オブジェクトの広場

［Happy Squeaking!!］

― オブジェクト指向再入門 ―

[第一回] [第二回] [第三回] [第四回] [第五回]

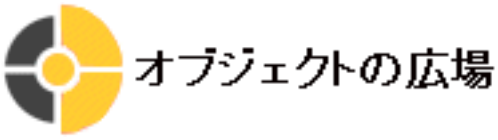
第一回：あらまし、環境構築　編

ＩＮＤＥＸ

4. 参考文献

記事に対するご意見・ご感想をお待ちしています。
umezawa@tyo.otc.ogis-ri.co.jp
気軽にお寄せください。

HOME TOP



［Happy Squeaking!!］

1．はじめに

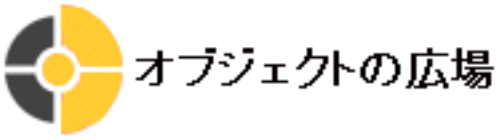
1．1　もう一度、オブジェクト指向！

以前にくらべオブジェクト指向の概念はかなり一般的になってきました。
街に出ればオブジェクト指向の入門書はあふれかえっています。
皆さんの中にも、C++やJavaで普段からプログラムをされている方、さらに分析、設計のモデリング表記法としてUMLを採用されている方も多いのではないでしょうか。

そのような中で、あえてオブジェクト指向入門を開始したいと思います。
目標は、「できるだけわかりやすく、基本に忠実に」ということです。
時代の風潮には合わないかもしれませんが、正攻法で攻めていくことを目指しています。
「わかりやすく」は「易しく」を意味しません。多少の高度な概念でも、オブジェクト指向に本質的な部分はしっかりと丁寧に説明していきます。

そのため教材としてSmalltalkを使用します。
SmalltalkはPureなオブジェクト指向言語です。一切の妥協を許さず、すみからすみまでオブジェクト指向です。その点においてはオブジェクト指向の教育には最適であり、実際に米国、ヨーロッパの多くの大学でコースが開催されています。

Index Next



［Happy Squeaking!!］

1．はじめに

1．2　なぜ Smalltalk？

1．2．1 Simple and Pure

例えば、C++をオブジェクト指向の入門に使用すると、まず突き当たるのはその言語仕様の複雑さです。C++は手続き指向の言語であるCとの互換性を保つことを強いられたので、オブジェクト指向の本質とはあまり関係のない部分が多く言語仕様に入り込み、習得のし難い言語になってしまっています。(このように手続き指向をオブジェクト指向に拡張し、両者の概念の混在する言語をハイブリッド系OO言語といいます)。またC++のようなハイブリッド系言語は、手続き指向でのプログラム作成が全く問題なく行えてしまうので、「C++を使用しているが、オブジェクト指向になっていない(ただのCプログラミングをしている)」ということが起こってしまいます。

更に、なんとかC++を使いオブジェクト指向の考えでプログラミングできるようになったとしても、言語の性質上、メモリ管理の問題が常につきまとうことになります。もともとオブジェクト指向は、システムにまつわる問題を抽象的に捉えられるところに利点があり、メモリ管理といった低レベルの部分はなるべく気にせずに、アプリケーションドメインといった上位の概念に注目したいはずです。C++ではメモリ管理をみずから行わなければならないため、アプリケーションに本質的な処理と、単なるメモリ管理の処理が入り交じり、コード自体を非常にわかりにくくする側面があります。また人間の手によるメモリ管理は、必ずといっていいほどメモリリーク現象を引き起こします。このために、「あまり多くのオブジェクトを用意しないようにしよう」という考えが働くようになり、設計上は分離されていたはずのクラスが実装時に安易に合成され、奇麗なオブジェクト指向設計をくずしてしまうこともあります。

一方Javaはどうでしょうか。Javaはかなり奇麗にオブジェクト指向の考えを実現しているといえますが、数値や配列などの「基本データ型」はオブジェクトではありません。そのために「全てを統一的にオブジェクトとして扱えるはず」というオブジェクト指向の原則に合致せず、やはり不自然なコードを書かねばならない場面がでてきます。
例としてJavaでは、「何でも入れてしまえる順番つきの入れ物」を表すのに、Vectorというクラスが用意されています。このVectorに物をいれるときに、基本データ型が混ざっていると以下のようなコードを強いられます。

```
Vector ve = new Vector();  // 入れ物
Vector ve2 = new Vector(); // 入れ物の中に入る入れ物
```

```
ve.addElement(ve2); // ve2をve1に入れる（これはOK)
ve.addElement(new Integer(1)); // 数値をいれるときはInegerに変換
ve.addElement("hello"); //文字列は基本データ型ではないのでこれでOK
ve.addElement(new Character('a'));
        // 文字は基本データ型なのでCharacterに変換
```

なぜ、以下のようにすっきりと書けないのでしょう。

```
//全てをオブジェクトとして統一的に扱う。
ve.addElement(ve2);
ve.addElement(1);
ve.addElement("hello");
ve.addElement('a');
```

また、Vectorから物を取り出すときも、基本データ型かどうかを強く意識する場面がでてきます。

```
Enumeration enu = ve.elements();
while(enu.hasMoreElements()){
        Object o = enu.nextElement();
                //ここまでは統一的。要素を取り出している。
}
```

ところが、実際にその値を使うときには、以下のような形で変換が必要になります。

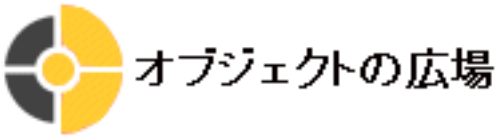
```
if(o instanceof Integer){
        int i = ((Integer)o).intValue();
                //Integerをもとの基本データ型に戻す
} else if( o instanceof String){
        String s = (String)o;
                //ObjectをもとのStringにダウンキャストする。
} else if ( o instanceof Character){
        char c = ((Character)o).charValue();
                //Characterをもとの基本データ型に戻す。
}
```

このようにif文でそのオブジェクトが実際に何者かを判断して、基本データ型か、そうでないオブジェクト型かによって処理を分けるというのは、非常に不自然なやり方と感じられないでしょうか。本来オブジェクト指向とは、オブジェクトがある程度インテリジェンスな振る舞いを持ち、使用する側では意識しなくとも、そのオブジェクトの特性に応じて処理を行ってくれるもののはずです。なぜ、アプリケーション開発者の側で、「これは基本型だから」「これはオブジェクト型だから」といちいち考えていなければならないのでしょう。

ところがSmalltalkでは、このような妥協、不自然な部分はみられません。
文法は、オブジェクト指向の実現に不可欠な部分のみを含み、非常に簡潔にまとまっています。習得に一日もかからないでしょう。メモリ管理は自動で、複雑なオ

ブジェクトを万のオーダで作成しても、メモリリークにおびえる心配もありません。キャストの必要もなく、全てがオブジェクトという概念世界が、非常に忠実に表現されています。


Prev. Index Next



[Happy Squeaking!!]

1．はじめに

1．2　なぜ Smalltalk？

1．2．2　Highly Interactive

もうひとつ、教材としてSmalltalkを取り上げる理由として、その環境のインタラクティブ性ということがあります。

オブジェクト指向を学ぶときに、「プログラムの中でのオブジェクトが実際に目に見えたらどんなにわかりやすいだろう」と思ったことはないでしょうか。
また、現在自分が使えるクラスにはどんなものがあり、どんなメッセージが利用できるのか、更に、あるメッセージはライブラリのどこで呼ばれているのか、あるクラスはどこで生成されているのかといったことをすぐに知ることができたらと考えたことはないでしょうか。

こうした要求は、オブジェクト指向で開発をする際には非常に自然であるといえるでしょう。オブジェクトにメッセージを送信するのがオブジェクト指向のもっとも基本的な部分とすると、開発者側も、ファイル(に書かれているプログラム)を対象とするよりは、そこで表現されているクラスやオブジェクトを対象としたいと考えるはずです。
このような考えに立てば、クラスやオブジェクトを対象として、インタラクティブなやり取りの中でシステムが開発できる姿が理想的なのですが、多くの統合開発環境は、オブジェクト指向言語をサポートしていたとしても、ファイルを中心に構成されているというのが現状です。

一方でSmalltalkは、最初からこのような理想系をかなり実現した形で存在しています。
Smalltalkは、もともと1980年の昔にXeroxのPalo Alto研究所で、Alan KayやDan Ingallsといった若手のエンジニアによって開発されました。当時彼らが夢見ていた理想は壮大なものがあり、その一つはDynaBookという名前で知られています。DynaBookは、子供も大人も使えるコンパクトな一種のノート型コンピュータで、マウスとビットマップディスプレイとネットワーク機能を持ち、ユーザの要求が、プログラミングレスで対話的なやり取りによって実現できるといったコンセプトを表していました。そのDynaBook上で、対話的インターフェースを提供する一種のOSとして考えられていたのがSmalltalkだったのです。(DynaBookのアイデアの多くは後にMacintoshなどに採用されていくことになります)。
このような経緯から、Smalltalkはもともと非常にインタラクティブな側面を持っていたのです。
通常のプログラミング言語と異なり、言語は開発環境にビルトインされた形で提供され、その高度な統合の具合は、既存のビジュアルツールの利用者でも舌を巻いて

しまうほどの完成度を持っています。

Smalltalk環境の中では、実際にオブジェクトに語りかけるようにして、プログラムの開発が可能です。開発に必要な操作の単位がオブジェクトレベルであり、既存のファイルを中心にした開発環境とは一線を画しています。
例えばtoString()というメソッドの実装されている箇所を一覧したいと思った場合、ファイルベースの環境では、全てのファイルに対して grep toString() などと外部ツールを起動して地道に該当する箇所を探し、複数のファイルを一々開いてみる必要がありました。これに対しSmalltalkでは、" Smalltalk browseAllImplementorsOf: #printString" (printStirngはJavaでのtoString()に該当)のSmalltalk式の実行で、たちどころに該当部分がまとめられたブラウザを開くことができます。

もうひとつ、対話的な機能を支えるものとして、バイトコードインタプリタという特徴があります。
これはJavaでも採用されている方法なのですが、Javaの場合はマルチプラットフォームの実現ということ以外に今一つその利点が活かされていません。Smalltalkは、システム内部にバーチャルマシンのシミュレータを持ち(これも全てSmalltalkで記述)、実際のVMの実行でバグが起こった場合に、そのシミュレータ上で実行させ、デバッグを行っていくことができます。これにより「直しながら実行」ということが可能です。ステップ実行させながら実装を修正し、いま修正した直前から実行を即座に再開できてしまいます。また、バイトコードへのコンパイルの単位はメソッドごとで、Javaのようにあるクラスの1メソッドを書き直したら、そのクラスをすべてクラスファイル単位でコンパイルし直さなければならないといった不便さはありません。1メソッドのコンパイルにかかる時間は一瞬であり、開発者は「コンパイル待ち」といった作業の流れを絶つような非効率的なことにとらわれずにオブジェクト指向に専念できます。

Prev. Index Next

オブジェクトの広場

［Happy Squeaking!!］

1．はじめに

1．2　なぜSmalltalk？

1．2．3　Meta, Meta, - Better, Better !

また、Smalltalkでは、メタプログラミングを行うための多くの機能が備えられています。インスタンス、クラスという通常の抽象度の区分に加えて、クラスの記述を行うための、更に抽象度を高めたメタクラスが用意されています。オブジェクト指向関係の入門書では、このメタクラスの存在について触れながらも、きちんとした説明を行っていないものが多く見られます。
Smalltalkでのメタクラスがどのようなものなのか理解することによって、メタという概念についての理解が進むことでしょう。

メタ機能は通常の小規模なプログラミングではあまり問題になることがありません。しかし、柔軟な変更が要求される再利用可能なソフトウェアを作成するには今後非常に重要な概念として認識されていくことが考えられます。
例えばCORBAのデザインパターンで有名なTomas Mowbleyは、「CORBA Design Pattern アーキテクチャからプログラミングまで」(IDGコミュニケーションズ刊)の中で、HVMというパターンについて述べています。Hは相互運用性を高めるための水平インターフェース(Horizontal)、Vはドメインごとに分割された垂直インターフェース(Vertical)を指します。これに対し、Mは、HとVを柔軟に変更可能にするためのメタインターフェースを指しています。メタ機能の採用により、大規模な分散環境内で各部分を単純に保ったままで、実行時の複雑なカスタマイズ要求に対応できるようになるのです。

このようなメタ機能(自己記述、変更能力)が素直に実現できているのがSmalltalkです。他の言語ではトリッキーに処理せざるを得ないこの機能もSmalltalでは実に簡潔に表現されています。

例えば、あるクラスで定義されている全てのインスタンス変数名を知るには、Javaでは以下のように記述する必要があります。

```
Some s = new Some();          //あるクラス(Some)のインスタンス生成
Class cls = s.getClass();     //インスタンスからクラスの代理オブジェクト生成
Field[] flds = cls.getDeclaredFields();
                  //クラスから宣言されたフィールドオブジェクトを取得
String[] names = new String[flds.length];
                  //名前の文字列の配列を用意
for(int i = 0 ; i<flds.length; i++){
        names[i]="flds[i].getName();"
                  //名前を取り出し配列に格納
}
```

これに対しSmalltalkでは

```
names := Some allInstVarNames. "クラスに対し直接フィールド名を聞く"
```

とするのみです。

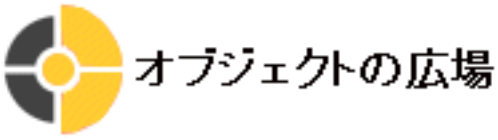
更に、Someの実行時に生成されている全てのインスタンスを取得するには、

```
insts := Some allInstances.
```

と同様に実現できます。Javaでこれを行う場合には、そのままではできず、SomeInstancesといったスタティック変数をSomeに用意し、その変数にSomeインスタンスが生成時に格納されるようにコンストラクタをオーバーライドしなければならないでしょう。

メタ機能実現の容易性により、Smalltalkでは、HVMのM部分をより積極的に使った柔軟なシステムが作りやすくなります。Smalltalk以外の言語を使う開発においても、開発者がメタ機能によりどのような部分が便利になるのかを知っておくことは有益と考えられます。

Prev. Index Next



［Happy Squeaking!!］

1．はじめに

1．2　なぜ Smalltalk？

1．2．4　Pattern Sensitive

皆さんの中にはMVCという言葉を聞かれたことのある方がおられるかと思います。これはModel、View、Controllerの略で、もともとSmalltalkでのGUI作成のためのパターンとして考えられたものです。
MVCは、「デザインパターン」(ソフトバンク)の冒頭に、非常にオーソドックなデザインパターンの例として紹介されています。Smalltalkのクラスライブラリには、MVC以外にも、多くの示唆を与える重要なパターンが表現されており、この「デザインパターン」本に対して多くの影響を与えています。実際この本にも非常に多くのSmalltalkによるパターン実装例が紹介されています。

Smalltalkは、言語はシンプルである反面、環境とともに提供されるクラスライブラリ群は非常に膨大であり、それらは、Smalltalk-80からの長い歴史によって磨かれ「デザインパターンの宝庫」と呼べるほどに洗練された、豊富なものになっているのです。

日本語訳は出ていないのですが、「デザインパターン」本の姉妹編として、"Design Patterns Smalltalk Companion" (Addison-Wesley 1998) という本もあります。これはオリジナルのデザインパターン本でガンマらが挙げた23パターンをSmalltalkライブラリ内での別の例で、再び説明、補足したものです。これもSmalltalkライブラリが実に多くのパターンを保持しているということを示しています。

このようにデザインパターンとSmalltalkの関係は非常に親密であり、Smalltalkのクラスライブラリを学んでいくことによって、デザインパターンのよい実例を知ることができるのです。

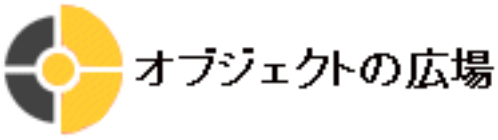
以上、Smalltalkを教材として使う理由をまとめます。

1. オブジェクト指向の考えが素直に表現されている。
2. 環境がインタラクティブなので、学習していて楽しい。
3. メタクラスなど、他には見られない高度な概念もあり新たな視野が開ける。
4. デザインパターンの良い例が随所にちりばめられている

Smalltalkは、残念ながら日本ではかなりマイナーです。その言語の持つ良さは多くの人の目に触れることはありません。しかしオブジェクト指向の先駆者である米国やヨーロッパでは、意外に広く産業界で使われています。商用の実装も多くあります。
グローバルな昨今、次の開発はSmalltalkということになるかもしれません.

ここらで食わず嫌いはやめてやってみましょうか！

Prev. Index Next



［Happy Squeaking!!］

1．はじめに

1．3　教材

教材にはSqueakというフリーの開発環境を使用します。
入手の仕方などについては後述します。

1．4　対象となる読者

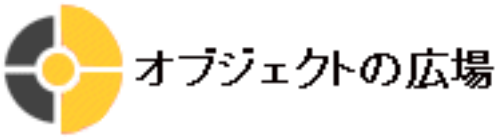
「プログラミング」の概念がわかっている方でしたら基本的に大丈夫です。オブジェクト指向の基本概念の理解があればベターですが、いままでオブジェクト指向が難しく思えて、取りかかれなかったかたも恐れることはありません。
また、普段からC++やJavaを当たり前のように使っているというハイグレードな方にも、それなりの楽しみはあるかと思います。純粋なオブジェクト指向に触れてきっと新たな発見があることでしょう。

1．5　全体構成

大まかに以下のような構成を予定しています。

1: Smalltalk環境(Squeak)の導入(今回)
2: オブジェクト指向の基礎固め
3: メタ機能に触れる(メタクラス、イントロスペクション、インターセッション)
4: Smalltalkでのデザインパターン(MVC、Morphic、Adapter、Proxy)

Prev. Index Next



［Happy Squeaking!!］

2. Squeak の入手

2. 1 Squeak とは？

フリーのSmalltalkの実装です。現在フリーのSmalltalkはSqueakのほかにも幾つかあります。そんな中でSqueakを選んでいるのは、やはりちょっとした理由があります。

●マルチプラットフォーム
Squeakは様々なプラットフォームで動作しますので、多くの方に実際に試してみていただくことができます。

・ Windows-NT
・ Windows-95
・ Dos
・ Windows-CE
・ Macintosh
・ OS/2
・ Acorn
・ BeOS
・ UNIX
・ Linux
・ Zaurus

Javaと同じようにVM上で動作しますので、ソースコードは、これらのプラットフォームで全て共通になります。

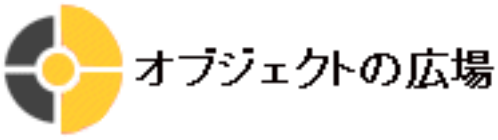
● 血筋が良い
Squeakの設計者には、Dan Ingalls、Alan Kayといった、Smalltalk-80のオリジナルの開発に携わったメンバーが含まれています。商用のSmalltalkに比べてもクラスライブラリは非常に奇麗です。

● コンパクト
一昔前のマシンでも快適な速度で動作します。ディスク容量も10Mもあれば十分です。マシン環境が整っていなくても大丈夫!!
Squeakという語はハツカネズミなどがチューチューなく声を表すのですが、まさにそうしたかわいい雰囲気を持っています。

● オープン
ソースコードは、VMも含めてすべて公開されています。多くのコミュニティがメーリングリストなどを通じて活発な議論を行い、Squeakの発展に貢献しています。

ここではSqueakの生い立ちについてこれ以上の詳しい解説はしません。
興味のある方は、umejavaさんのSqueak紹介ページを見てください.。
(http://www.mars.dti.ne.jp/~umejava/smalltalk/squeak/index.html)

Prev. Index Next



［Happy Squeaking!!］

2. Squeak の入手

2. 2 Squeak のダウンロード

2. 2. 1 Squeak の HomePage にアクセス

それでは、さっそくダウンロードからはじめましょう。

Squeakのリソースは、イリノイ大学のSqueak Archive Home Pageで管理されています。
(http://squeak.cs.uiuc.edu/) <= CLICK NOW!!

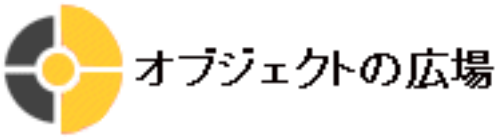
アクセスできましたでしょうか?

FTPサイトには、zipなどでアーカイブされたものと、個々のファイルに分割されたものの両方がおいてあるはずです。ここでは、初心者向けに、アーカイブされたものをダウンロードすることにしましょう。

また、バージョンは結構頻繁に更新されますので、なるべく最新のものをとってくるようにします。

以後はWindows版、Squeak2.2をダウンロードしたと仮定して話を進めます。

Prev. Index Next



［Happy Squeaking!!］

2. Squeak の入手

2. 2 Squeak のダウンロード

2. 2. 2 アーカイブファイルの展開

Windows版の、アーカイブファイルはSqueak.zipとなっていました。
これをWinZipなどで展開すると、以下のようなファイルができます。

- Readme.txt
- Squeak2.2.image
- Squeak2.2.changes
- SqueakBeta.exe
- SqueakV2.sources

全てを同じディレクトリにおいてください。
仮にここでは c:\lang\Smalltalk\Squeak に展開したとします。
ではさっそくですが、起動してみましょう。

Prev. Index Next

オブジェクトの広場

［Happy Squeaking!!］

2. Squeak の入手

2. 2 Squeak のダウンロード

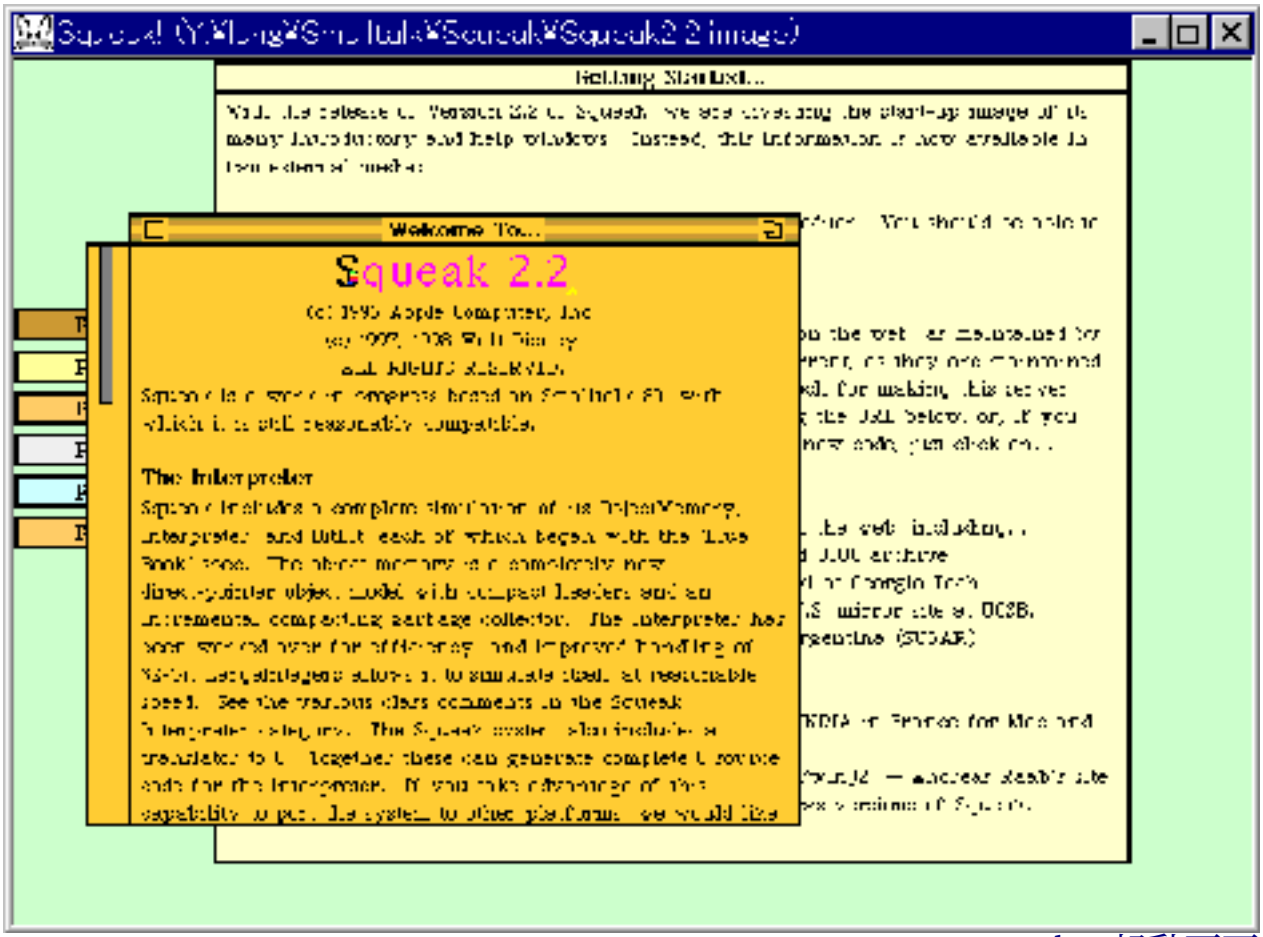
2. 2. 3 Squeak の起動

Windowsならば、エキスプローラで、Squeak2.2.image をSqueakBeta.exe にDrag&Dropします。
Unixユーザ、またはコマンドラインの好きなかたはおもむろに展開されたディレクトリに移動して、

```
C:/> cd C:/lang/Smalltalk/Squeak
C:/lang/Smalltalk/Squeak> SqueakBeta.exe Squeak2.2.image
```

などとします。

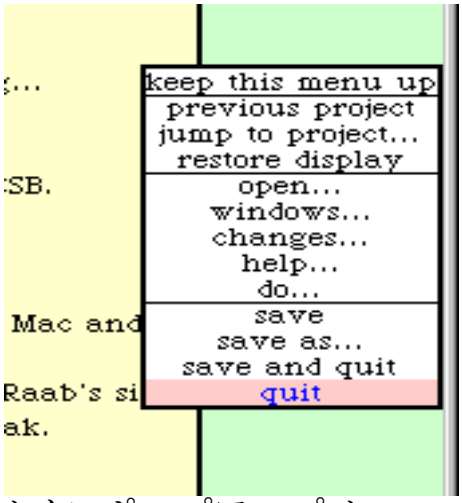
さて、こんな画面が立ち上がってきましたね。



Squeakの起動画面

なかなかカラフルです。ちょっとレトロですがいい味をだしています。

終了はバックグラウンド(黄緑の背景)をマウスでクリックするとポップアップメニューが出てきますのでquitを選んで終了します。Saveするか聞いてきますがnoで終了してください。



メインポップアップメニュー

オブジェクトの広場

［Happy Squeaking!!］

2. Squeak の入手

2. 2 Squeak のダウンロード

2. 2. 4 各ファイルの説明

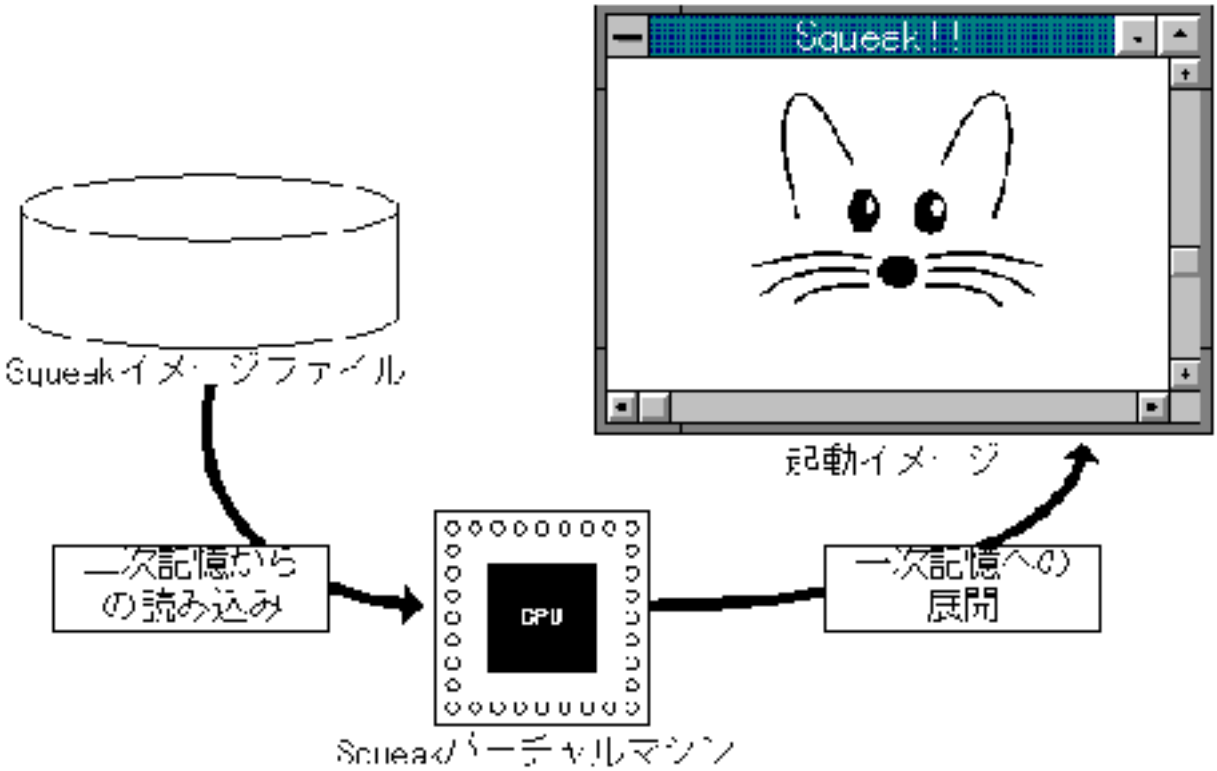
バーチャルマシン、イメージファイル

今の操作でお気づきの方もおられると思いますが、SqueakBeta.exeは、Squeakを動かすためのバーチャルマシンです。Javaでいうとjava.exe に該当します。
(VMにBetaとついているのは、御愛敬です。実はSqueakのVMはインタープリタ版とジャストインタイムコンパイラ版があったのですが、2.2からは一つになっています。このような事情からBetaなのです)。

そして、SqueakBeta.exe に引数として与えたSqueak2.2.imageは、イメージファイルと呼ばれます。

ここでのイメージは画像ではなく、「生き写し、そのもの」といった意味です。
Squeak2.2.imageには、Squekのバーチャルマシン上で動作するオブジェクト群が一体となってバイナリ形式で格納されています。
ノートPCではよくサスペンド機能というものがあります。これは一次記憶上(RAM)のメモリイメージの内容をそのまま二次記憶(ハードディスク)に移し替え、PCを落としてしまうものです。
リジュームにより、二次記憶上のメモリ内容は再びRAM上に読み込まれ、電源を落とす前の状態でPCのデスクトップがよみがえってきます。
これと同じことがSqueakでも行われています。Squeak2.2.imageは、二次記憶上にフリーズドライされており、SqueakVMに引数として与えられることにより再びRAM上に展開されて動き始めるのです。

図で示すと以下のようになります。



Squeakのバーチャルマシンとイメージファイルの関係

Javaでは起動時に一つのクラスファイルを指定します。Java仮想マシンは、そのファイルのmainメソッドから参照される全てのオブジェクト群の生成に必要なクラスファイルを、CLASSPATH環境変数に設定されたパスの中から検索し、起動時にオブジェクト群の固まりを作り上げます。これに対し、Smalltalkでは、最初からオブジェクト群は一つの固まりのファイルになっているのです。
(商用Smalltalkでは、起動時に複数のファイルから、起動に必要なオブジェクトをロードするものもありますが、Squeakではこのような方式はサポートされていません)。

設定ファイル

さて、終了するとSqueak.iniというファイルができています。ここにはSqueakBeta.exeを起動するときの設定情報が書きこまれます。

```
[Global]
DeferUpdate=1
ShowConsole=0
DynamicConsole=1
ReduceCPUUsage=1
3ButtonMouse=0
```

これはただの設定ファイルです。それぞれについては細かいのでここでは説明しません。デフォルト値で十分です。ただし3つボタンマウスを使われているかたは最後の行を

```
3ButtonMouse=1
```

というふうに変更しておいてください。

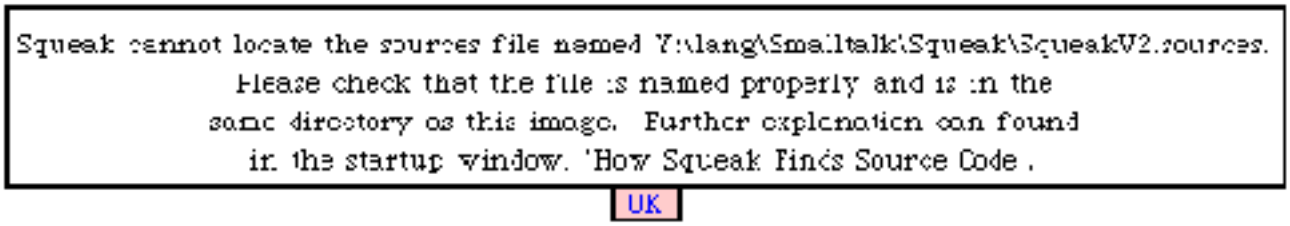
ソースファイル、チェンジファイル

次にSqueak2V.sourcesを見てみましょう。

これはSqueakの開発環境が参照するSmalltalkのソースコードが格納されたテキストファイルです。

```
'From Squeak 2.0 of May 22, 1998 on 22 May 1998 at
4:32:15 pm'!Object subclass: #AbstractScoreEvent
instanceVariableNames: 'time ' classVariableNames: ''
poolDictionaries: '' category: 'Music-Scores'!!AbstractScoreEvent
commentStamp: 'di 5/22/1998 16:32' prior: 0!Abstract
class for timed events in a MIDI
score.!!AbstractScoreEvent methodsFor: 'all' stamp:
'jm 12/31/97 11:46'!isNoteEvent ^ false! !!AbstractScoreEvent
methodsFor: 'all' stamp: 'jm
```

といった形で延々とコードが続いています。これはSqueakの開発環境(システムブラウザ)から参照されるもので、通常このファイルをエディタなどで見ることはありません。
システムブラウザの詳しい説明は後の章で行います。このファイルがないとSqueakの起動時にソースが参照できないという警告がでますので必ずこのディレクトリにおいておきましょう。



ソースファイルが見つからないという警告メッセージ

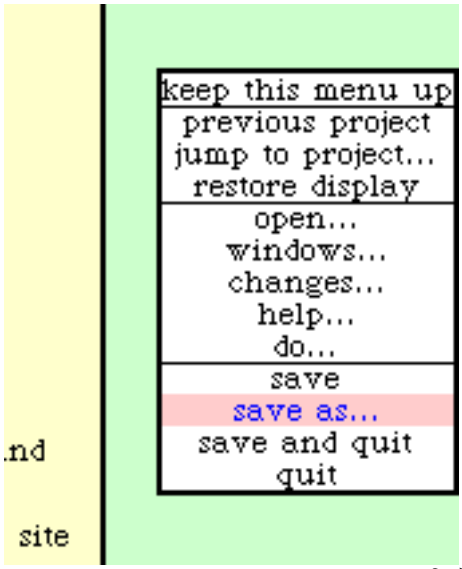
さて残るはチェンジファイルです(Squeak2.2.changes)。これは開発者がソースコードに対して行った変更、追加、削除などの記録がすべて保存されていく一種のジャーナル(ログ)ファイルです。ソースファイルと同じようにテキストファイルですが、ソースファイルがリードオンリーなのに比べチェンジファイルはどんどん書き込まれていく点が異なります。

ソースファイルには、安定したバージョンのソースが書き込まれており、チェンジファイルは、それらのソースに対する変更分が書き込まれます。

通常、チェンジファイルはイメージファイルとセットで個人ごとに所有されます。例えばumezawa用の開発環境としては、umezawa.imageとumezawa.changesがセットになります。名前も拡張子以外は同じになります。

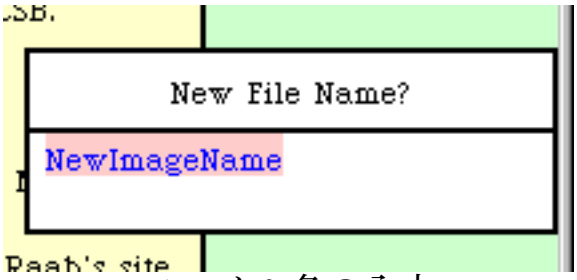
では、そろそろ自分用の環境を作りましょう。もう一度Squeakを起動してみてください。

起動したらメインメニューから、今度は save as… を選択してください。



メインメニューからsave as… を選択

すると新たなイメージファイル/チェンジファイルの名前を聞かれますので、好きな名前を入れます。



ファイル名の入力

これであなた専用のSqueakの環境が手に入りました。

Squeakを起動したディレクトリに、umezawa.image、umezawa.changesといった二つのファイルができているはずです。

今後の起動は、新しくできたイメージファイルを使って行っていきます。
以下のようなバッチファイルを作っておくと便利でしょう。
(イメージファイル、チェンジファイルは好きなディレクトリにおいてかまいません。下の例の場合は、c:\Home\umezawa\Squeak\image にumezawa.image とumezawa.changesを移動したと仮定しています)。

```
ファイル名 MySqueak.bat
@echo off
set SQVM=c:/lang/Smalltalk/Squeak/SqueakBeta.exe
set MYVI=c:/Home/umezawa/Squeak/image/umezawa.image
%SQVM% %MYVI%
```

以後は MySqueak とたたけばOKになります。

さあ、これで準備が整いました。Squeakの世界にでかけましょう!

オブジェクトの広場

［Happy Squeaking!!］

3. Squeak に親しむ

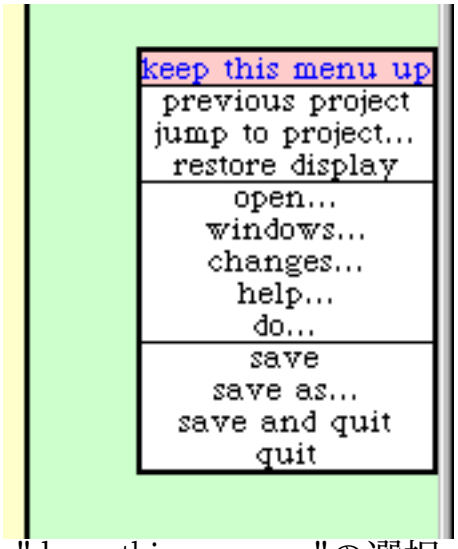
3. 1 インタラクティブな環境に慣れる

Squeakには多くのデモプログラムがついています。Squeakの環境に慣れるために幾つかを起動してみることにしましょう。

実行の窓 - ワークスペース を開く

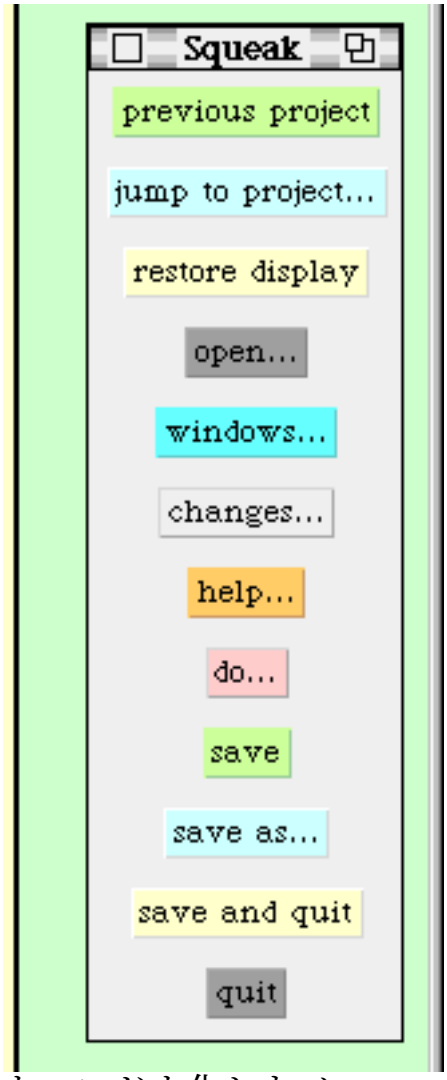
デモの起動には、実行プログラムを書くためのウィンドウが必要です。Smalltalkでは、この実行用のウィンドウのことをワークスペース(作業場)とよんでいます。Squeakを起動して、メインのポップアップメニューを出します。

以後、このメインメニューはよく使うので、"keep this menu up"を選択してメニューを出したままにすると便利でしょう。



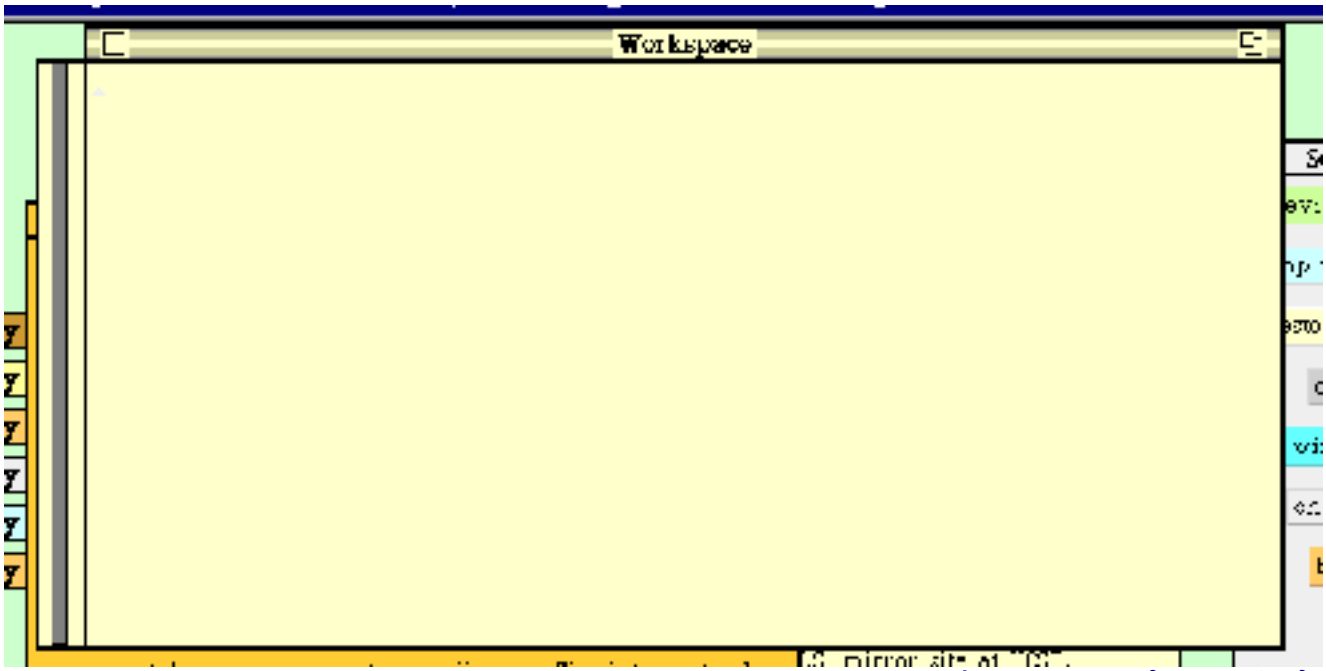
" keep this menu up"の選択

この操作によりポップアップメニューがウィンドウ化します。



ウィンドウ化したメニュー

メインメニューからopen… さらに workspace(上から二番目)を選ぶと以下のようなウィンドウが立ち上がります。

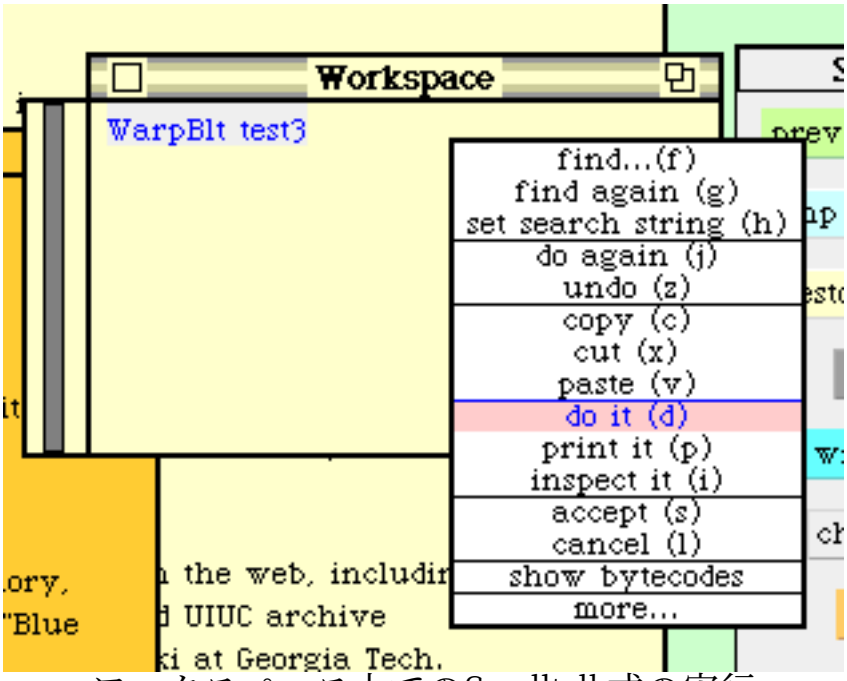


ワークスペースウィンドウ

ワークスペースでは任意のSmalltalkの式を書いて、それを即座に実行できます。

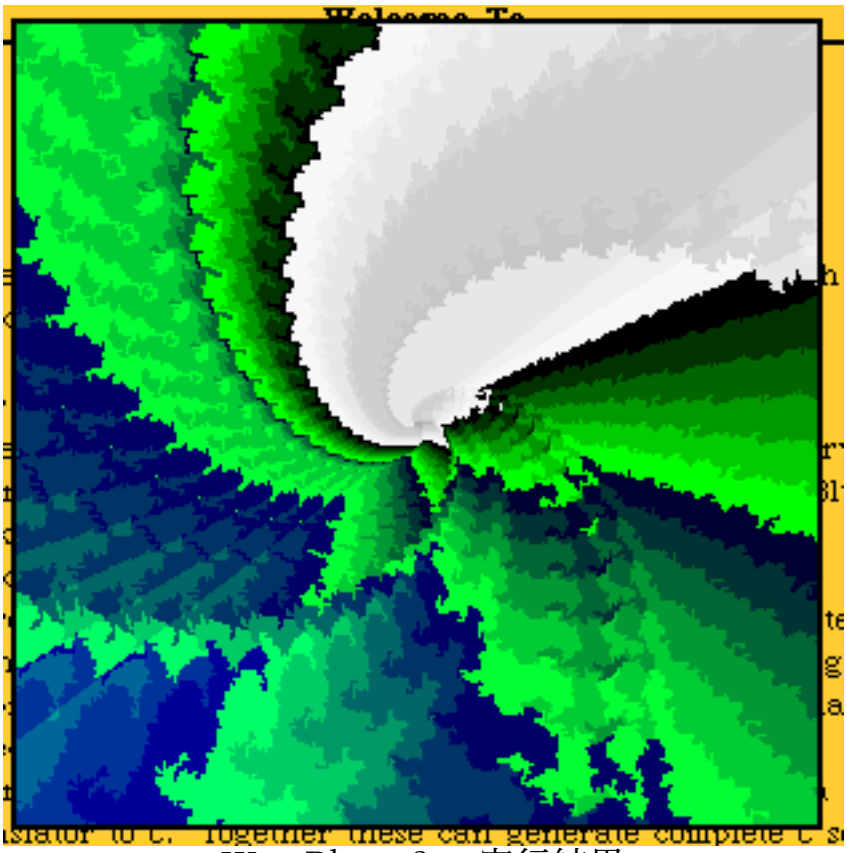
まずは、デモプログラムのうち、WarpBltのデモを起動してみることにします。WarpBltとは、Squeak持つビットマープデータ処理機能です。

ワークスペースに、WarpBlt test3 と打ち込んでください。
そしてその文をすべて選択して、右クリックで "do it"を選びます。



ワークスペース上でのSmalltalk式の実行

すると、、、、!!
万華鏡のような華麗な映像を写すウィンドウが立ち上がります。(実際にはアニメーションしています。ここでお見せできないのが残念)。


WarpBlt test3 の実行結果

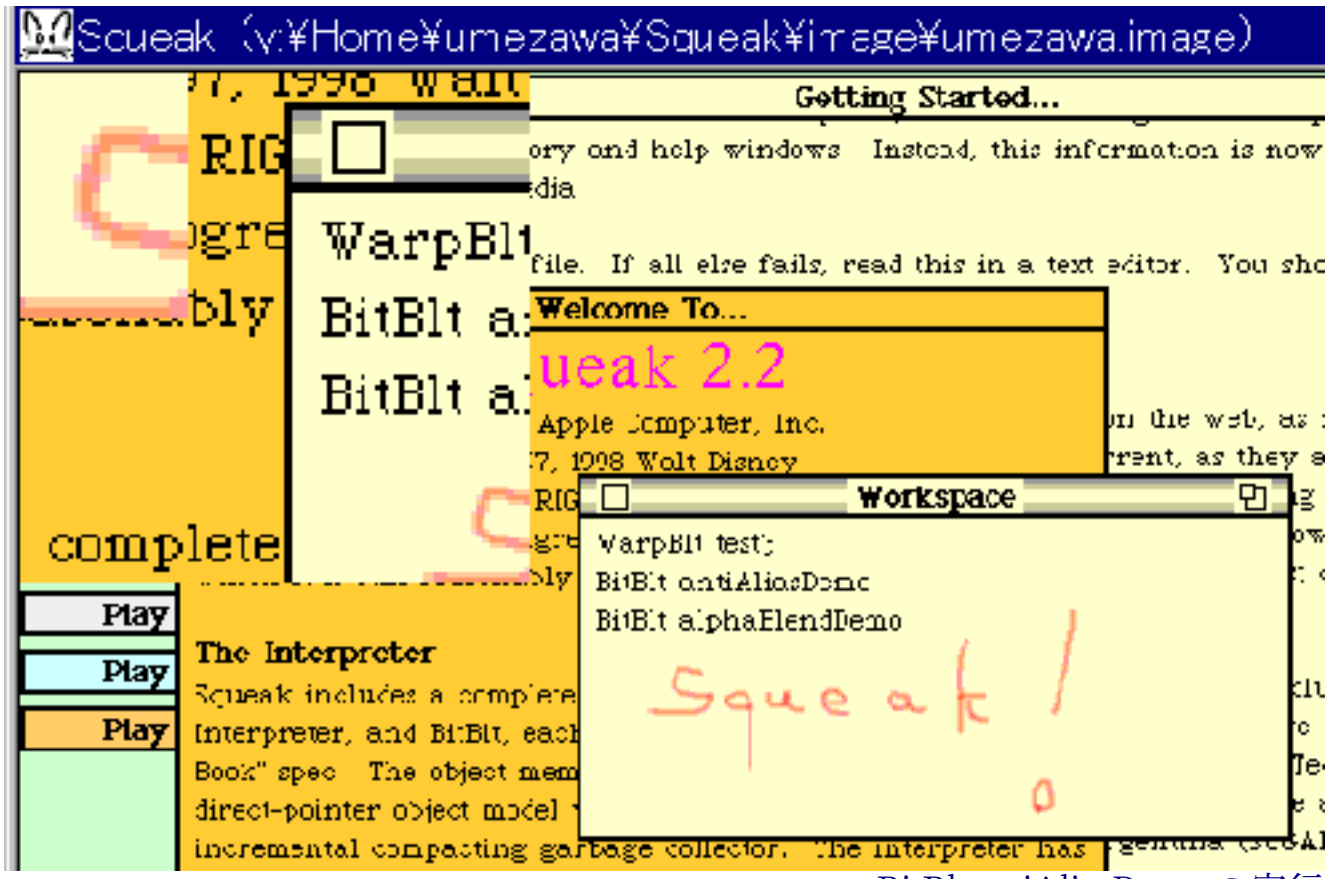
ある程度楽しんだら、メインメニューの"restore display"を選択して画面をもとにもどしておきましょう。

Workspaceでは、このように、Smalltalkの式を書いて選択してdo itする操作が基本になります。

C++やJavaと違い、コンパイルの手間をかけずに即座に実行できるところが、なんとも気軽です。

ワークスペースでの実行をもう少し行ってみましょう。

今度は BitBlt antiAliasDemo と打ち込んでdo it してみてください。
(BitBltも同様にビットマップ処理を行います)。
以下のように、左上がカーソル周辺の拡大になり、またマウスの左ドラッグで線が引けるようになります。



BitBlt antiAliasDemo の実行

終了は、右クリックです。前と同じように"resotore display"を選択して画面をもとにもどしておきます。

まだもの足りない方は BitBlt alphaBlendDemo を実行してみてください。(画面イメージは省略します).

更につづく

Prev. Index Next

オブジェクトの広場

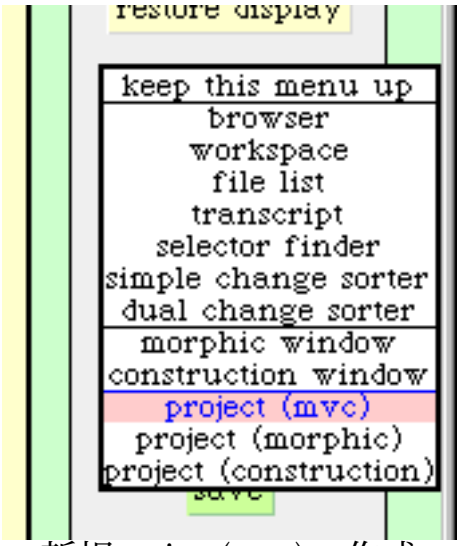
［Happy Squeaking!!］

3. Squeak に親しむ

3. 2 プロジェクトの作成

以後は、オブジェクト指向の概念を学びながら、実際のプログラミングを行っていきます。今後の作業を考えて、プロジェクトを作成しておいたほうがいいでしょう。
プロジェクトとは、個人が行う環境を作業の種類単位で分割したものと考えられます。先ほどイメージファイルを作成し、個人専用の環境を作りましたが、それをさらに、これから行う作業の性質により細分化できるのです。(例えばアプリケーションAを開発するプロジェクト、サンプルプログラムBをテストするプロジェクトBなどで分けることができます)。
現在開いているSqueakの画面には多くのデモ用のウィンドウなどがアイコン化されて置かれています。このままでは自分が開いているワークスペースなどのウィンドウと区別が付かなくなり、煩雑になります。そこでこれから、「オブジェクト指向基本学習」用のプロジェクトを作り、以後はその中で作業を行うことにします。

メインメニューから、"open…"、さらに"project (mvc)"を選択してください。



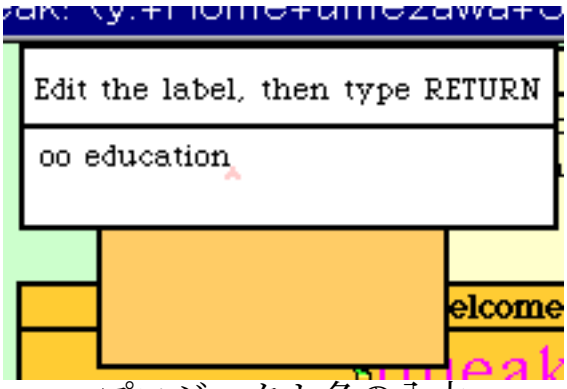
新規project(mvc)の作成

”project(mvc)"のMVCとは、Smalltalkに古くから使われているGUIフレームワークです。MVCはもう少しこの講座が進んだ段階で詳しく説明します。とりあえずは一番ノーマルなプロジェクトを作成していると考えてください。

さて、選択すると以下のような小さなウィンドウが立ち上がります。タイトルがない(Untitled)のでつけてあげましょう。

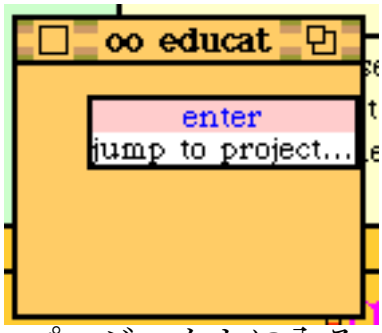
プロジェクトウィンドウの起動

タイトルバーをクリックすると入力ダイアログが出てきます。この名前を"oo education"に変えましょう。これがプロジェクト名になります。



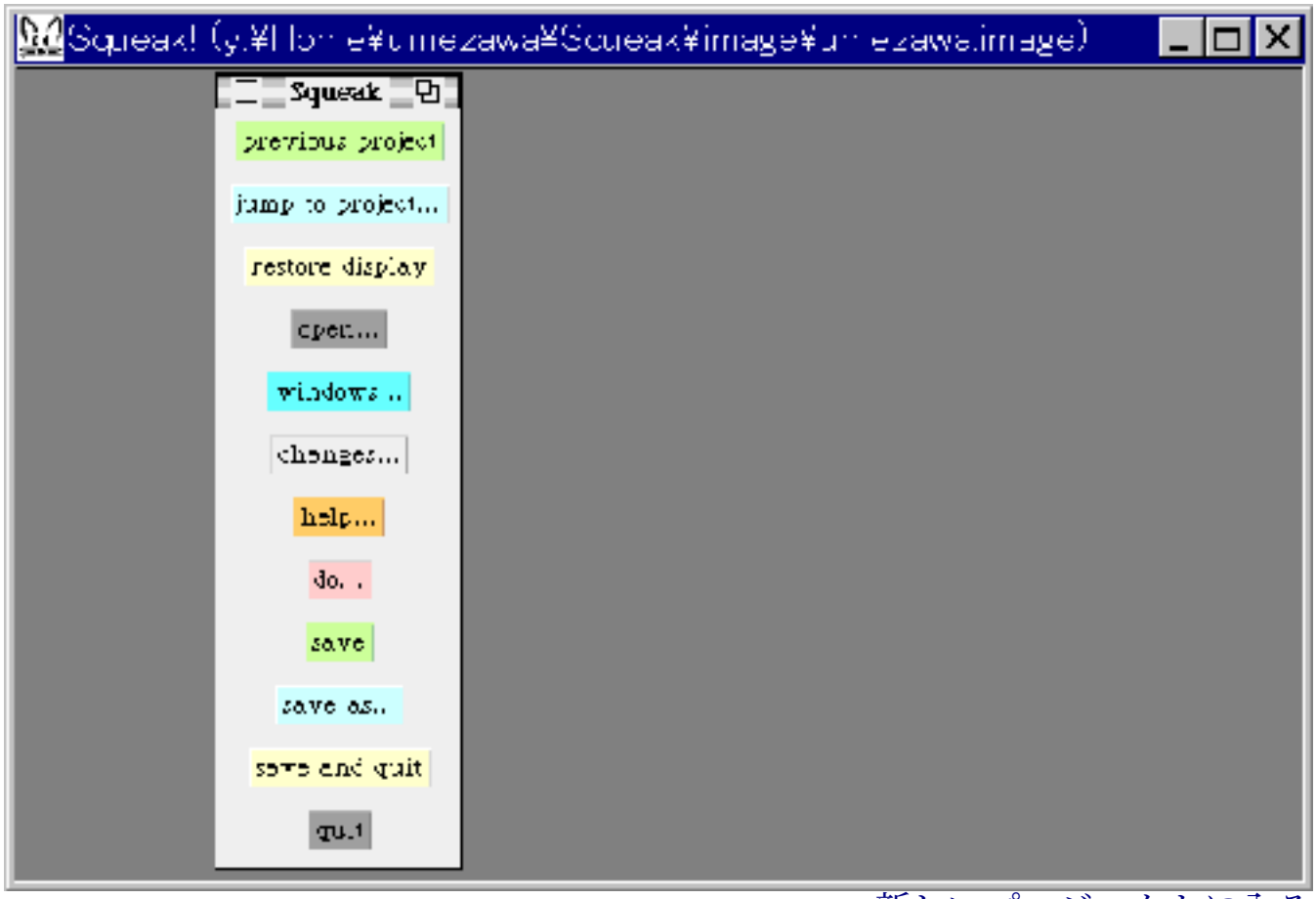
プロジェクト名の入力

さて、新しくできたプロジェクトに入ることにしましょう。左クリックでポップアップメニューが出ますので"enter"です。



プロジェクトに入る

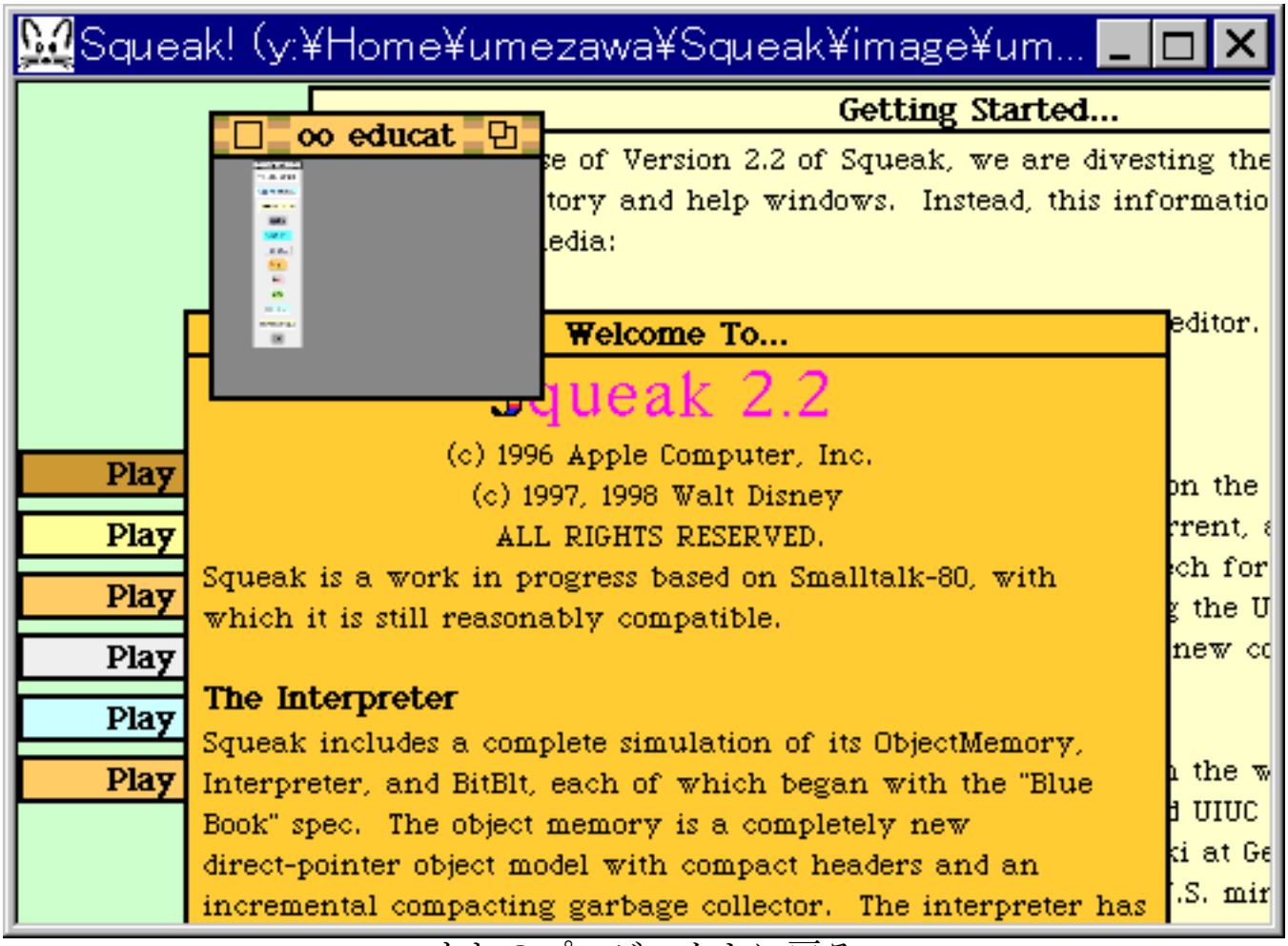
プロジェクトに入ると真っ暗なので驚かれたかもしれません。単にウィンドウが一つも表示されていないだけです。メインメニューのウィンドウでも立ち上げておきましょう。



新しいプロジェクトに入る

このプロジェクト内で、先ほど紹介した、ワークスペース、トランスクリプト、システムブラウザなどを使って、プログラミングしていくことになります。

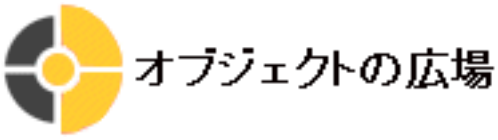
もとのプロジェクトに戻るには、メインメニューの"previous project"を選択します。プロジェクトウィンドウには小さく先ほど作成したプロジェクトの様子が再現されています。



もとのプロジェクトに戻る

以後プロジェクトの移動は、先ほどのようにプロジェクトウィンドウの"enter"を使って行えますし、また、メインメニューの"jump to project…"でいくこともできます。("jump to project…"の選択では、デモ用のプロジェクトが幾つかリストアップされています。興味のあるかたはenterしてみてください)。

Prev. Index Next



［Happy Squeaking!!］

3. Squeak に親しむ

3. 3 Squeakの環境とは？

これでオブジェクト指向の基本概念、更にSmalltalkの学習を進めていく基本的な準備は整いました。

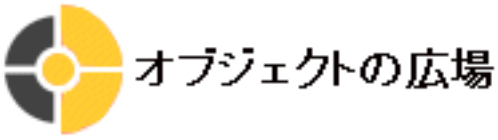
今まで環境について主にみてきましたが、Squeakの環境自体もSmalltalkにより実現されています。ですからいつかSmalltalkに詳しくなったときには、この環境自体をどんどん自分の好きな姿に発展させていくことが可能です。
Smalltalkは言語というよりは一種の有機体という人がいます。自らの姿を自ら変容させる力を持つという意味で確かに生物を連想させます。
慣れない環境で多少戸惑ったと思いますが、Smalltalkは使いこなせれば非常にパワーのある言語なのです。

ただしそのようなSuper Smalltalkerの段階に至るには、まずオブジェクト指向の基本概念、Smalltalkの文法の知識を身につけなければなりません。ものごとには順序というものがありますから…

以後はオブジェクト指向の学習に入っていきましょう!

と、いいたいところですが、紙面の都合で今回はここまで。次回のお楽しみです。

Prev. Index Next



[Happy Squeaking!!]

4. 参考文献

連載の待ちきれないかたは以下の文献による予習をお勧めします。

● 和書

「サクサクSmalltalk」東京電機大学出版 1996

商用のSmalltalkであるVisualWorksで解説を行っている入門書。"The Art and Science of Smalltalk" Prentice Hall 1995 の翻訳 。MVCやアダプタに ついての解説もある。

「Smalltalkイディオム」 ソフトリサーチセンター 1997

日本のSmallalkerとしては第一人者である青木淳さんの書き下ろし。スレッドや例外処理などアドバンスな内容を含む。

● 洋書

"Smalltalk, Objects, and Design" Prentice Hall 1996

Smalltalkを学びながら、オブジェクト指向、デザインパターンについて学 んでいく。本講座のスタンスに最も近い。入門書でありながら洞察は深い。

"On to Smalltalk" Addison Wesley 1998

日本でも「ウィンストンの..」で多くの訳が出ているPatrick Henry Winston
によるSmalltalkの入門書。クックブックスタイルで簡潔に説明がされており実用的。日本語訳も近いうちに出版される。

Let's "do it"!!

Prev. Index