研究ノート

任意のインスタンスに描画機能を実装する "JustPad" クラスの開発

丸　山　裕　孝

［要旨］ アプリケーション Flash のオブジェクト指向開発言語 ActionScript3.0 にて、インタラクティブコンテンツ開発初心者の利用を想定した "JustPad" クラスを開発した。小規模の開発ながら、複数のスタッフにて発案から仕様の検討を行い、基本仕様の纏まる兆しが見えたあたりから開発属性を分担する実務的な作業形態をとった。丸山は開発内容についてはもとより､スタッフ育成と開発進行の均衡を取ることにも注力した。

教材開発を目的とした研究成果としては漸くスタートラインに立ったところではあるが、研究開発はもとよりスタッフの育成についても多くの示唆を得ることができたので、情報の整理と共有のために "JustPad" クラスの開発工程をここに掲載するものである。

［キーワード］ Flash、ActionScript3.0、オブジェクト指向、教材開発

1. はじめに

本来の研究目的は Flash によるインタラクティブコンテンツ開発に関する教育基盤整備であり、既に丸山が開発した幾らかのクラス群を調整及び整理して解説資料を作成することにあった。作業には手伝いを必要としたが、補助スタッフには Flash アプリケーションの使い方と ActionScript3.0 によるオブジェクト指向プログラミングの知識が必須である。丸山ゼミの学生であれば Flash の使い方を概ねマスターしているが、彼らを頼りとしても補助に値するまでの育成から始める必要があったため、多くを補助スタッフの学習時間に費やすことになった。

本開発は、ActionScript3.0 によるオブジェクト指向プログラミングの学習を終えたスタッフと共に本来の研究開発の先駆として着手したが、スタッフが何処までオブジェクト指向プログラミングを理解できているか計る上で手頃であった。また、この開発経験をもってすれば本来の研究目的を遂行するに十分な技能の習熟をスタッフにもたらすことが期待され、概ね良好な成果を得ることとなった。

2. スタッフ育成について

スタッフ育成は丸山ゼミナールより希望者を募り 5 名にて「ActionScript 虎の穴」と銘打って開始した。内 3 名が丸山の元にて学習を続け、1 名は通信にて独学を進めた。

学習は「標準 ActionScript 3.0 入門」【参考文献 (1)】の解説に合わせて進め、「Adobe Flash CS4 詳細！ ActionScript3.0 入門ノート」【参考文献 (2)】も参照しつつ、全員で ActionScript3.0 の要約ノートを作成して頭の中にある学習内容のインデックスを箇条書きにまとめる手法をとった。成果として上がった「ActionScript 3.0 Basical Reference 2011」（末尾掲示のウェブにて参照できます。）は、今後の ActionScript3.0 学習者の学習及び復習インデックスとして有効活用が期待できる。

本開発は、プログラミングの基礎学習を経てオブジェクト指向のコーディングを概ね習得した学生が、研究補助を行いつつ学習成果を実感し、更に習熟できるようにスタッフ育成の延長として考案した。本開発の補助スタッフとして、丸山の元で学習

した 3 名を「虎の穴のプログラマー」として採用した。（男性 2 名、女性 1 名）

3. 〝JustPad〞クラス開発の主旨と要件

立案に先立ち、丸山の開発クラス群に加えることを留意した上で、Flash の初心者でも簡単にインタラクティブなコンテンツを制作できるクラスの開発を検討した。

インタラクティブコンテンツ制作の開発初心者は、その開発成果の最初の利用者に他ならないので、開発者も利用者もインタラクションを直感できるものが良いだろうということになり、ホワイトボード、メモパッド、お絵描きキャンバス（以降、描画する対象オブジェクトを「描画キャンバス」と表現する）など、マウスを使って手描きのできるコンテンツを最小限度の ActionScript コードで実現できるようなクラスを想定した。丸山により技術的に開発可能であることが確認されたので、概ね以下のような要件を実装したクラスの開発を目標とした。

(1) 描画キャンバス用に作った任意のインスタンスをクラス生成時に参照するだけでそのインスタンス上に描画ができる。
(2) 描画キャンバスは描画色の変更ができる。
(3) 描画キャンバスは描画太さの変更ができる。
(4) 描画キャンバスは描画スタイルの変更ができる。
(5) 実装仕様は容易に理解でき、最小限度のプログラムコードで実装ができる。
(6) クラスファイルは 1 ファイルによるポータビリティーを維持する。

4. 開発工程

本稿では開発の工程順に従って記述する。全ての作業は丸山の監督のもと、補助スタイルと共に検討を行い、補助スタッフが進めたものである。

(1)〝DrawLineBasic1〞クラスの開発にて、Flash の描画機能を確認する。
1-1　クラスのファイルの体裁を作成する
1-2　Flash の線描画に関する仕様を確認する
1-3　マウスによるインタラクティブな描画機能を実装する

(2)〝DrawLineBasic2〞クラスの開発にて、描画機能の実装仕様を模索する。
2-1　実装仕様を検討する
2-2　キャンバス用インスタンスに描画機能を実装する

(3)〝DrawLineBasic3〞、〝DrawLineBasic4〞クラスの開発にて、描画仕様確立を検討する。（その 1）
3-1　星形インスタンスの上での描画を検討する
3-2　星形インスタンスに合わせて描画するための処方を検討する
3-3 描画動作の整合性について検討する

(4)〝DrawLineBasic5〞クラスの開発にて、描画仕様確立を検討する。（その 2）
4-1　星形インスタンスと同形体のマスク用インスタンスを生成する
4-2　描画動作の整合性について再検討する

(5)〝DrawLineBasic6〞クラスの開発にて、描画仕様確立を完了する。
5-1　描画仕様が判り易いものか再検討する
5-2　マウスイベントをキャッチする形態について再検討する

(6) 描画線の太さと色を変更できる機能を加える。
6-1　描画線の太さを変更できるようにする
6-2　描画線の太さに合わせてマウスカーソルの表示を変更できるようにする
6-3　コンテキストメニューによる描画線の太さを変更できるようにする
6-4　描画線の色を変更できるようにする

(7) 描画線のスタイルを変更できる機能を加える

7-1 ラインスタイルを拡張する可能性を準備する
7-2 ぼかし描画スタイルを加える
7-3 グラデーション描画スタイルを加える
7-4 マジック風描画スタイルを加える
7-5 機能を実装する

(8) 三班に別けた開発を結合して "JustPad" クラスを完成する。

＜開発環境＞

オペレーションシステム：MacOS 10.6/10.7
開発アプリケーション：Adobe Flash CS5/CS5.5
推奨ランタイム環境：Adobe Flash Player 10.2 以上
表示環境：最新のウェブブラウザ（2010.12.8 現在）

＜留意事項＞

Flash の開発言語 ActionScript3.0 はオブジェクト指向言語であり、実現しなかった JavaScript2.0 の傍系にあたる。更に JavaScript（当時はネットスケープコミュニケーションズ）は JAVA（当時はサン･マイクロシステムズ）と関係もあったためか、ActionScript3.0 のクラスコーディングは JAVA 風であり、ソースは JavaScript（参考 ECMAScript）の風体である。

また､Flash にある「シンボル」と「インスタンス」は Flash で視覚的にクラスとオブジェクトを取り扱うために付けられた用語である。文中では、ディスプレイクラスを継承するものを「シンボル」や「インスタンス」と表記するが､「シンボル」はクラス､「インスタンス」はオブジェクトと読み替えて理解いただいても概ね同じ意味になる。同様に「プロパティ」は変数､「メソッド」は関数､「パブリッシュ」はコンパイル＆リンケージである。

5. 開発

プログラムの詳細についての理解を全ての読者に要求することはできないので、ここではスタッフに対して作業概念を説明したのと同様に、開発仕様を実現するための「目的と処方」について述べることに注力する。

(1)〝DrawLineBasic1〞クラスの開発にて、Flash の描画機能を確認する。

Sprite クラスの graphics プロパティを介して、Graphics クラスのオブジェクトにアクセスして線描画を行う。

＜要件＞
1-1 クラスのファイルの体裁を作成する
1-2 Flash の線描画に関する仕様を確認する
1-3 マウスによるインタラクティブな描画機能を実装する

＜要件 1-1 ＞

以下は ActionScript3.0 におけるクラスファイル記述の基本構成である。

```
package {
    import flash.display.Sprite;
    import flash.events.MouseEvent;

    public class DrawLineBasic1 extends Sprite
{
        // 定義
    }
}
```

package はクラスファイルを記述することを意味している。

import で始まる行は、Flash に用意された機能をクラスに取り込むことを宣言している。

public から始まる行が "DrawLineBasic1" クラスの実体となる。Sprite クラスを継承しており、生成したインスタンス（実体化した自身）を描画キャンバスとする仕様になっている。

クラスファイル内「// 定義」の部分に、マウスのボタンを押したままドラッグするイベントに合わせて、自身に描画を行うためのプログラムコー

ドを書き込む。描画処理は、インスタンス内の graphics プロパティを介して Graphics クラスのオブジェクトにアクセスし、その機能を利用する。処理過程は、概ね以下のように行われる。

＜要件 1-2 ＞

まず、描画線の太さと色を決める。graphics プロパティの lineStyle メソッドに、パラメータに太さ 5（pixel）と緑色（16 進数）を入れて呼び出す。（「this.」というのは「自分自身の」という意味）

```
this.graphics.lineStyle(5, 0x00aa00);
```

＜要件 1-3 ＞

あらかじめ、マウスボタンが押されると同じオブジェクト内の startDraw メソッドが呼び出されるように、マウスボタンを押したイベントをキャッチする機能を実装しておく。

```
this.addEventListener(MouseEvent.MOUSE_DOWN, this.startDraw);
```

＜要件 1-2 ＞

呼び出された startDraw メソッド内で、描画位置をマウスの位置に移動する。（moveTo メソッド）

```
this.graphics.moveTo(this.mouseX, this.mouseY);
```

マウスボタンを押したまま移動すると、別途実装したマウス移動イベントをキャッチして移動した位置まで線を描画する。（lineTo メソッド）

```
this.graphics.lineTo(this.mouseX, this.mouseY);
```

以上のように行われる。

尚、Flash ムービー自身にマウスイベントをキャッチする機能を実装した場合は、描画がされていない限りはイベントをキャッチする実体が存在しないため、全く描画が始まらない。イベントを実装するオブジェクトの選択に注意が必要である。

"DrawLineBasic1" クラスでは、あらかじめ Flash ムービー全面に色を描画して透明化し、イベントをキャッチできるようにしてある。

Flash にて新規ファイルを作成し、フレームのアクションに以下のようにコードを書き込めば、パブリッシュしたムービー上で自在に線を引くことができる。"DrawLineBasic1" クラスは Sprite クラスを継承しているので、"DrawLineBasic1" から生成された canvas オブジェクトは、Flash 上のインスタンスとなる。

```
var canvas:DrawLineBasic1 = new DrawLineBasic1(this);
addChild(canvas);
```

【図：成果物（1）】（末尾掲示のウェブにて参照）

（2）〝DrawLineBasic2〞クラスの開発にて、描画機能の実装仕様を模索する。

描画キャンバスになる実装対象オブジェクトの種類と状態を検討し、実装方法も検証する。

＜要件＞

2-1　実装仕様を検討する

2-2　キャンバス用インスタンスに描画機能を実装する

＜要件 2-1 ＞

実装仕様につき、以下 2 項目の検討を行った。

2-1-a Flash ムービーの土台である「stage」とインスタンスの両方を描画キャンバスとするか。
2-1-b 描画キャンバスに割り当てるのは、シンボルとインスタンスのどちらを対象とするか。

・2-1-a

前項の "DrawLineBasic1" クラスは「stage」を描画キャンバスにした状態である。開発目標では stage 上の任意インスタンスを描画キャンバスの対象としていたが、開発の自由度を鑑み、stage 自体を描画キャンバスとする可能性を模索した。残念ながら「stage」と任意インスタンスの成因が異なることから、両方を描画キャンバスとするためには、ソースが複雑になること、それと利用者に「stage」の理解を求める必要が発生する。本開発は初心者を対象と考えているので、理解が容易であることを優先してインスタンスだけを対象とすることにした。

・2-1-b

Flash にはシンボルに本開発クラスを割り当てて（クラスの継承と同じ）、クラスの機能を実装する方法がある。（Flash ファイル自体にもクラスを割り当てる方法もある。2-1-a の場合と同様に、利用者にこれら機能の理解を求める必要が発生するため、2-1-b についても、理解が容易であることを優先してインスタンスだけを対象とすることにした。

＜要件 2-2 ＞

以上の作業に平行して、実装仕様に基づいた "DrawLineBasic2" クラスを作成した。

以下、基本体裁である。

```
package {
    import flash.events.MouseEvent;

    public class DrawLineBasic2 {
        // コンストラクタ
        public function
DrawLineBasic2(obj:Object):void {
            // 定義
        }
    }
}
```

まず、クラス自身に描画しなくなったので Sprite クラスを継承する必要はなくなった。よって、import も flash.display.Sprite クラスの記述を削除した。

コンストラクタのパラメーター「obj:Object」として、描画キャンバスとなるインスタンスを指定できるようになっている。（「obj:Object」の「:Object」はデータ型の記述である。実際はパラメーター「obj」が引き渡されると理解して良い）Flash ムービー開発者は、以下のようにプログラムコードを記述することになるので、コーディング意思が明示的になる。

new DrawLineBasic2(this.drawBoard)

（「this.drawBoard」の「drawBoard」は描画キャンバスにする「stage（this.）」上にあるインスタンスの名前）

さらに、描画機能の実装コードは以下の一行で済み、初心者にも優しくなった。"DrawLineBasic2" クラスはディスプレイクラスを継承していないのでディスプレイオブジェクトではない、よって "DrawLineBasic2" から生成された canvas オブジェクトは、通常のオブジェクトとなる。よって、名前を「canvas」から「canvasManager」に変更した。

```
var canvasManager:DrawLineBasic2 = new
DrawLineBasic2(this.drawBoard);
```

渡されたパラメーター「obj」はコンストラクタの中でプロパティー「canvas」に代入される。

```
canvas = obj;
```

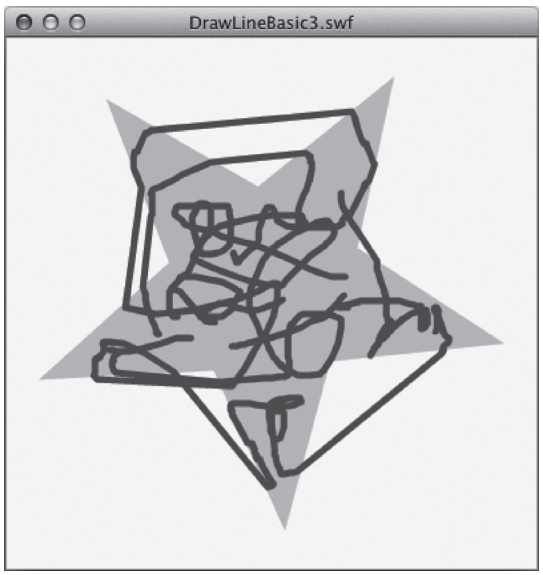
インタラクションと描画機能は前述の〝DrawLineBasic1〞クラスと同様であるが、描画 graphics プロパティやマウスのイベントは描画キャンバスインスタンスに由来するように変更してある。

例えば、描画線の太さと色を決める。graphics 内の lineStyle メソッドは以下のように描画キャン

バスインスタンスの参照を渡された「canvas.」で始まり、描画キャンバスインスタンスに描画することになる。
canvas.graphics.lineStyle(5, 0x7777aa);

パブリッシュすると、任意の形体の描画キャンバスインスタンス（以降は「星形インスタンス」とする）上でマウスボタンを押したまま動かすと描画が始まり、マウスボタンを放すと描画が終わるムービーが出来上がる。星形インスタンス上だけでしかマウスイベントをキャッチしないので描画動作が奇妙ではあるが、当座の目的は達成した。

【図：成果物（2)】（末尾掲示のウェブにて参照）



現況にて確認できた問題点を以下に列記する。

issue1 星形インスタンスの下で描画が行われている（動作が判り易いように星形インスタンスは半透明に着彩してある）。

issue2 星形インスタンスの形にぴったり合わせた描画ができない。

issue3 星形インスタンス上に描画しているため、描画線自体も星形キャンバスの一部となっている。よって境界で描画が止まらない現象が起こる。

issue4 星形インスタンスから高速でマウスアウトすると境界内で線が途切れる。

issue5 星形インスタンスを外れた場所でのマウスアップに対応していないため、マウスボタンを押してなくても描画される現象が起きる。

(3)〝DrawLineBasic3〞、〝DrawLineBasic4〞クラスの開発にて、描画仕様確立を検討する。（その 1）

Flash ムービー開発者の利用を想定して、仕様確立のために試行錯誤を行った。

＜要件＞

3-1　星形インスタンスの上での描画を検討する

3-2　星形インスタンスに合わせて描画するための処方を検討する

3-3　描画動作の整合性についての検討する

＜要件 3-1 ＞

既に描画（星形）のある星形インスタンスの graphics プロパティで描画を始めると星形の下に描画されてしまうという動作については、丸山も始めて知った。解決策として、単純に別途インスタンスを生成して上に乗せようということで検討を行った。

先ずは、"DrawLineBasic1" クラス同様に Sprite クラスの継承を復活させて星形インスタンスの上に親子関係で紐付けて乗せて、クラス自身に描画する形を取ることを検討した。作業は成功したが、他の問題点の解決を鑑みると、不整合が大量に発生する可能性が予想できたので決定を保留した。

次に、Sprite クラスの継承はせず、開発クラス内で別途インスタンスを生成してコントロールする方法を検討した。クラス自身がインスタンスとしての縛りにとらわれることもなく、複数のインスタンスを生成する事が可能となり、他の機能拡張も容易になるだろうと予想できる。よって、こちらを描画仕様の基本とした。

サッカーチームの監督が自ら選手としてプレーするのと、監督に徹してチームをコントロールする場合の違いを想像すればよいだろう。わざわざ、運用

を複雑にする必要もないので開発クラスには監督に徹してもらうこととした。

以下、〝DrawLineBasic3〞クラスの冒頭部分になる。

```
package {
    import flash.display.Shape;
    import flash.events.MouseEvent;

    public class DrawLineBasic3 {
        private var mf:Boolean = false;
        private var canvas:Object;
        private var gf:Shape;

        // コンストラクタ
        public function
DrawLineBasic3(obj:Object):void {
            canvas = obj;
            gf = new Shape();
            canvas.addChild(gf);

            // 他定義
        }
        // 他メソッド定義
    }
}
```

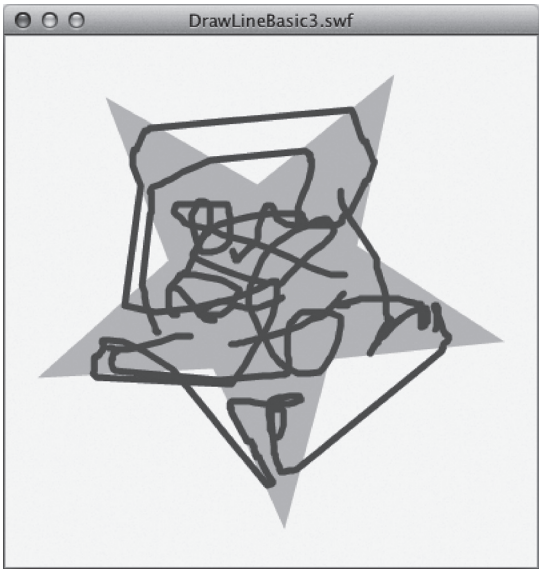
別途描画用のインスタンスを生成するため、import で flash.display.Shape を加えている。（ここでは、描画だけできれば良いので Shape を利用する。）

次の一行目で描画用インスタンス「gf」を生成し、二行目で星形インスタンスに親子関係で紐付けるという処理になる。

```
gf = new Shape();
canvas.addChild(gf);
```

パブリッシュすると、星形インスタンスの上に描画されることが確認できる。

【図：成果物（3)】（末尾掲示のウェブにて参照）



成果物（2）で確認された問題点の内、「issue1」が解決されたので仕様も決定とし、要件 3-1 も充足された。

＜要件 3-2 ＞

同じく「issue2」は星形インスタンスの形に由来する。

最初は、マウスがインスタンスから外へ出たときのイベントをキャッチして描画を止める方法をとってみたが、親子関係なので描画線自体が星形インスタンスの一部となり、「issue3」や「issue4」と同じ現象が起きるため「issue2」は解決されない。

次にゲームコンンテンツなどで使う当たり判定「hitTestPoint」なども利用してみたが、状況は変わらなかった。

また、長くなるのでここでは詳しくは記載しないが、後述する別途マウスイベントをキャッチするための星形インスタンスにて同様の判定を行うことにより概ね正常の動作を得るに至ったが、描画線に面積があるために境界線が凸凹になって星形インスタンスの形体を損なってしまった。よって、そのアプローチも却下した。

試行錯誤の末、良好なテスト結果を得たのがマ

スク機能の利用であった。これによって、「issue2」と「issue3」、「issue4」の解決が見込まれた。平行して「issue5」の解決に付いても他の問題との複合的な解決を考え、マウスが星形インスタンスの外へ出たマウスアウトイベントも実装してみた。

以下、"DrawLineBasic4" クラスによる検証になる。

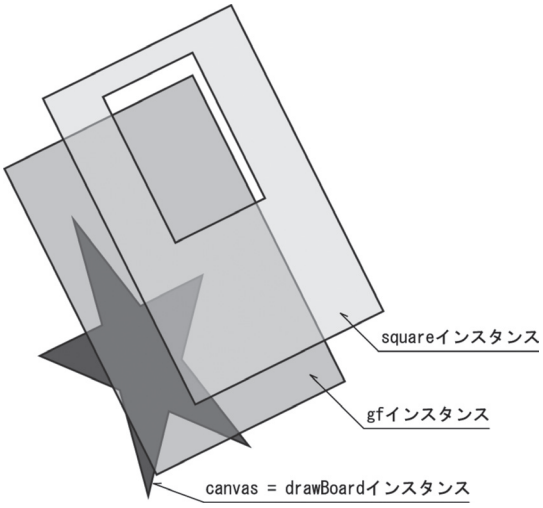
```
package {
    （略）
    public class DrawLineBasic4 {
        （略）
        // コンストラクタ
        public function
DrawLineBasic4(obj:Object):void {
            （略）
            square = new Shape( );
    square.graphics.beginFill(0xFF0000);
            square.graphics.drawRect(0, 0,
canvas.width-100, canvas.height-100);
            square.graphics.endFill();
            canvas.addChild(square);

            canvas.addChild(gf);
            gf.mask = square;
            （略）
        }
        （略）
    }
}
```

コンストラクタ内でマスク用インスタンス「square」を生成して、その graphics プロパティに赤色の四角を描画する。描画用インスタンス「gf」と同様に星形インスタンスに親子関係で紐付けて、「gf」のマスクに割り当てた。（マスクになると透明になるので「square」の赤色は見えなくなる）以下、該当コードである。

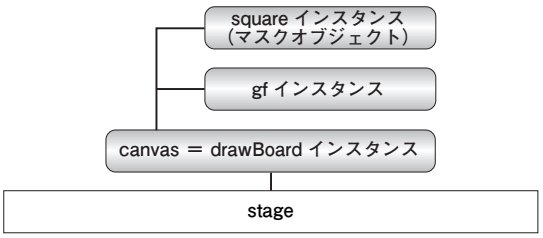
```
gf.mask = square;
```

【図：インスタンス構成の概念図】



注）square インスタンスはマスク化した状態であるため、背景に図が切り抜かれた表現としてある。

【図：インスタンス構成図】



【図：成果物（4）】（末尾掲示のウェブにて参照）

パブリッシュすると、【図：成果物（4）】のように四角い形に合わせた描画ができるようになる。星形のマスクを割り当てることができれば、「issue2」と「issue3」、「issue4」の解決処方が見えてきた。

また、マウスアウトイベントの実装により、星形インスタンスから外れると描画が途切れるため、「issue5」に付いて少し改善が見られた。

描画仕様の確立はこれからであるが、一応の検討成果はあったので、要件 3-2 と 3-3 の充足とした。
概ね描画仕様に関する仕様が固まって来たので、この後、開発属性を三班に別けて開発を進めることになった。

一班、描画仕様の確立する

二班、描画線の太さと色を変更できるようにする

三班、描画線のスタイルを変更できるようにする

本文は、引き続き一班の開発を辿り、二班と三班の成果は後述の完成版にて紹介する。

（4）〝DrawLineBasic5〞クラスの開発にて、描画仕様確立を検討する。（その 2）

Flash ムービー開発者の利用を想定して、仕様確立を進めた。

＜要件＞

4-1　星形インスタンスと同形体のマスク用インスタンスを生成する

4-2　描画動作の整合性について再検討する

＜要件 4-1 ＞

マスクの割当てが成功したので、星形インスタンスと同形体のマスク用インスタンスを用意することが必要になった。

調査の結果、参照している星形インスタンスのコンストラクタを呼び出すだけでインスタンス生成ができることを確認した。別途マスク用に「maskObj」プロパティを準備して星形インスタンスを生成し前述の「square」と同様にマスクに割り当てた。以下、該当コードである。（一行目がコンストラクタ呼び出しになる）

```
maskObj = new obj.constructor( );
canvas.addChild(gf);
canvas.addChild(maskObj);
gf.mask = maskObj;
```

これにより「issue2」が解決された。

＜要件 4-2 ＞

星形インスタンスをマウスイベントの受け皿としているために問題が発生すると考え、星形インスタンスより大きい面積、マウスイベントをキャッチするムービー全面サイズのインスタンスを生成することとした。

前述で利用されなくなった「square」を透明（アルファチャンネルを 0%）にして「stage」のサイズに拡大、星形インスタンス内オブジェクトレイヤーの最前面に配置される様に一番最後に親子関係で紐付けた。（以下、該当コード）

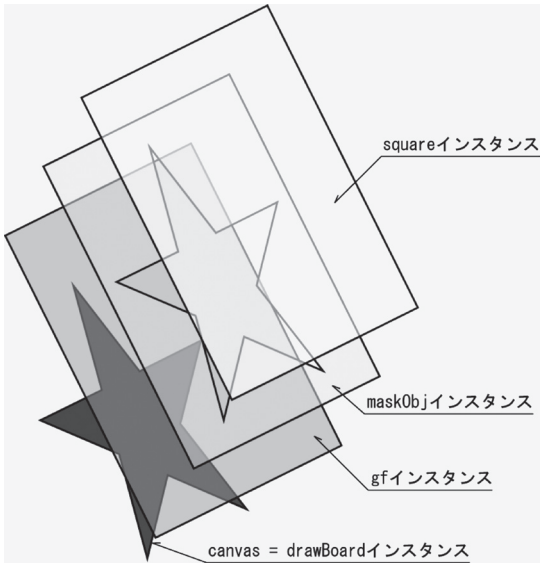
```
square.graphics.beginFill(0xFF0000,0);
square.graphics.drawRect(0,0,canvas.stage.
stageWidth,canvas.stage.stageHeight);
square.graphics.endFill( );
canvas.addChild(square);
```

（アルファチャンネルの 0 は、一行目の最後の 0 になる）

そしてマウスのイベントは全て「square」に対して実装した。（以下、該当コード）

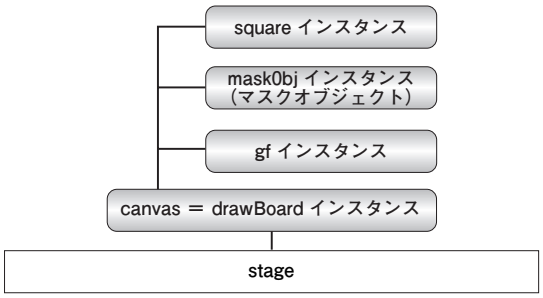
```
square.addEventListener(MouseEvent.MOUSE_DOWN,startDraw);
square.addEventListener(MouseEvent.MOUSE_UP,stopDraw);
square.addEventListener(MouseEvent.MOUSE_MOVE, this.lineDraw);
```

【図：インスタンス構成の概念図】



注）mask0bj インスタンスはマスク化した状態であるため、背景に図が切り抜かれた表現としてある。

【図：インスタンス構成の図】



【図：成果物（5)】（末尾掲示のウェブにて参照）

【ウェブにて成果物（f）を参照】

これで、「issue3」と「issue4」は解決された。

しかし、検討の必要な事が二点ある。

第一に現在はお絵描きパッドの外側から描画を始めることができ、且つ、通過して描画を進めることができてしまうが、ユーザーにとって好ましいものかどうか検討する必要がある。

第二に現在のムービーサイズ全面でマウスイベントをキャッチしている状態は、お絵描きパッドと他インスタンスの関係を考えていないので、他インスタンスと同居させる必要があった場合は干渉が予測される。

これらを次項の要件とする。

（5）〝DrawLineBasic6〞クラスの開発にて、描画仕様確立を完了する。

お絵描きパッドなどの Flash ムービーユーザーのために利用仕様の検討を行い、描画仕様の完成を目指した。

＜要件＞

5-1　描画仕様が判り易いものか再検討する

5-2　マウスイベントをキャッチする形態について再検討する

開発仕様の細かい詰めにおいて、本要件 5-1 と 5-2 は相互に且つ複合的に関係するものである。よって、順次解説すると冗長になるので、様々に検討をした結果だけを解説する。

まずは描画について、"DrawLineBasic5" クラスでは星形インスタンスの外側から描画を始める事ができる訳だが、実際のノート等をメタファーとしたお絵描きパッドなどの描画仕様としては不合理である。かといって、星形インスタンスからマウスが外れるとそれきり描画が止まってしまうというのも描画の柔軟性に欠けるので、星形インスタンスから描画が始まって再度星形インスタンス内へマウスが戻って来た場合の再描画は維持しようと判断した。

となると、他インスタンスとの干渉の可能性を排除する事も考えて、描画開始に関わるマウスイベントは星形インスタンスと同形体にしてマウスボタンが押されたイベントに特化して良いだろうとなった。

そしてマスク用インスタンスと同様に星形インスタンスのコンストラクタを呼び出して新たにインスタンスを生成し、プロパティ「touchObj」に代入して「square」と同様に星形インスタンスに親子関係で紐付けた。以下、プログラムコードの最後の行でアルファ値を 0 にして透明にした。

```
touchObj = new obj.constructor();
touchObj.name = "touchBoard";
canvas.addChild(touchObj);
touchObj.alpha = 0;
```

更に「square」と同様にマウスボタンが押されたイベントをキャッチする様に設定した。

```
touchObj.addEventListener(MouseEvent.MOUSE_DOWN, this.startDraw);
```

しかし、マウスボタンを放したイベントは「touchObj」でキャッチせずに、ムービー画面上の何処で放してもキャッチできる様に「touchObj.stage」と設定した。

```
touchObj.stage.addEventListener(MouseEvent.MOUSE_UP, this.upOnStage);
```

次に新規に、マウスが星形インスタンスの外に出た時のイベントを「touchObj」でキャッチする様に設定した。これにより外に出てからのマウスボタンの操作に対応できるようになった。

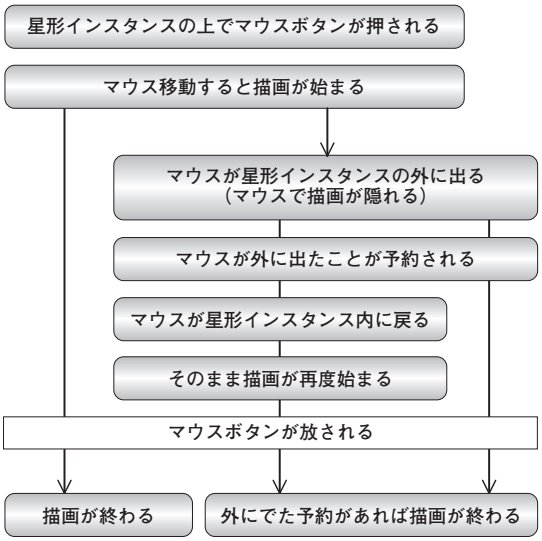
```
touchObj.addEventListener(MouseEvent.MOUSE_OUT, this.outCanvas);
```

そして前項まで利用していたマウス移動イベントを却下して、エンターフレームイベント（Flash ムービーが表示されている限り発生している時間を刻むようなイベント）にて描画の継続を行う事とした。「touchObj」でキャッチする様に設定してあるがマウスイベントとは関係が無くなっている。マウスイベントをキャッチするインスタンスが星形になったため、描画を途切れなく行うためにこの処方となった。描画動作が ON の場合は描画し続けることになる。

```
touchObj.addEventListener(Event.ENTER_FRAME, this.drawLine);
```

これらの仕様調整で決定した描画フローは、以下の通りである。

【図：描画動作に関するフローチャート】



【図：成果物（6)】（末尾掲示のウェブにて参照）

ここまで描画仕様が固まったところで、星形インスタンス上でマウスをクリックだけをしたときは何も描画されない事に違和感を感じる様になった。星形インスタンス上でクリックだけをしたときは移動がないので描画されないのは仕様上は当たり前だが、お絵描きするユーザーはそうは感じないだろう。

そこで、星形インスタンス上でクリックだけした時には微妙に移動した事として描画がされるように計らった。試行の結果、以下のように最小限度 0.15 ピクセルだけ移動した事にすると描画サイズの点が打てるようになった。

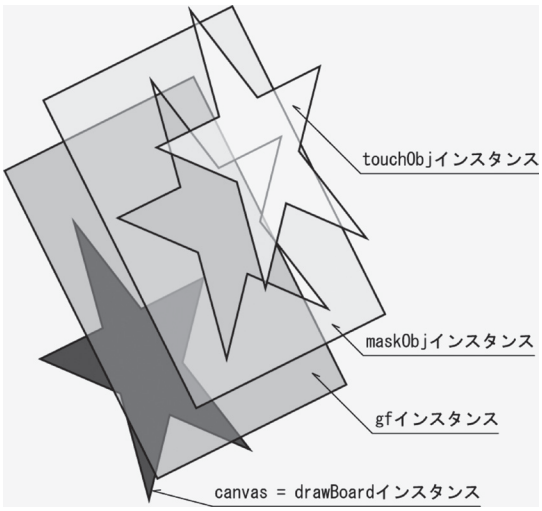
```
gf.graphics.lineTo(e.target.mouseX, e.target.mouseY+0.15);
```

ところが、そのためにバグが発生した。星形インスタンスの外でマウスボタンを押してから入って来てマウスボタンを放した場合も点を打ってしまうようになった。

直ぐに条件を加えてプログラムコードの順番を調整して対応した。

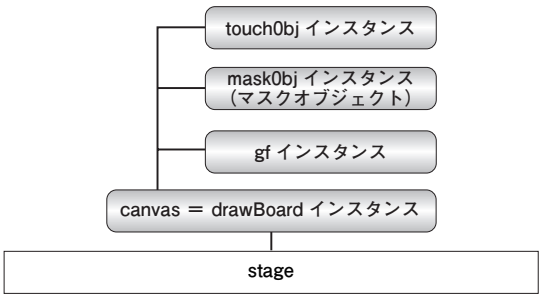
他、今のところ描画仕様に関するバグには遭遇していない。

【図：インスタンス構成の概念図】



注）mask0bj インスタンスはマスク化した状態であるため、背景に図が切り抜かれた表現としてある。

【図：インスタンス構成図】



以上、描画仕様が "DrawLineBasic6" クラスとしてまとまった。

(6) 描画線の太さと色を変更できる機能を加える。

三班に別けた開発の内、ユーザーインタフェースに関係する拡張仕様になる。お絵描きパッドなどのユーザーの描画操作に自由度を拡張できるが、Flash ムービー開発者には負荷が増えることになる。

＜要件＞

6-1　描画線の太さを変更できるようにする

6-2　描画線の太さに合わせてマウスカーソルの表示を変更できるようにする

6-3　コンテキストメニューによる描画線の太さを変更できるようにする

6-4 描画線の色を変更できるようにする

＜要件 6-1 と 6-2 ＞

特に線の太さを変更する事に関しては、Flash ムービー開発者には、選択用インスタンス制作やイベントリスナーの実装などに理解が必要となる。

以下、本クラス内での仕組みは比較的簡単で、線の太さを受け取るメソッドと設定するメソッドにて構成される。

線の太さを受け取る setLineSize メソッド内では、一行目でパラメーター「n」を「lineSize」に代入して、二行目で描画を様々に設定する setRegistLine メソッドを呼び出しているだけである。

```
public function setLineSize(n:int):void {
    lineSize = n;
    this.setRegistLine( );
}
```

setRegistLine メソッドには描画形態に関する管理機能を集約させてあるが、ここでは線の太さに関係する項目だけを紹介する。メソッド内二行目「gf」で始まる行は、何度か出て来た lineStyle メソッドを呼び出してパラメーター「lineSize」を渡しており（続くパラメーター「lineColor」は線の色）、線の太さが変更される。四行目にある「this.changeMouse();」は、マウスカーソルの代わりに描画線の太さと色を表示するためのメソッドで、ユーザビリティー向上のために用意した拡張機能である。続けて changeMouse メソッドを掲載する。尚、この機能は本クラスを参照する際に、星形インスタンスのパラメーターの後ろに「true」キーワードを付加すると実現するようになっている。（詳しくは「(8) 三班に別けた開発を結合して〝JustPad〞クラスを完成する」を参照）

```
private function setRegistLine( ):void {
    （略）
    gf.graphics.lineStyle(lineSize, lineColor);
    （略）
    this.changeMouse();
}
```

以下、マウスカーソルの代わりに描画線の太さと色を表示するためのメソッドである。マウスカーソルとこのインスタンスの表示非表示は描画状態の入り切り（トグル）で切り替わる。尚、このインスタンスは、星形マスクで切られないように一番上のレイヤーに星形インスタンスと親子関係で紐付けてある。

```
private function changeMouse( ):void {
    if(MouseC != null) {
        MouseC.graphics.clear( );
```

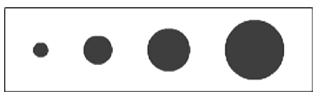
```
        MouseC.graphics.lineStyle(0, 0x000000, 0.5);
        MouseC.graphics.beginFill(lineColor);
        MouseC.graphics.drawCircle(0,0,lineSize/2);
        MouseC.graphics.endFill();
    }
}
```

機能を実装するためには、開発される Flash ムービー上の星形インスタンスに添えて、描画の太さ毎にボタンになるインスタンスを作り、太さ変更用インターフェースを用意する。そして、同一フレーム内に以下のコードを書き込んで、本クラスの canvasManager オブジェクトと連携をさせなくてはならない。

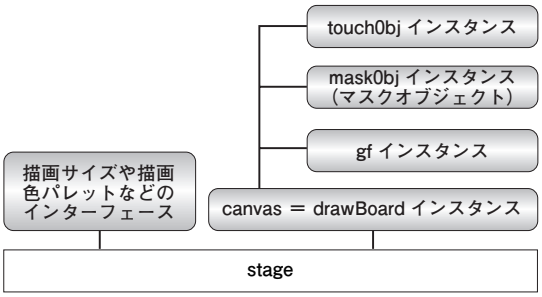
```
s6.addEventListener(MouseEvent.MOUSE_DOWN,function(e){canvasManager.setLineSize(6)});
s11.addEventListener(MouseEvent.MOUSE_DOWN,function(e){canvasManager.setLineSize(11)});
s16.addEventListener(MouseEvent.MOUSE_DOWN,function(e){canvasManager.setLineSize(16)});
s22.addEventListener(MouseEvent.MOUSE_DOWN,function(e){canvasManager.setLineSize(22)});
```

コードの内容は、各太さ毎のインスタンスにマウスが押されたイベントをキャッチする機能を実装して、前述の開発クラス内 setLineSize メソッドに太さの数値をパラメーターとして投げるというものである。テストでは 4 サイズを用意した。

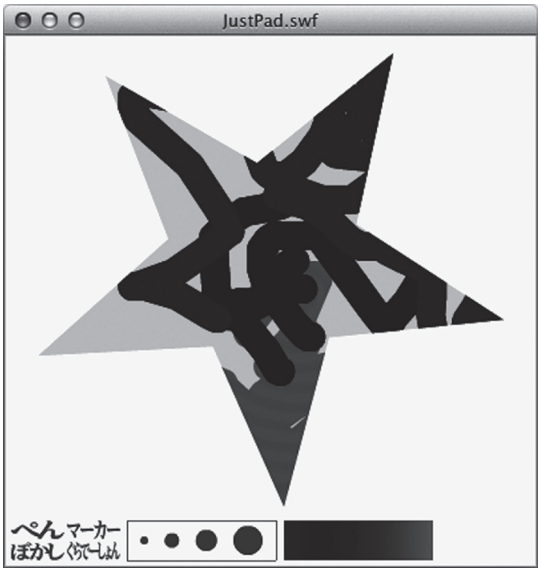
【図：描画サイズ変更用インターフェース（s6、s11、s16、s22 インスタンスのセット）】



【図：インスタンス構成図】



【図：成果物（7)】（末尾掲示のウェブにて参照）



【ウェブにて成果物（h）を参照】

＜要件 6-3 ＞

追加で、インターフェースを用意しなくても描画線の太さが変更できるように、コンテキストメニューによる太さ調節の機能も実装した。これを利用した場合、Flash ムービー開発者にはほとんど負担がかからない。以下、コンテキストメニュー機能をクラスに装備する宣言である。

```
import flash.ui.ContextMenu;
import flash.ui.ContextMenuItem;
import flash.events.ContextMenuEvent;
```

新規にコンテキストメニューの生成を行う。

```
menu_cm = new ContextMenu();
```

増設メニュー分のメニューアイテムを生成して、メニュー項目の設定を行う。以下、参考まで「sizeM1」メニュー項目の設定コードである。（「sizeM1」から「sizeM4」まで生成、以下 sizeM1 の設定の抜粋）

```
// メニューアイテムを作成
sizeM1 = new ContextMenuItem("");
// キャプション名（コンテキストメニューに表示される）
sizeM1.caption = "SIZE:1";
（略）
// 押されたときに呼び出される関数とパラメーター「2」を設定する（インターフェースを用意した場合と同じく、setLineSize メソッドを利用する）
sizeM1.addEventListener(ContextMenuEvent.MENU_ITEM_SELECT,function(e){setLineSize(2)});
```

生成したメニューアイテムは、新規コンテキストメニューの customItems の配列プロパティに代入する。

```
menu_cm.customItems = [sizeM1,sizeM2,sizeM3,sizeM4];
```

星形インスタンスの上で右クリックした時のコンテキストメニューに描画太さ変更のメニューが現れるようにキャンバスに新規コンテキストメニューを実装する。

```
canvas.contextMenu = menu_cm;
```

【同じく、成果物（7）を参照】

＜要件 6-4 ＞

任意のインスタンスをパレットとして引き渡し、そのインスタンス内の色を使って描画色を変更できる機能拡張も行った。以下、インスタンスを画像として取り込むための機能をクラスに装備する宣言である。

```
import flash.display.DisplayObject;
import flash.display.BitmapData;
```

インスタンスを画像として取り込むためのプロパティを用意する。

```
private var bmd:BitmapData;
```

以下、setPalet メソッドでは、渡されたパラメーター「plt」のインスタンスを画像にして、マウスボタンを放したイベントをキャッチできるように設定している。これにより、任意の画像やグラデーションをパレットとして指定することができる。

```
public function setPalet(plt:DisplayObject):void {
    bmd = new BitmapData(plt.width, plt.height, false, 0x000000);
    bmd.draw(plt);
    // パレット上でマウスを押した時のイベントを実装
    plt.addEventListener(MouseEvent.MOUSE_UP,this.setPalletColor);
}
```

以下、setPalletColor メソッドはマウスボタンを放したイベントで呼び出される。一行目はマウスのあった位置の色を拾って「lineColor」に代入し、二行目で描画太さと同様に setRegistLine メソッドを呼び出す。

```
public function setPalletColor(e:MouseEvent):void {
    lineColor = bmd.getPixel(e.target.mouseX, e.target.mouseY);
    this.setRegistLine();
}
```

機能を実装するためには、開発される Flash ムービー上の星形インスタンスに添えて、描画色を選択するためのインスタンスを作ってパレットインターフェースとする。そして、同一フレーム内に以下のコードを書き込んで、本クラスの canvasManager

オブジェクトと連携をさせなくてはならない。

コードの内容は、前述の開発クラス内 setPalet メソッドにパレットインターフェースをパラメーターにして投げるというものである。テストでは「palet」を用意した。

```
canvasManager.setPalet(this.palet);
```

【図：パレット用インターフェースのインスタンス】
（青から赤へのグラデーションで彩色された矩形）

【同じく、成果物（7）を参照】

(7) 描画線のスタイルを変更できる機能を加える

単なる線による描画だけでなく、ラインスタイルにバリエーションがあった方が楽しいものができると考えて検討した。結果、描画機能を利用して変化が加えられる事が判り、とりあえず、三種類を追加して今後の拡張の可能性として加えた。

＜要件＞

7-1　ラインスタイルを拡張する可能性を準備する
7-2　ぼかし描画スタイルを加える
7-3　グラデーション描画スタイルを加える
7-4　マジック風描画スタイルを加える
7-5　機能を実装する

＜要件 7-1 ＞

以下、描画プロパティに関する機能を可能な限り利用するため、クラスに機能を装備する宣言である。

```
// ぼかし描画スタイル用
import flash.filters.BlurFilter;
// グラデーション描画スタイル用
import flash.display.GradientType;
import flash.geom.Matrix;
import flash.display.SpreadMethod;
// マジック風描画スタイル用
import flash.display.CapsStyle;
import flash.display.JointStyle;
import flash.display.LineScaleMode;
```

ラインスタイルの変更に関しては太さの変更と同様に、Flash ムービー開発者に選択用インスタンス制作やイベントリスナーの実装などに理解が必要となる。

クラス内 setLineStyle メソッドではパラメーター「s」にスタイル名が引き渡され、一行目で「lineStyle」に代入して、二行目で描画を様々に設定する setRegistLine メソッドを呼び出しているだけである。

```
public function setLineStyle(s:int):void {
    lineStyle = s;
    this.setRegistLine();
}
```

引き続き、setRegistLine メソッド内で処理される各ラインスタイルの設定は次の通りである。

＜要件 7-2 ＞

「ぼかし」の処理は比較的シンプルである。一行目の lineStyle メソッドで通常の描画準備を行い、BlurFilter クラスのコンンストラクタにボケ味のプロパティを渡して、生成したボケオブジェクトを「gf」描画インスタンスの filters プロパティーに代入するだけである。

```
gf.graphics.lineStyle(lineSize, lineColor);
var myBlur:BlurFilter = new BlurFilter(8,5);
gf.filters = [myBlur];
```

現況では描画した全てに「ぼかし」がかかるが、基本仕様外なので動作確認までで良しとする。

＜要件 7-3 ＞

最も厄介なのはこの「グラデーション」で、今までの lineStyle メソッドではなく lineGradientStyle メソッドで指定する必要がある。以下、個別に簡単に解説を入れる。

```
// 通常の描画準備は行っておく。
gf.graphics.lineStyle(lineSize, lineColor);
// グラデーションの塗りを指定するキーワードで
```

「LINEAR」は線状、「RADIAL」は放射状である。
var type:String=GradientType.RADIAL;
// グラデーションで使用する色の配列指定である。現在は二色であるが複数指定ができる。（RGB16 進数カラー値）
var colors:Array=[lineColor,0x0000FF];
// 上記の配列の各色に割り当てる透明度（アルファ値 0 ～ 1）の指定で、現在の lineAlpha には 0.5（50%）が代入されているので、一色目と二色目は同じく透明度が 50% になっている。
var alphas:Array=[lineAlpha, 0.5];
// 同じく上記の配列の色分布比率の配列で、0 ～ 255 の範囲の値を指定する。（詳細には触れない）
var ratios:Array=[50,255];
//Matrix クラスを利用してグラフィック変換行う。（基本的な設定に留める）
var mtrx:Matrix=new Matrix();
// グラフィック変換の設定は以下の通りである。
//「15, 15」はグラデーションの幅を指定、「45」グラデーションの角度（ここでは放射状設定のため関係ない）、後はグラデーションの中心点の設定になる。
mtrx.createGradientBox(15, 15, 45, (canvas.width-canvas.x)/2, (canvas.height-canvas.y)/2);
// グラデーションの塗りの繰り返し方を指定する定数値で、「REFLECT」は反転させながら繰り返す指定である。
var spread:String=SpreadMethod.REFLECT;
// 以上の設定を lineGradientStyle メソッドにパラメーター群として渡すと、グラデーションによる描画が可能となる。
gf.graphics.lineGradientStyle(type,colors,alphas,ratios,mtrx,spread);

＜要件 7-4 ＞

これまで利用して来た lineStyle メソッドは太さと色だけでなく、多くのパラメーターを渡す事ができるので、それらを使ってマジック風のラインスタイルを作成した。

gf.graphics.lineStyle(lineSize, lineColor, lineAlpha, true, LineScaleMode.NORMAL, CapsStyle.NONE, JointStyle.BEVEL);

以下、コード内の拡張されたパラメーターの解説になる。

「lineAlpha」は透明度（アルファ値 0 ～ 1）の指定で、現在の lineAlpha には 0.5 が代入されているので描画の透明度は 50% になる。

「true」はヒンティングと呼ばれる指定で、曲線の描画を微調整して滑らかにするかどうかを「true」と「false」で入り切りするものである。

「LineScaleMode.NORMAL」は今回は機能していないが、パラメータとして記載の必要がある。

「CapsStyle.NONE」は描画線のキャップスタイル（線の途切れ方）の指定する定数値で「NONE」は線の長さで途切れる。他に「SQUARE」と「ROUND」がある。

「JointStyle.BEVEL」は描画線の結合スタイル（折れ線の角の形）を指定する定数値で「BEVEL」は折れ線に角が出る。他に「MITER」と「ROUND」がある。

＜要件 7-5 ＞

ここまでの機能を実装するためには、開発される Flash ムービー上の星形インスタンスに添えて、描画スタイル毎にボタンになるインスタンスを作り、スタイル変更用インターフェースを用意する。そして、同一フレーム内に以下のコードを書き込んで、本クラスの canvasManager オブジェクトと連携をさせなくてはならない。

```
penBtn.addEventListener(MouseEvent.MOUSE_DOWN,function(e){canvasManager.setLineStyle("PEN")});
markerBtn.addEventListener(MouseEvent.MOUSE_DOWN,function(e){canvasManager.setLineStyle("MARKER")});
blurBtn.addEventListener(MouseEvent.MOUSE_DOWN,function(e){canvasManager.setLineStyle("BLUR")});
```

gradationBtn.addEventListener(MouseEvent.MOUSE_DOWN,function(e){canvasManager.setLineStyle("GRADATION")});

コードの内容は、各スタイル毎のインスタンスにマウスが押されたイベントをキャッチする機能を実装して、前述の開発クラス内 setLineStyle メソッドにキーワードをパラメーターとして投げるというものである。テストでは penBtn、markerBtn、blurBtn、gradationBtn の各インスタンスに「PEN」「MARKER」「BLUR」「GRADATION」のキーワードを用意した。

【図：描画スタイル変更用インターフェース（penBtn、markerBtn、blurBtn、gradationBtn インスタンスのセット）】

【図：成果物（7）】（末尾掲示のウェブにて参照）

（8）三班に別けた開発を結合して〝JustPad〞クラスを完成する。

＜要件＞

(6)(7) の機能を "DrawLineBasic6" クラスに加えて、「JustPad」と正式名称とし開発を修了した。

以下、インスタンス生成コードになる。

var canvasManager:JustPad = new JustPad(this.drawBoard, true);

尚、星形インスタンスのパラメータの後に、マウスの表示に関する MousePenMode パラメータ（真偽値）を追加した。描画時に線の太さと色をマウスの代わりに表示する機能の入り切りを指定するパラメータである。（「（6）描画線の太さと色を変更できる機能を加える」を参照）

以下、クラスファイル内のコンストラクタでは初期値が「false」になっており、パラメータの指定がなければ機能は反映されない。

```
public function JustPad(obj:Object, MousePenMode:Boolean=false):void {
    // 処理
}
```

【図：成果物（7）】（末尾掲示のウェブにて参照）

以上、"JustPad" クラスが完成した。

6. おわりに

＜開発作業について＞

実際の開発作業はこのようにスムーズに進んだ訳ではない。時間を隔ててコツコツと進行したこともあり作業が前後する事もあった。また、クラスファイル内のコメントに留まり、利用に関するドキュメンテーションにまで手が届かなかったのは、スケジューリング上の反省点である。

参加スタッフの技能習得には目を見張るものがあり、実務教育として考えれば良い成果であった。このように、教育を進めながら研究開発を行うという手法が取れるのであれば理想的であろうが、5 人程度までの少人数のクラス編制が必要であり、手間と時間もかかるので教育以外の業務が嵩む中では現実的とは考えられない。

＜成果物について＞

成果物の基本仕様練度は十分であると考えるので、インタラクティブコンテンツ開発初心者の学習教材としては問題ないと判断する。しかし、インタラクティブコンテンツ開発の実運用に耐える品質には仕上がっていない。本クラスを十分に利用して出来上がったお絵描きパッドなどがウェブコンテンツとしてユーザーの利用に値するかというと不十分極まりないと予想される。

特にマウスカーソルが矢印のままであるため、子供向けのコンテンツなどではユーザーへの親和性に不足を感じることになるであろう。また、描画スタ

イル毎に描画オブジェクトを生成した方が合理的とも考えている。製品レベルに押し上げるための細やかな開発と仕様の再調整も必要であろう。

引き続きブラッシュアップを進め、テスト Flash ムービーと Flash ファイルとクラスファイルをウェブ上に公開する所存である。

成果物（ウェブページリンクより参照）

http://metarial.com/mir/pub/justpad/index.html

成果物（1）DrawLineBasic1 クラスによる
成果物（2）DrawLineBasic2 クラスによる
成果物（3）DrawLineBasic3 クラスによる
成果物（4）DrawLineBasic4 クラスによる
成果物（5）DrawLineBasic5 クラスによる
成果物（6）DrawLineBasic6 クラスによる
成果物（7）JustPad クラスによる
成果物（8）本原稿を執筆後に開発した JusPad クラス開発仕様完成版による
成果（8）のソース（以下 2 点の圧縮ファイル）
1　完成版を実装した Flash ファイル
2　完成版の ActionScript クラスファイル

学習メモ「ActionScript 3.0 Basical Reference 2011」

参考文献

（1）標準 ActionScript 3.0 入門　吉岡 梅（著）　ソフトバンククリエイティブ
（2）Adobe Flash CS4 詳細！ ActionScript3.0 入門ノート [完全改訂版]　大重 美幸（著）　ソーテック社
（3）Adobe Flash CS4 詳細 !ActionScript3.0 入門ノート　大重 美幸 (著)　ソーテック社
（4）基本からしっかりわかる ActionScript 3.0　森 巧尚 (著)　毎日コミュニケーションズ
（5）Flash ヘルプ（Adobe Flash CS5 に実装されているヘルプコンテンツ）

The Development of a Class 'JustPad' , to Provide Drawing Functions for Instances in ActionScript 3.
by Hirotaka MARUYAMA