

本ドキュメントは、

アメリカ O'Reilly 社の web サイト <http://www.oreilly.com/catalog/0596006527/index.html>にある

“Essential ActionScript 2.0”のサンプルチャプター

(<http://www.oreilly.com/catalog/0596006527/chapter/ch02.pdf>)を

ヒム・カンパニー 永井勝則が独自に日本語に翻訳したものです。

“Essential ActionScript 2.0”の著作権は、著作者である Colin Moock 氏にあり、翻訳/出版権は O'Reilly 社(及びオライリー・ジャパン社)にあるものと考えられますが、上記のように、“サンプルチャプター”として、PDF 版で“Essential ActionScript 2.0”の一部が公開されていますので、それに基づき、チャプター 2 部分のみ、翻訳を公開します。

なお、本翻訳文は、翻訳作業の初期段階であり、見直しなどは行っていません。

また内容に関し、当方ではいっさいお答えできません。

さらに、チャプター 2 という非常に中途半端な部分での翻訳文ですので、一般的な参考書代わりになるとは到底思えません。

さらに詳しく ActionScript2.0 を学びたいと思われる方は、“Essential ActionScript 2.0”を購入されるのがよろしいかと思えます。

当該関係各社より、当文書の削除を求められた場合は、早急に削除します。

ヒム・カンパニー

永井 勝則

<http://www.himco.jp/>

オブジェクト指向の ActionScript

皮肉なことに、オブジェクト指向プログラミング(OOP)をこれから始めようとする Flash ユーザーは、その形式的な名前を知らないまま、オブジェクト指向の多くのコンセプトに慣れ親しんでいる場合があります。本チャプターでは、新人プログラマーが、キーとなる OOP のコンセプトに精通できるよう、用語を分かりやすく解説します。それはまた Flash の開発へ初めて踏み出す熟練プログラマーにとっても、ActionScript に関する解説にもなります。

手続き型プログラミングとオブジェクト指向プログラミング

これまでのプログラミングは、プロシージャにまとめられたさまざまな命令によって構成されています。プロシージャは、別のさらに大きなプログラムに関する知識や関係を持つことなく、特定の命令を実行します。例えば、プロシージャは計算したり結果を返したりします。Flash の手続き型プログラムでは、関数内に繰り返す命令を持ち、変数内にデータを保持します。プログラムは、通常、入力処理と出力を目的として、関数を実行し変数値を変更することで動作します。手続き型プログラミングは、特定のアプリケーションでは実用的ですが、アプリケーションが巨大化、複雑化したりプロシージャ（とそれを扱うプログラマー）間の連携が膨大になればなるほど、手続き型プログラミングは扱いにくいものになってしまいます。手続き型プログラミングはメンテナンスやデバッグ、アップグレードすることが難しいのです。

オブジェクト指向プログラミング(OOP)は、通常大きな手続き型プログラムで発生する開発やメンテナンスの問題解決を意図し、プログラミングに対し異なるアプローチを行います。OOP は、アプリケーションを独立した部分、つまり連携し合うモジュールに分割することで、複雑なアプリケーションを管理しやすいように設計されたものです。OOP によって、抽象的な概念や現実世界の実物を、それに類似するプログラムの部分(OOP でいう“オブジェクト”)に置き換えることができます。また複数のアプリケーションを作成管理できるように設計されていて、ユーザーインターフェイスを必要とするときもあります。例えば、シミュレーションでは20台の車が必要かも知れませんが、ゲームでは2人のプレイヤーが、入力フォームでは4つのチェックボックスが必要かも知れません。

適切に利用すると、OOP はプログラムに概念的な構造レベルを追加します。相互に関連する関数や変数をクラス別にまとめ、各クラスは、自分のことは自分で責任を持つという自己独立的な、プログラムの部分です。クラスは、関数を実行したり変数を設定して、プログラムに動作させる、個別のオブジェクトを作成するために使用します。

コードをクラスに構造化することで、現実世界の構成要素を使って現実世界の問題を写しとるプログラム作成が容易になります。本書のパート と では、ActionScript で直面するよくある状況を扱い、それに OOP を適用する方法を述べます。しかし、その前に、少し OOP の基本コンセプトについて考えてみましょう。

オブジェクト指向プログラミング・コンセプトの鍵

オブジェクトはそれ自体独立したモジュールであり、関連する関数(メソッド)と変数(プロパティ)を持ちます。個々のオブジェクトは、オブジェクトのメソッドやプロパティに青写真を提供するクラスから作成されます。つまり、クラスはオブジェクトを作成するひな形なのです。クラスは、タイマーのような理論上の概念となったり、プルダウンメニューやスペースシップのような、プログラム内で実体的に存在するものになることができます。1つのレシピからいくつでもマフィンが焼けるように、1つのクラスを使って同じ構造を持つオブジェクトをいくつでも作成することができます。例えば、OOP スペースファイティングゲームでは、1つの SpaceShip クラスから作成された SpaceShip オブジェクトが、画面上に1度に20機必要かも知れません。同様に、そのゲームには数学的なベクトルを表す1つの 2dVector クラスが必要で、しかもゲーム内には何千もの 2dVector オブジェクトが必要かも知れません。

用語インスタンスはオブジェクトの同意語としてよく使用されます。例えば、“新しい SpaceShip インスタンスを作成する”と、“新しい SpaceShip オブジェクトを作成する”という言い方とは、同じ意味です。クラスから新しいオブジェクトを作成することは、インスタンス化と呼ばれたりします。

オブジェクト指向のプログラムを作成するには、

1. 1つかそれ以上のクラスを作成します。
2. そのクラスからオブジェクトを作成(インスタンス化)します。
3. オブジェクトに動作を伝えます。

オブジェクトの動作はプログラムのビヘイビア(振る舞い)を決めます。

クラスの作成に加え、プログラムは作成したクラスをどれでも、Flash Player 内で使用

することができます。例えば、プログラムは Sound オブジェクトを作成するビルトインの(予め組み込まれている)Sound クラスを使用できます。個々の Sound オブジェクトは、1つのサウンドやサウンドのグループを表したり、制御できます。Sound オブジェクトの setVolume()メソッドは、サウンドの音量を上げたり下げたりできます。loadSound()メソッドは、MP3 サウンドファイルを取得し、再生できます。さらに duration プロパティを使って、ロードしたサウンドの長さをミリ秒単位で知ることもできます。Flash では、ビルトイン・クラスと自分で作成するカスタム・クラスを使って、OOP アプリケーションの基本的なブロックを形成していくのです。

クラスのシンタックス

では実際の例を見ていきましょう。前に、スペースファイティング(宇宙戦争)ゲームでは SpaceShip クラスが必要かもしれないと述べました。クラスを定義する ActionScript のソースコードは、サンプル 2-1 のような形をしています(この類のコードが全く初めてであっても心配は要りません。これからこのチャプターで詳しく学んでいきます)。

サンプル 2-1:SpaceShip クラス

```
class SpaceShip{
    //speed という名前の public プロパティ
    public var speed:Number;

    //damage という名前の private プロパティ
    private var damage:Number;

    //コンストラクタ関数、各 SpaceShip インスタンスを初期化します
    public function SpaceShip(){
        speed = 100;
        damage = 0;
    }

    //fireMissile()という名前の public メソッド
    public function fireMissile():Void{
        //ミサイルを発射するコードを書きます
    }
}
```

```
//thrust()という名前の public メソッド
public function thrust():Void{
    //シップを移動させるコードを書きます。
}
}
```

SpaceShip クラスがプログラム内の関連する項目ごとにきちんと分類されていることに注意します(どのクラスでも同じです)。スペースシップに関連する speed や damage といった変数(プロパティ)は、スペースシップを移動させたり、武器を発射する関数(メソッド)とは分類されています。スコアを保存したり背景を描画するというような、プログラムの他の項目は、それら自身のクラスの中に分けて保存します(このサンプルでは示していません)。

オブジェクトの作成

オブジェクトは new 演算子で以下のように作成(インスタンス化)します。

```
new ClassName()
```

ClassName はオブジェクトを作成するクラスの名前です。

例えば、サンプルのゲームで SpaceShip オブジェクトを新規に作成するには、以下のコードを使用します。

```
new SpaceShip()
```

ActionScript2.0 でオブジェクトを作成するコードのシンタックス(ここでは new SpaceShip())は、ActionScript1.0 と同じです。しかし、ActionScript2.0 でクラスを定義シンタックスは、ActionScript1.0 と異なります。

ほとんどのオブジェクトは、作成すると後で使用できるように保持します。例えば、SpaceShip インスタンスは変数 ship に、以下のように保持します。

```
var ship:SpaceShip = new SpaceShip();
```

各オブジェクトは、変数や配列エレメント内、また他のオブジェクトのプロパティにも保持できる個別のデータ値です。例えば、20のエイリアン・スペースシップを作成すると、通常は、1つの配列に20のSpaceShipオブジェクトへの参照を保持します。これによって配列内を巡回し、例えば各オブジェクトにSpaceShipクラスのメソッドを実行することで、複数のオブジェクトの操作が容易になります。

オブジェクトの使用

オブジェクトのメソッドは、“ ミサイルの発射 ”、“ 移動 ”、“ スクロール・ダウン ” というような能力(ビヘイビア)を提供します。オブジェクトのプロパティは、現在の状態を示すデータを保持します。例えば、ゲームのある時点では、ship の現在の状態は speed が 300 で、damage が 14 となります。

オブジェクトのクラスで public として定義されたメソッドやプロパティは、プログラム内のどこからでもアクセスできます。対照的に、private として定義されたメソッドやプロパティは、クラスかサブクラスのソースコード内でしか使用できません。チャプター 4 で学ぶように、メソッドとプロパティは、外部からアクセスする必要がある場合のみ public として定義します。

メソッドを実行するには、ドット演算子(ピリオド)と関数を呼び出す演算子(かっこ)を使用します。例えば、

```
//ship オブジェクトの fireMissile()メソッドを実行します。  
ship.fireMissile();
```

プロパティを設定するには、ドット演算子と等記号を使用します。例えば、

```
//ship の speed プロパティを 120 に設定します。  
ship.speed = 120;
```

プロパティ値を取得するには、それ自体にドット演算子を使用します。例えば、

```
//speed プロパティの値を出力パネルに表示します。  
trace(ship.speed)
```

カプセル化

オブジェクトは、言ってみれば、プログラムの他の部分からプロパティ値やメソッドのソースコードを隠してカプセル化したようなものです。適切に設計されていると、オブジェクトの private プロパティやメソッドで使われている内部コード(public メソッドも含む)はそれ自体のものになります。つまり、オブジェクトの変更はプログラムの他の部分の変更を必要としないのです。メソッド名(とパラメータ、戻り値)が同じである限り、プログラムの他の部分は、書き換えることなくひきつづきそのオブジェクトを使用できるのです。

カプセル化は、独立した別のクラスで別のプログラマーが作業できるという理由から、オブジェクト指向の設計において重要な側面となります。プログラマーが、やりとりに使用する `public` メソッドの名前を変えない限りは、クラスは個別に開発できます。さらに、`public` で使用できるメソッドや必要なパラメータ、戻り値を示す仕様を開発することで、デプロイ(統合して配置)する前に、時間をかけてテストすることができます。その同じテストコードは、たとえそのクラスのコードがリファクタリング(コードの体質改善：パフォーマンスを改善したり、現在備えている機能を変更することなくソースコードを単純化すること)されていたとしても、クラスの働きが正確か再確認することにも使用できます。

チャプター 4 では、プログラムの他の部分からメソッドやプロパティにアクセスできないようにする `private` 修飾子の使い方を学びます。

データ型

オブジェクト指向プログラムでは、個々のクラスは、データ独自の種類を定義するものと考えることができます。形式的にはこれをプログラムのデータ型と呼びます。

クラスは、実際にはカスタムデータ型を定義することです。

たぶん、`Date` クラスなどのビルトイン `ActionScript` クラスで定義されたカスタムデータ型にはもうなじみがあるでしょう。`new Date()` を使って `Date` オブジェクトを作成すると、このオブジェクトは、ストリングでもなく数値でもない、特定の年月日を定義した複雑なデータ型を持った値を返します。このように `Date` データ型は日付に関する独特のプロパティとメソッドをサポートしています。

データ型は、パラメータとして使用する変数や、戻り値として渡す変数内に保持できるものに制限を与えるために使用します。例えば、前に定義した `speed` プロパティでは、そのデータ型として `Number`(太文字で表示)を指定しました。

```
//説明：“: Number”は speed のデータ型を定義します。
```

```
public var speed : Number;
```

試しに数字でない値を `speed` プロパティに入れてみると、コンパイル時にエラーが生じます。

ムービーをプレビューして Flash の出力パネルに、“タイプが一致しない”というエラーが表示された場合は、プログラム内のどこかで間違ったデータの種類を使用していることが分かります(コンパイラは正確にその場所を教えてくれます)。データ型は、プログラムが意図しない方法で使われないように保証して、プログラマーを助けてくれます。例えば、speed のデータ型を数値に指定することで、誰かが speed にストリング “very fast” を設定するなどといった、意図しない設定を避けることができます。以下のコードはデータ型が一致しないためコンパイル時にエラーが生じます。

```
public var speed : Number = “very fast”;      //エラー！
                                              //Number 型の変数にストリングは設定でき//
                                              //ませ//ん。
```

継承

オブジェクト指向のアプリケーションを開発するとき、継承を使って、1つのクラスに他のクラスのメソッドやプロパティ定義を適用することができます。継承を使うと、アプリケーションを階層的に構造化して、1つのクラスの機能を再利用できるようになります。例えば、Car、Boat、Plane クラスは、汎用的な Vehicle(乗り物)クラスの機能を再利用することで、アプリケーションから冗長(余剰)分をはぶくことができます。冗長を少なくすることは、書いたりテストしたりするコードが減ることを意味します。さらに、コード変更が容易になります。例えば、1つのクラスで行う移動のアルゴリズムのアップデート作業は、いくつかのクラスを通してアップデートするよりも、容易で間違いも少なくてすみます。

他のクラスからプロパティやメソッドを継承するクラスは、サブクラスと呼ばれます。サブクラスがプロパティやメソッドを継承するクラスは、サブクラスのスーパークラスと呼ばれます。当然、サブクラスは、スーパークラスから継承したプロパティやメソッドに加えて、それ自体のプロパティやメソッドも定義できます。1つのスーパークラスは、1つ以上のサブクラスを持てますが、1つのサブクラスは1つのスーパークラスしか持てません(たとえスーパークラスのスーパークラスから継承できたとしても)。継承に関してはチャプター6で詳しく扱います。

パッケージ

大きなアプリケーションでは、クラスのグループを含むパッケージを作成することが

できます。パッケージによって、クラスを論理的なグループに系統だて、クラス間での名前コンフリクトを避けることができます。これはコンポーネントやサードパーティ製のクラスライブラリが含まれているとき特に役立ちます。例えば、Flash MX 2004 の Button という名前の GUI コンポーネントは、mx.controls というパッケージ内にあります。GUI コンポーネントの Button という名前のクラスは、mx.controls パッケージの一部として識別されなければ、Flash のビルトインの Button クラスと混同してしまいます。実際は、パッケージはディレクトリで、クラスファイルのコレクション(.as ファイルを集めたもの)です。

パッケージ内のクラスを参照することで名前コンフリクトを避けることは、他のことから含めチャプター 9 で学びます。

コンパイル

OOP アプリケーションを Flash ムービー(.swf ファイル)として書き出すと、各クラスはコンパイルされます。つまりコンパイラが、各クラスをソースコードからバイトコード—Flash Player が理解でき実行できる命令--に変換しようとしています。クラスにエラーが含まれていると、コンパイルは失敗し、Flash のコンパイラが Flash オーサリングツールの出力パネルにエラー内容を表示します。前述したデータ型の不一致といったエラーメッセージは、問題の原因を突き止め解決する助けとなります。ムービーのコンパイルが成功しても、プログラムの実行中にエラーが発生するかも知れません。これをランタイムエラーと呼びます。Player から発生するランタイムエラーとプログラムから発生するランタイムエラーについては、チャプター 10 で学びます。

ここまで、Flash における OOP の概論で、オブジェクト指向のアプリケーションはクラスとオブジェクトから作られることを見てきました。しかしまだ実際には、アプリケーションの動作開始方法を学んでいません。Flash アプリケーションはどれも、クラスやクラスを含む外部アセットがいくつであろうと関係なく、Flash Player にロードされた 1 つの .swf ファイルとして動作をスタートさせます。Flash Player は、新たに .swf ファイルをロードすると、フレーム 1 のアクションを実行した後、フレーム 1 の内容を表示します。

オブジェクト指向 Flash アプリケーションを作成するときは、最も単純な場合、以下のように作成を始めます。

1. .as ファイルに 1 つかそれ以上のクラスを作成します。

2. .fla ファイルを作成します。
3. .fla ファイルのフレーム 1 に、クラスのオブジェクトを作成するコードを記述します。
4. 場合によっては、オブジェクトのメソッドを実行して、アプリケーションをスタートさせます。
5. .fla ファイルから.swf ファイルを書き出します。
6. Flash Player に.swf ファイルをロードします。

CHAPTER 5 と 11、12 では、オブジェクト指向 Flash アプリケーションを作成、動作させるさらに複雑な方法を学びます。

ではどのように OOP を使うのか？

OOP の基本を学んだ多くの人々は、“専門用語もコンセプトも分かった、でもそれをいつどうやって使えばいいかさっぱり分からない”と言います。もし今そのように感じていたとしても心配は要りません。それが普通です。現段階は、オブジェクト指向デザイン(設計)(OOD)を学ぶ次のステップに上がる準備ができた段階なのです。

OOP の核となるコンセプト(クラス、オブジェクト、メソッド、プロパティなど)はただの道具にすぎません。本当のチャレンジは、それらの道具を使って作りたいものをデザイン、設計することです。ハンマーや釘、木を理解したといっても、壁や部屋、椅子を実際に作れるようになる前に、まだ計画の青写真(設計図)を書く必要があります。オブジェクト指向デザインは、アプリケーションの総体を一連のクラスとして組織しながら、オブジェクト指向プログラミングの“計画の設計図を書く”段階です。プログラムをクラスに分割することは、OOP の作業で毎日のように直面する、基礎的な設計上の問題をはらんでいます。本書では OOP のこの重要な側面に常に戻って検討をつづけていきます。

全ての Flash アプリケーションにオブジェクト指向が必要なのかというとそうではありません。Flash では手続き型プログラミングもオブジェクト指向プログラミングも両方をサポートされており、1つの Flash ムービーに両方のアプローチを組み合わせで使用することもできます。場合によっては、プロジェクトの1つの部分だけに OOP を使うのが適切な場合もあります。おそらく写真を表示するセクションを持った web サイトを作っていることでしょう。サイト全部をオブジェクト指向にする必要はないのです。OOP は写真を表示するモジュールを作成するために使えばよいのです(実際にこの作業を CHAPTER 5 と 7 で行います！)

では、Flash が手続き型とオブジェクト指向のプログラミングを両方ともサポートしているからといって、プロジェクトにそれらをどのくらい使うのが正しいのでしょうか？ この問いに対する最良の答えを得るには、まず各 Flash ムービーの基本構造を理解する必要があります。Flash ドキュメント(.fla ファイル)の基本的な組織構造は、1つかそれ以上のフレームを持つタイムラインです。各フレームでは、ステージと呼ばれるグラフィカルなキャンバスで表示する内容を示します。Flash Player では、フレームは、アニメーションの効果をかもし出す – 正確にはスライドのフレーム(パラパラ漫画)のような、リニアシーケンス内で1度に1フレーム分表示されます。

Flash を使った開発では、タイムラインがインタラクティブなアニメーションやモーション・グラフィックによく使用されます。 この開発スタイルでは、コードは直接タイムラインのフレームやグラフィックコンテンツ両方に記述されます。例えば、ムービーが25フレーム分のアニメーションを表示し、停止、他のアニメーションを表示するランダムな計算をして、再び停止した後、背景でまた別のアニメーションを表示しながらユーザーにフォームへの記入を求める、というものです。つまり、シンプルなアプリケーションでは、タイムラインの別フレームを使用すると、別のプログラムの外見を表すことができるのです(外見は、**プログラム内でユーザーが???**)。例えば、1つのフレームではウェルカム・スクリーンを表し、別のフレームではデータを入力スクリーンを、また別のフレームでは、エラーを表示したり出口を表す画面として、といった具合に使用できるのです。もちろん、アプリケーション内にアニメーションがあれば、各プログラムの外見は、1フレームでなく何フレームかに渡って表されるかも知れません。例えば、ウェルカム・スクリーンにはループするアニメーションが含まれている可能性もあります。

モーション・グラフィックスに強く依存するコンテンツを開発するときは、正確にグラフィック要素を制御できるという理由で、タイムラインを使うのが理にかなっています。このスタイルでの開発では、コードは通常、Action パネル(F9)を使ってタイムラインのフレームに適用します。フレームのコードは、そのフレームの内容が表示される直前に実行されます。またコードはステージ上のグラフィック部分にも直接適用されます。例えば、ボタンは、クリックされたときに発生させることを決定するコードが記述できます。

タイムライン・ベースの開発は通常、再生ヘッドがあるフレームに達したときある動作を起こしたいという理由から、手続き型プログラムと連携して進められます。Flash での“**手続き型プログラミング**”は、ドキュメント内のフレームやステージ上のグラ

フィック部品で、コードを実行し、関数を定義し、変数を設定することを意味します。

しかし Flash コンテンツの全てにタイムライン・ベースのモーションが必要かという
とそうではありません。ビデオゲームを作成する場合、タイムラインを使ってモン
スターやプレイヤーのキャラクターの位置を決めることは不可能です。同様に、いつユ
ーザーがモンスターを撃つか、ほかの行動を起こすか、正確には分かりません。それ
ゆえタイムラインでなく ActionScript を使ってユーザーの行動に反応する(または、何
らかの知的な方法でモンスターを制御するアルゴリズムに反応する)キャラクターの位
置を決める必要があるのです。予め決められているアニメーション・シーケンスを含
んだ、タイムライン・ベースのプロジェクトでなく、キャラクターやその行動がコー
ド内に全て書かれている非リニアのプロジェクトを作成します。

このタイプの開発は、当然のことながら、例えば、プレイヤーのキャラクターやモン
スターを表すオブジェクトに向いています。開発におけるもう一方に、アプリケーシ
ョンがクラスのグループとして存在する、伝統的なオブジェクト指向プログラミング
があります。純粋なオブジェクト指向 Flash アプリケーションでは、.fla ファイルは、
ただアプリケーションのメインとなるクラスをロードし、そのクラスのメソッドを実
行してアプリケーションをスタートさせるという、1フレームだけを含むものなの
かも知れません。もちろん、OOP をただのビデオゲーム作成に使用するのはいまい
いことです。例えば、Flash ベースのレンタカー予約システムには、タイムラインの
コードは必要ありませんし、クラスからユーザーインターフェイス要素を全て作成
ことになるのです。

最も実地的な Flash アプリケーションは、タイムラインを使うだけの開発と、純粋な OOP
開発の両端の間に位置します。例えば、2つのボタンが画面の真ん中にスライドし、“英
語”か“フランス語”かをユーザーに選ばせる、Flash ベースの Web サイトを考
えてみてください。ユーザーは好みの言語ボタンをクリックし、両方のボタンは画
面をスライドし消えます。その後アニメーションのシーケンスで会社情報が表示
され、製品デモを見せるビデオが映ります。ビデオは MediaPlayer コンポーネ
ントで制御されています。

この仮定の Web サイトには、以下のような手続き型プログラミングと OOP が両方
含まれています。

- フレーム 2 と 3 には、プリローダーのコードが書かれています。
- フレーム 10 には、ボタンがスライドするアニメーションを開始するコードが書

かれています。

- フレーム 11 から 25 には、ボタンがスライドするアニメーションが入っています。
- フレーム 25 には、言語特定のムービーをロードする、ボタンのイベントハンドラを定義するコードが書かれています。
- ロードした言語特定のムービーのフレーム 1 には、MediaPlayback コンポーネントを制御するコードが書かれています。

この例では、フレームに直接配置されたコード(プリローダーのコード)は手続き型です。しかしボタンや MediaPlayback コンポーネントは、外部の.as ファイルに保持されたクラスからもたらされたオブジェクトです。これらを制御するには、オブジェクト指向プログラミングが必要です。さらに面白いことに、Flash コンポーネントそれ自体がムービークリップです。Flash 独特のムービークリップは、タイムラインとフレームを内部に持つ、独立したオブジェクトだと考えられます。コンポーネント(実はムービークリップ)は、オブジェクトであっても、内部のフレームに手続き型のコードを持つことができます。こうしたことが Flash 開発の自然な姿なのです。手続き型コードを含むアセットは、オブジェクト指向コードと複数のレベルで混在させることができます。

序論で述べたように、本書ではムービークリップを理解し、これまでに使用した経験があることを前提としています。他の言語を学ばれて、新たに ActionScript を始められるプログラマーの方は、プログラマーの立場から見たムービークリップについて ActionScript for Flash MX: The Definitive Guide(O'Reilly)(<http://moock.org/asdg/samples>で見られます)のチャプター 13 を参考にざっと勉強してみてください。

Flash では、グラフィカルなタイムライン開発環境で、手続き型とオブジェクト指向のコードを組み合わせることができるので、他に例がないほど融通がききます。この柔軟性は強い見方でもあり危険でもあります。一方では、Flash ではほんの小さなアニメーションやインターフェイスの動きが、C++や Java などの言語ではカスタムコーディングに多くの時間が必要になるという側面があります。しかし他方では、タイムラインのフレームやステージのコンポーネントに適用したコードは、それを見つけ修正するのに時間がかかってしまいます。タイムライン・コードの使いすぎによって、プロジェクトはすぐさま(そして静かに!)維持しづらい混乱状態に突入してしまいます。オブジェクト指向のテクニックでは、グラフィックやサウンドなどのアセットからコードを切り離すことが強調され、そのことによって、タイムライン・ベースのプログラムと比べて、オブジェクト指向のアプリケーションの変更や再利用、拡張が容易に

なるのです。もし変更直面したタイムライン・ベースのプロジェクトの真っ只中にいて、その作業にうんざりしているのなら、そのプロジェクトは、オブジェクト指向で開発を始めるかっこうの機会となります。OOP には余計な開発時間がかかると思えるかも知れませんが、そのプロジェクトに費やした多くの時間を、大きなプロジェクトで取り戻すことができます。

最終的に作業で使用する OOP の総量は、経験やプロジェクトに必要なものによって変化するもので、各自で決定すべきことです。以下のリストは、いつ OOP を使うか、いつ手続き型のタイムライン・コードを使うかを決めるときの判断材料になります。しかし、これはただのガイドラインであることを覚えておいてください。アプリケーションを作成する方法は常に何通りもあります。最終的にソフトウェアが動作し、維持できれば、行ったことは正しかったと言えます。

以下を作成するとき、OOP の使用を検討します。

- 画面トランジションがほとんどない標準的なユーザーインターフェイスを持った、従来のデスクトップスタイルのアプリケーション
- サーバーサイドロジックを含むアプリケーション
- 複数のプロジェクトで再利用する機能性
- コンポーネント
- ゲーム
- 複雑なトランジションを含む高度にカスタマイズされたユーザーインターフェイス

以下を作成するとき、手続き型プログラミングの使用を検討します。

- 動きや基本的なインタラクティブ性を制御する短いスクリプトが書かれたアニメーション
- 1 ページの製品オーダーフォームやニュースレター講読フォームのような単純なアプリケーション
- 複雑なトランジションを含む高度にカスタマイズされたユーザーインターフェイス

お気づきでしょうが、“複雑なトランジションを含む高度にカスタマイズされたユーザーインターフェイス” は、OOP にも手続き型プログラミングでも使用するケースに含まれています。両方のタイプとも、コンテンツの作成に効果をあげることができます。しかし、通常 Flash の OOP はタイムライン・コードよりもメンテナンスしやすく、バージョン管理システムや他の外部制作ツールへの統合が容易であることを忘れてはい

けません。高度にカスタマイズした UI を1つ以上のプロジェクトで使用できるかどうか疑わしい場合は、使用可能なクラスライブラリや OOP を使ったコンポーネント・セットとして開発できないか、熟考すべきです。

Flash の従来からのタイムラインというメタファーに加え、Flash MX Professional 2004 では Screen 機能(Slide と Form を含む)が導入されたことにも注目する必要があります。Screen は従来からのタイムラインというメタファーを覆う外観を提供します。Slide と Form は、HyperCard のような、プログラムのカードベースのメタファーと同じようなものです。Slide は PowerPoint スタイルのスライド・プレゼンテーションを意識して、Form は複数ページのフォーム上で作業する VB 開発者を意識しています。タイムライン・ベースのアプリケーションのように、Screen ベースのアプリケーションも、オブジェクト指向コード(クラス内コード)と手続き型コード(ビジュアル・コンポーネントやアプリケーションの Screen に配置したコード)両方を含みます。序論で述べたように、本書では Screen について詳しく扱いませんが、本書で学ぶ基礎的な OOP スキルは、自分で Screen について学ぶより多く身に付くことでしょう。

本番です！

このチャプターでは、Flash における OOP の核となるコンセプトの要点を述べました。さあ移動する準備はできました。パート の残りでは、コンセプト全てについてもう一度詳しく見、方法にそって実践的な状況に適用します。もう OOP に対してリラックスできサンプルの中に飛び込んでいきたい場合は、チャプター 5 , 7 , 12、パート 全文をご覧ください。そこでは実際的なオブジェクト指向コードの詳しい勉強が待っています。

さあ、始めましょう。