3-2 同様に画像をマウスカーソル中心で描画する。</p> <p>なお、ダブルバッファリングを有効にすること。</p>
3-3	動作画面例	<p>図 4. 動作画面例</p>

課題 3-3：解答例 (MyFrame.java)

```
// import文は省略する.

// ラジオボタンと描画用のPanelを格納するためのFrameを開発する.
// ラジオボタンの切り替わりを取得したいのでActionListenerを実装する.
public class MyFrame extends JFrame implements ActionListener{
    private PaintPanel pp;           // 描画用の独自Panel (詳細は後述する)
    private JRadioButton radio1, radio2, radio3;
    public MyFrame(){
        // JFrameの初期設定関連
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        this.setSize(800, 600);
        this.setVisible(true);

        // 描画用の独自Panelを作成しコンテナに追加する.
        pp = new PaintPanel();
        this.getContentPane().add(BorderLayout.CENTER, pp);
        // ラジオボタン表示用のPanelを作成しコンテナに追加する.
        JPanel panel = new JPanel();
        this.getContentPane().add(BorderLayout.SOUTH, panel);

        // ラジオボタンを作成, グループ化, コンテナへ追加,
        // ActionListenerの登録を実施する.
        radio1 = new JRadioButton("●", true);
        radio2 = new JRadioButton("★");
        radio3 = new JRadioButton("画");
        ButtonGroup bg = new ButtonGroup();
        bg.add(radio1); bg.add(radio2); bg.add(radio3);
        panel.add(radio1); panel.add(radio2); panel.add(radio3);
        radio1.addActionListener(this);
        radio2.addActionListener(this);
        radio3.addActionListener(this);
    }

    @Override
    // ラジオボタンが押されたときには独自Panel ppのメソッドを呼び出し
    // 独自Panel側のdisplayフィールド値を変更する.
    public void actionPerformed(ActionEvent e) {
        if(e.getSource() == radio1){
            this.pp.setDisplay(PaintPanel.FLAG_ARC);
        }else if(e.getSource() == radio2){
            this.pp.setDisplay(PaintPanel.FLAG_STAR);
        }else if(e.getSource() == radio3){
            this.pp.setDisplay(PaintPanel.FLAG_IMAGE);
        }
    }
}
```


課題 3-3：解答例 (PaintPanel)

```
// Panel側の実装をPaintPanelで行う.
public class PaintPanel extends JPanel implements
    MouseListener, MouseMotionListener{

    public static final int FLAG_ARC = 1;
    public static final int FLAG_STAR = 2;
    public static final int FLAG_IMAGE = 3;

    // どのマークを表示するかを示すフラグ
    private int display = FLAG_ARC;
    // 画像のインスタンス
    private Image img;
    // 前回の座標(X,Y)をフィールドとして保持する
    private int lastX = -1, lastY = -1;

    public PaintPanel(){
        // MouseListenerとMouseMotionListenerの登録
        this.addMouseListener(this);
        this.addMouseMotionListener(this);

        // Imageの取得
        ImageIcon icon = new ImageIcon("small.png");
        img = icon.getImage();
    }

    // どのマークを表示するのかを外部から変更するためのメソッド
    public void setDisplay(int flag){
        this.display = flag;
    }

    // 今回使用しないメソッド群
    @Override
    public void mouseClicked(MouseEvent e) { }
    @Override
    public void mousePressed(MouseEvent e) { }
    @Override
    public void mouseReleased(MouseEvent e) { }
    @Override
    public void mouseEntered(MouseEvent e) { }
    @Override
    public void mouseDragged(MouseEvent e) { }

    // マウスが動いた時座標を記録して再描画を行う.
    @Override
    public void mouseMoved(MouseEvent e) {
        this.lastX = e.getX();
        this.lastY = e.getY();
        this.repaint();
    }
}
```

```
// マウスが範囲外に出た時残像が残らないように
// 表示座標を初期化して再描画を行う.
@Override
public void mouseExited(MouseEvent e) {
    this.lastX = -1;
    this.lastY = -1;
    this.repaint();
}

@Override
public void paintComponent(Graphics g){
    Dimension d = this.getSize();
    g.setColor(Color.WHITE);
    g.fillRect(0, 0, (int)d.getWidth(), (int)d.getHeight());
    g.setColor(Color.BLACK);
    if(this.lastX > -1 && this.lastY > -1){
        // 現在のフラグに応じて表示対応を変える.
        switch(this.display){
            case FLAG_ARC:
                // ●のときは標準のAPIで.
                g.fillOval(this.lastX-10, this.lastY-10, 20, 20);
                break;
            case FLAG_STAR:
                // ☆のときは課題1のメソッドを利用する.
                drawStar(g, this.lastX, this.lastY, 20);
                break;
            case FLAG_IMAGE:
                // 画像のときは課題3-2を参考に.
                int imgw = img.getWidth(this);
                int imgh = img.getHeight(this);
                g.drawImage(this.img, this.lastX-imgw/2,
                           this.lastY-imgh/2, this);
                break;
        }
    }
}

// コメントは課題1を参照する.
public void drawStar(Graphics g, int x, int y, int r){
    Dimension d = this.getSize();

    int[] xarr = new int[5];
    int[] yarr = new int[5];
    for(int i=0;i<5;i++){
        xarr[i] = x + (int)(r*Math.cos(
            Math.toRadians(90 + i*144)));
        yarr[i] = y - (int)(r*Math.sin(
            Math.toRadians(90 + i*144)));
    }
    g.drawPolygon(xarr, yarr, 5);
}
}
```