

Chapter 2 Javaのプログラムの作成、実行の仕方

2.1 Javaの開発環境とファイル

■Javaの開発環境について

使用しているオペレーティングシステムやコンピュータによって、同じJava言語でプログラムを開発すると言ってもまったく異なると思います。ここでは、プログラムを開発するためのソフトウェア群を指して開発環境（Development Environment）と呼びます。この章では、個々の開発環境に依存しない最低限の共通事項をまとめてみました。

代表的な開発環境については、付録にそのいくつかを掲載いたしました。そちらを参照してください。ただし、開発環境は基本的には、次のような2つの分類ができます。

★標準型の開発環境

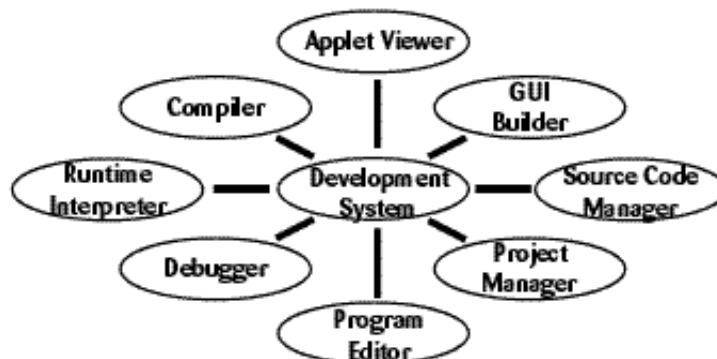
この開発環境では、基本的にはコンパイラとランタイム・インタプリタとアプレットの表示のために必要なアプレットビューワ（Applet Viewer）から構成されています。プログラムを記述するためには、別途何らかのエディタを自分で用意する必要があります。また、この開発環境はネットワークからダウンロードしたりして無償で手に入れる（ユーザ登録を必要とするものもあります）ことができるものが多いのです。この開発環境の代表的なものとしては、標準となっているSun社のJDK（Java Development Kit）や、Apple社のMRJ SDK（Macintosh Runtime for Java Software Development Kit）などがあります。



図2-1 標準型の開発環境

★統合型の開発環境

上記の開発用ソフトウェアの他に、エディタや複数のプログラムを管理するツール（注1）が付属しています。プログラムの実行を途中で止めたり、その途中の状態を見ることができるデバッガ（Debugger）と呼ばれるツールも付属しています。また、グラフィックベースでソフトウェア開発を行なえるRAD（Rapid Application Development）ツールを付属していたり、むしろそちらを主体にしているものがあります。総合的な開発環境になっていますので、ボタンやメニューで簡単にコンパイルしたり、実行させたりすることができます。この開発環境は、有償の製品と供給されるものがほとんどです。代表的なものとしては、Metrowerks社のCodeWarriorや、Microsoft社のVisual J++あるいは、RADを主体としたSymantec社のVisual Cafe、Borland社のJBuilderなどがあります。



注1 開発のためのソフトウェアをツール（Tool）と呼ぶことがあります。

■Javaのプログラムファイル

前章で述べましたように、実行するまでは一つのプログラムに対して、幾つものプログラムファイルが生成されます。これらに関して一度整理しておきましょう。

★ソースファイル（ソーステキストファイル／ソースプログラムファイル）

Javaプログラミング言語で記述されたプログラムファイルです。これは自分で作成する必要があります。ファイル名に関しては、次のような約束事があります。

.javaというファイル名になっていなければいけません。の部分は、適当な名前を付けても構いませんが、ただし少し制限もあります（後で説明します）。また、オペレーティングシステムによっては、付けられる名前に制限があります。

例えば、Sample.javaという名前のファイルは、ソースプログラムファイルです。



Sample.java

★クラスファイル

中間コードに翻訳されたファイルです。これは、コンパイラによって生成されます。Javaでは、実行の単位がクラス（後の章で説明します）と呼ばれるものになっていますので、このような名前がついています。このファイル単体では実行することはできません。実行にはランタイム・インタープリタを必要とします。アプレットなどがネットワークを介して転送されるときは、このファイルが転送されます。この種類のファイルも名前に関して約束があります。

***.classというファイル名という名前になっていなければいけません。

例えば、Sample.classという名前のファイルは、クラスファイルです。



Sample.class

★パッケージファイル（クラスライブラリィ）

クラスファイルが幾つか合わさったものになっています（注1）。これは、人目につかないところに保存されていて自動的にコンパイラやランタイム・ライブラリが利用するファイルですので、今後特に必要でない限り、本書では触れないことにします。



QT4Java.jar

★プロジェクトファイル

総合型の開発環境では用意されているもので、複数のソーステキストからプログラムが構成されていたり、実行に必要なとされている画像・音声ファイルまでまとめて管理してくれるものです。



注1 もともと複数のファイルが一つにまとめられたファイルをアーカイブ（Archive）ファイルと呼ぶことがあります。

2.2 アプリケーションプログラムの作成・実行の仕方

■Java 言語で書かれたプログラムが実行されるまで

プログラムを作って実行するまでは、次の3つの過程が必要であることは、これまでの説明からおわかりのことでしょう。

1. プログラムを編集（作成）する
2. プログラムをコンパイルする
3. 実行する

編集・作成の結果として、ソースファイルが作成されますし、コンパイルの結果としてクラスファイルが自動的に作成されます。各過程について以下にもう少し詳しく説明しましょう。ただし、特定の開発環境に依存する部分もありますので、付録を参照するなり、マニュアルを読むなりして下さい。また、アプリケーションとアプレットの場合では、少しやり方が異なりますので、アプレットは次節で別に説明します。

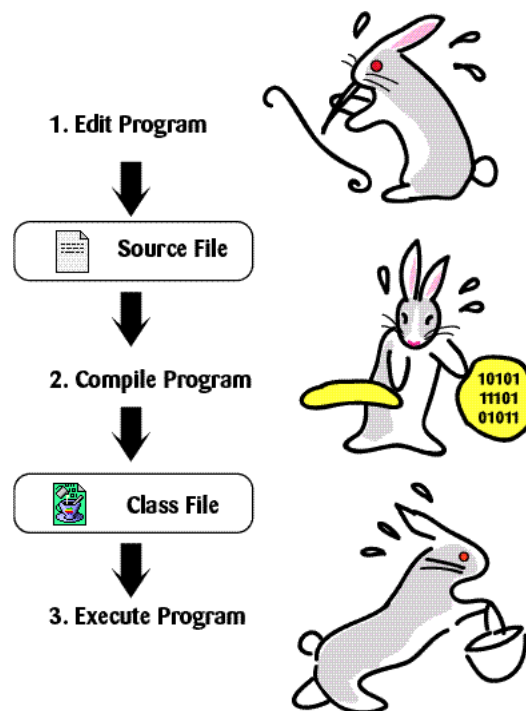


図2-3 プログラムを作って実行するまで

なお、総合的な開発環境の場合は、プログラムを作成する前に、プロジェクトを新規に作成する必要があるものが多いようです。この場合は、マニュアル等に従って新規のプロジェクトを一つ作成してください。

■Java 言語プログラムの作成の仕方

ワープロを使ったことがある人であれば、通常の文書を書くのと同様で、次のような手順となります。なお、総合型の開発環境では、「新規ファイル」等のメニューを選択すれば、自動的にエディタが起動されます。

1. 統合型の開発環境では新規のプロジェクトを作成する
2. エディタを起動し、新規に文書を作成するか既にある文書を編集する
3. エディタの中でJava言語のプログラムを入力・編集していく
4. プログラムを記述終了後、ファイルに保存する

プログラムを保存した後、必ずしもエディタを終了する必要はありません。前に述べたように、ファイルを終了する際に、Java言語のプログラムでは、ファイル名が必ず「.java」で終わっていないことに注意してください。ファイル名の終わりの接辞（ピリオド以降）をサフィックス（Suffix）と呼ぶことがあります。なお、ファイル名はサフィックスの前の部分はクラス名（後で述べます）と同じにしなければならないという制約があります。

■Java言語プログラムのコンパイルの仕方

プログラムを中間コードに変換するために、コンパイルします。これも開発環境によって様々ですが、次のいずれかに大別されます。

- ★標準型の開発環境では、コンパイラを起動して、コンパイルするファイルを指定します。
- ★総合型の開発環境では、「コンパイル」や「ビルド」、あるいは「make」といったメニューを選択します。

標準型の開発環境でも、例えばMRJ SDKの場合にはコンパイラのアイコン上にプログラムのファイルのアイコンをドラッグして離してやる（ドラッグ&ドロップ）すると、それだけでコンパイラが起動し、コンパイルを行ってくれるというわかりやすいやり方もあります。

プログラムの記述の仕方に問題がなかった場合は、接尾詞の.javaが、.classに変わった形の名前のファイルが自動的に作成されます。成功した場合は、往々にして何も表示されないことが多いため、本当に成功したかどうか疑問に思うことがあります。逆に、プログラムの記述に間違いがあったときは、間違った場所がエラー（Error）として表示されますので、失敗したことがわかんと思います。エラーについては別節で説明します。

■Java言語アプリケーションプログラムの実行の仕方

めでたくもコンパイルに成功した場合について説明しましょう。アプリケーションとして記述されたプログラムをコンパイルした結果として生成されたクラスファイルは、そのままランタイム・インタプリタを使って実行することができます。

- ★標準型の開発環境では、ランタイム・インタプリタを起動して、クラスファイルを指定して実行します。
- ★総合型の開発環境では、「実行」あるいは「Run」といったメニューを選択します。

総合型の開発環境では、標準でデバッガが起動される設定になっているものもあります。そのような環境では、メニューに「デバッグ」あるいは「Debug」といった項目が表示されています。

■サンプルのアプリケーションを作る

実際に、Java言語で書かれたアプリケーション用のプログラムを作成して実行してみましょう。ここでは、例として標準とされているMetrowerks社のCodeWarriorという開発環境を基本的に用います。この開発環境では、エディタ、コンパイラやランタイム・インタプリタなどは、すべて連動する形で設計されているため、メニューやダイアログを選ぶだけで簡単に起動することができます。

1. CodeWarrior IDEを起動してください。ここで、「ファイル」メニューから「新規」あるいは「New」項目を選択して、アプリケーション用のプロジェクトを新しく作る必要があります。次のようなダイアログが表示されます。

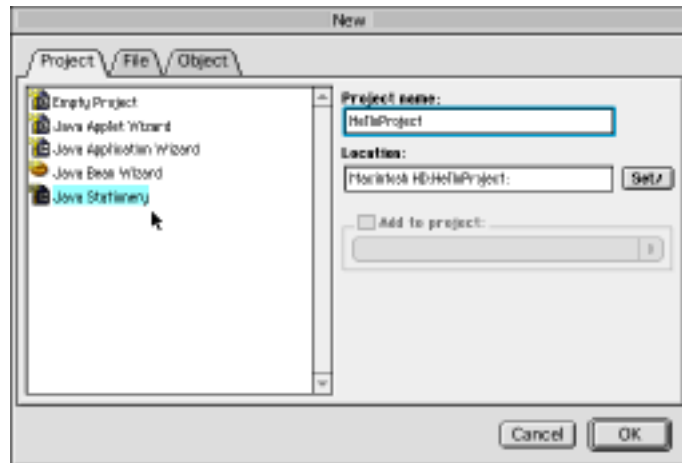


図2-4 新規にプロジェクトを作るためのダイアログ

ダイアログが表示されたら、図のように、Java Stationery (ステーションナリー) を選択します。そして、プロジェクト名を入力します。プロジェクト名は、「HelloProject」でよいでしょう。プロジェクトに対して、1つのフォルダが作成されます。必要であれば、プロジェクトのフォルダをどこに作るか調整してください。ステーションナリーを選びOKボタンを押すと、どのようなJavaのプロジェクトを作るか尋ねてきます。

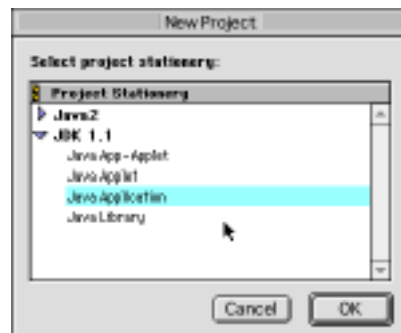


図2-5 プロジェクトの種類を尋ねるダイアログ

このダイアログでは、Java2とJDK1.1が表示されています。そのどちらでも構いませんから、右向きの三角をクリックして、下向きにしてください。4つの選択肢が現れてきます。今回はアプリケーションを作成しますので、「Java Application」を選択して、OKボタンを押します。これで、プロジェクトの指定はできました。そうするとプロジェクトウィンドウが現れます。

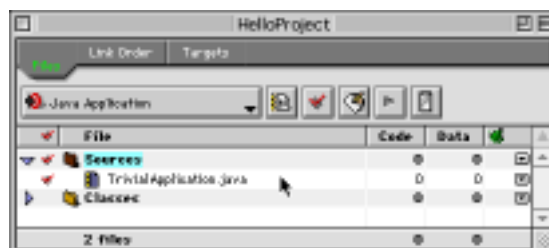


図2-6 作られたプロジェクトウィンドウ

このプロジェクトウィンドウで、プログラムのソーステキストファイルやクラスライブラリを管理しています。このウィンドウはプロジェクトに1つだけ作られます。「Sources」の項目の左にある右向き三角をクリックして、下向きにしてください。仮に作られたソーステキストファイルとして、「TrivialApplication.java」という項目がある筈です。

2. そのプロジェクトに含まれるソーステキストファイルを編集します。

プロジェクトウィンドウにある「TrivialApplication.java」をダブルクリックしてください。自動的に、エディタが起動して、その中味を表示してくれます。



図2-7 TrivialApplication.java ファイルの中味を表示したエディタ

ここで、新たにプログラムを入力するか、既にステーションナリーで作ってくれたプログラムを修正するかします。どちらの場合もプログラムを記述するときは、大文字小文字の区別をしてください。{ } の波括弧（英語では brace と呼ばれています）も正確に打ち込んでください。本書では、わかりやすくするために文字が太字になっていたり斜体になっていますが、大抵のエディタではそんな機能はありませんので、気にせずに入力してください。空白があるところとないところの区別もしてください。また、2行目から4行目に掛けて行なわれている左側にテキストをずらす記述の仕方は字下げ（Indent）と呼ばれていますが、これでもしてみてください。字下げにはTABキーを用います。

```
public class Hello {  
    public static void main( String [ ] args ) {  
        System.out.println("Hello, World!");  
    }  
}
```

3. エディタで上記のプログラムをファイルに保存します。「ファイル」メニューの「Save As」、あるいは「別名で保存」やショートカットボタンなどを利用して保存してください。ファイル名は、「Hello.java」とします。ファイル名のHも大文字にしてください。ファイルは、プロジェクトのフォルダを選んでその中に保存してください。保存されれば、自動的にプロジェクトウィンドウのソースファイルのファイル名もHello.javaに更新されている筈です。

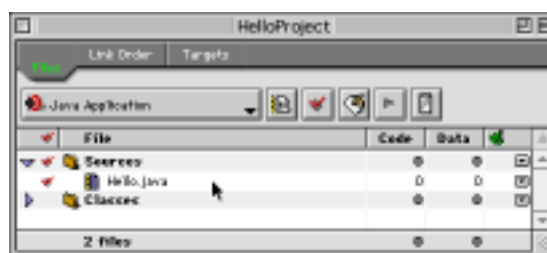


図2-8 ファイル名が更新されたプロジェクトウィンドウ

4. コンパイルをしてみましょう。「プロジェクト」メニューの中から、「Compile」あるいは「コンパイル」というメニュー項目を選んでください。「ビルド（Make）」という項目でも構いません。「ビルド」の方を選んだ場合は、コンパイルする以外に、クラスライブラリとのリンクも行ない、クラスファイルではなく、実行環境の設定を含むアプリケーションファイルまで自動的に作成してくれます。

5. もし成功したら、コンパイル最中に出ていたダイアログが消えて、元の状態に戻ります。プロジェクトウィンドウから、チェックマークが消えています。これは、コンパイルが終わったことを示しています。何かエラーが出たら、エディタで編集し直しましょう。編集し直したら、保存するのを忘れずに。

6. ランタイム・インタープリタで実行してみましょう。統合型の開発環境であるCodeWarrior IDEでは、「プロジェクト」メニューから、「Run」あるいは「実行」というメニュー項目を選んでください。自動的にアプリケーションが起動される筈です。



図2-9 アプリケーションが起動されて実行されたウィンドウ

画面上に、Hello, World!と表示されましたでしょうか？あんなに苦労したのにこれだけ？と思うかも知れませんが、最初の一步ですので、これでよしとしましょう。

7. 実行が終わったら、アプリケーションのメニューから「Quit」や「Exit」あるいは「終了」という項目を選んで終了させます。このアプリケーションを終了させておかないと、これ以降同じプログラムに対して、何か修正をした場合に、もう一度「ビルド」することができなくなります。注意してください。

■プログラムの簡単な説明

ここでは、聞きなれないクラスやメソッドという名前を用いてますが、その詳細は次章以降に解説するとして、簡単にそれぞれの行が何をしているかだけ説明します。

1行目：**public class** Hello {

Helloという名前でクラスを定義しています。

2行目：**public static void** main(String [] args){

mainという名前の実行メソッドを定義しています。

3行目：System.out.println("Hello, World!");

システム（ここではオペレーティングシステムのことを指します）にHello, World!と表示して改行せよと命令しています。他にも何かメッセージを表示したければ、この行は、複数書くことができます。

4行目：}

メソッドの定義を終了させています。

5行目：}

クラスの定義を終了させています。

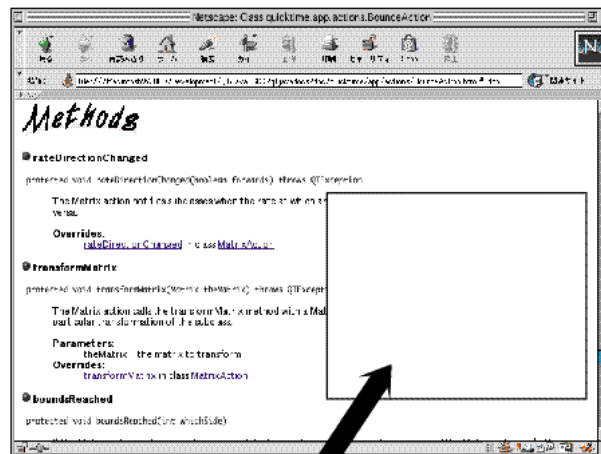
2.3 アプレットプログラムの作成、実行の仕方

■アプレットの実行について

アプレットは、前章で触れましたように、ネットワーク上を Web ページと共に転送されて、Web ブラウザ上で見る形態のプログラムです。そのため、プログラム以外にHTMLの文書が必要となります。

- HTMLの文書中でアプレットを使い、その描画領域について指定する
- アプレットのプログラムでは、描画領域内に情報を表示する

以上の2つの要素が満たされていしないと、アプレットが正しく動いているのかどうか確認することはできません。そのため、アプリケーションプログラムよりもアプレットの方が、若干実行までの手数が多くなります。



Applet Area

図2-10 HTML文書上のアプレットの描画

■アプレットのプログラムをJavaで作る

次の手順を使って、Webページから実行できる基本のアプレットを作成してみましょう。開発環境はアプリケーションのときと同じく、CodeWarrior IDEで動かすことを基本としています。

1. 統合型の開発環境の場合は、アプレット用のプロジェクトを新しく作る必要があります。「ファイル」メニューから「新規」あるいは「New」項目を選択して下さい。そして、「Java Stationery」を選択し、プロジェクト名として、「HelloAppletProject」と入力します。プロジェクト名は、適当に自分が覚えやすい名前が結構です。プロジェクトが作られるフォルダの場所を変えたければSetボタンを押して指定し直します。

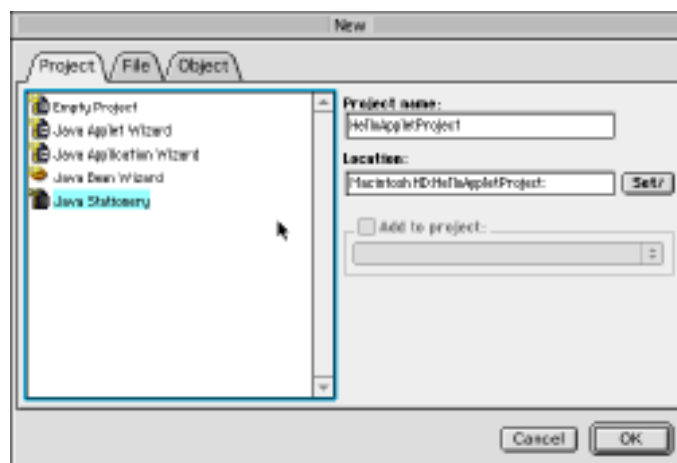


図2-11 プロジェクトの指定をするダイアログ

プロジェクトの種類を指定するダイアログが表示されたら、Java2でもJDK1.1のどちらでも構いませんから、その中の「Java Applet」を選びます。これで、アプレット作成用のプロジェクトが作成されます。

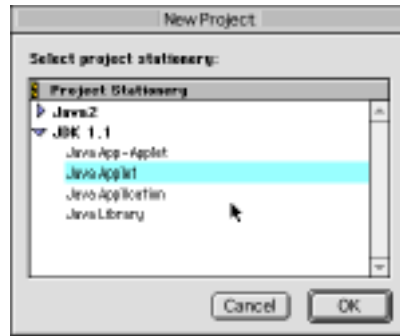


図2-12 プロジェクトの種類を選ぶダイアログ

指定がすべて終了しますと、プロジェクトウィンドウが表示されます。この中の「Sources」の項目の中には、JavaのプログラムファイルとHTMLファイルの2つのファイルが自動的に作成されています。

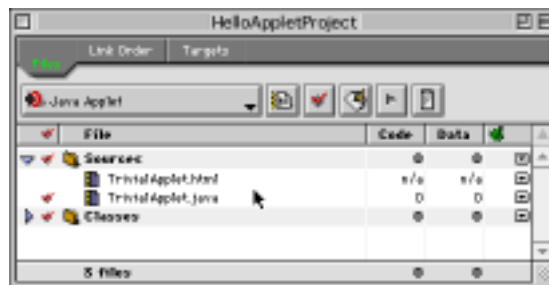


図2-13 作成されたプロジェクトウィンドウ

2. プロジェクトウィンドウに表示された「TrivialApplet.java」をダブルクリックして、プログラムのソーステキストファイルをエディタで編集します。



図2-14 起動されたエディタ

次のように修正してください。なお、以下のプログラムで `gc`と書かれている部分は、`g`のままでも構いません。また、`init`と書かれている一連の部分は、消さないでそのまま残しておいても構いません。最低限、クラス名の `HelloWeb`のところだけを直してください。

```
import java.awt.*;
import java.applet.*;

public class HelloWeb extends Applet {
    public void paint( Graphics gc ) {
        gc.drawString( "Hello, Window!", 50, 25 );
    }
}
```

- 修正したプログラムを保存します。「別名で保存」あるいは「Save As」を使って保存してください。ファイル名は、「HelloWeb.java」とします。自動的にプロジェクトウィンドウのファイル名も更新された筈です。
- 統合型の開発環境の場合は、メニュー項目の「コンパイル（Compile）」あるいは「ビルド（Make）」を選択して、コンパイルしてください。

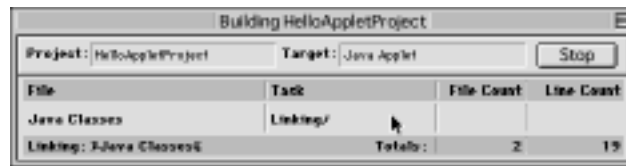


図2-15 ビルドをしている最中に表示されるダイアログ

- エラーがあったら、エディタ上で該当部分を修正して、2. からやり直しましょう。保存することを忘れずに。
- CodeWarriorでステーションナリーを利用してアプレットを作る場合は、大抵はプロジェクトに1つHTMLファイルが用意されていますので、そのファイルを編集してください（ちゃんと記述されている場合は編集しなくても構いません）。入力したテキストを保存します。なお、下記テキストの中で、code=の後のクラス名の部分だけを直してください。既に用意されているHTMLファイルでは、他にもいろいろな指定がありますが、それ以外のところは直す必要はありません。

```
<applet code=HelloWeb width=300 height=200></applet>
```

- 統合型の開発環境であるCodeWarrior IDEでは、「実行」のメニュー項目を選択すれば、自動的に予め設定してあるアプレットビューワが起動されます。



図2-16 アプレットビューワによって実行されたアプレット

- うまく表示されて、終了するには、メニューのExitあるいはQuitを選択してみてください。

第7過程で、実行の指定をしているのは後から作成したHTMLファイルであることに注意してください。クラスファイルが作成されますが、アプレットビューワには、HTMLファイルによって、code=HelloWebのところでクラスファイルを指定していますので、ここの記述を間違えないでください。システムによっては、「ファイルが見つからない」とエラーが表示されて動かないこともあります。その場合には、HTMLの記述を次のようにしてみてください。ファイル名を指定して、ダブルクォーテーションで囲みます。修正する前の元の記述もそうになっていると思います。

```
<applet code="HelloWeb.class" width=300 height=200></applet>
```

■プログラムの簡単な説明

1行目と2行目：**import** java.awt.*; **import** java.applet.*;

描画領域に描画するための機能が用意されているクラスライブラリであるawtパッケージと、アプレット用のクラスライブラリであるappletパッケージを利用することを宣言しています。

3行目：**public class** HelloWeb **extends** Applet {

アプレットであるようなクラスHelloWebを定義しています。

4行目：**public void** paint(Graphics gc) {

描画領域にアプレットが描画するときに実行されるpaintという名前のメソッドを定義しています。描画領域は、gcという名前で受け取ることにしています。

5行目：`gc.drawString( "Hello, Window!", 50, 25 );`

描画領域gcに対して、x座標が50、y座標が25の位置に「Hello, Window!」と表示しろと指定しています。

6行目：}

paintメソッドの定義を終了させています。

7行目：}

HelloWebクラスの定義を終了させています。

2.4 エラーとその対処

■エラーの種類

エラーは以下のようにいくつかのエラーに分類されます。

- ・構文エラー (Syntax Error) は、記述されたJava言語のプログラムが構文規則にあわないときにコンパイラによって発生します。
- ・実行時エラー (Runtime Error) は、プログラムを実行した段階で、起こるものです。例えば、プログラム中で指定したファイルが存在しなかったような場合に発生します。
- ・意味的エラー (Semantic Error) は、プログラムを設計する段階で内在するものです。例えば、プログラムが予想していなかった事態が発生して、プログラムが対処できなくなったりするような場合に発生します。

さて、ここでは、構文エラーについて単にエラーと呼ぶことにしましょう。プログラムに内在している実行時エラーや意味的エラーの要因は、コンピュータに虫が住み着いて、何か悪さをしているようにだと思われてきましたので、バグ (Bug) と呼ばれることが多いようです。このバグを潰すことは、構文エラーを修正するよりもはるかに頭を使う作業となります。コンパイラは、構文エラーまでは見抜けますが、実行時エラーや意味的エラーは見抜けません。

■コンパイルのときの構文エラーの見方

コンパイラは、構文エラーがあれば、該当箇所を表示して、英語（あるいは日本語）でエラーの内容を表示してくれます。基本的には、ファイル名、行番号、何が悪かったか、該当箇所、停止するまでにいくつエラーがあったかなどを表示してくれます。SUNのJDKの英語でのエラー表示の例をいくつか見てみましょう。

```
Hello.java:4: Identifier expected. (識別子がありません)
    public static void main( String arg[] ) {
        ^
    1 error
```

これは、4行めに問題があつて、staticとするところをstatecと入力してしまったものです。コンパイラが^の位置に

ある単語がおかしいことを示しています。

```
Hello.java:5: ';' expected.   (セミコロンが必要です)
      System.out.println( "Tako" )
                        ^
```

1 error

5行目がおかしいようです。最後のセミコロン (;) が抜けているみたいです。

```
Hello.java:6: '}' expected. (} が必要です)
    }
    ^
```

最後の)を書き忘れたようです。波括弧 (Braces) は、開いたら ({を書いたら) 閉じる (} を書く) というのを忘れないでください。

```
Hello.java:5: Method println(java.lang.String) not found in class java.io.PrintStream.
      (java.io.PrintStreamクラスには、printlnというメソッドはありませんよ)
      System.out.println( "Tako" );
                        ^
```

1 error

printlnとかくところをprintlnと書いてしまったようです。大文字のアイ (I) と、小文字のエル (l) 、そして数字の1は、特に間違いやすいので綴りミスに注意してください。

■CodeWarriorでのエラーの表示

CodeWarrior IDEでは、コンパイラによって構文エラーが発見されると次のように、該当場所を矢印で表示してくれます。該当場所を示すダイアログの中でも、エディタの中でも、該当場所を修正することができます。修正が終わったら、保存 (上書き保存) して下さい。エラーが表示されている場合は、なくなるまで修正してください。

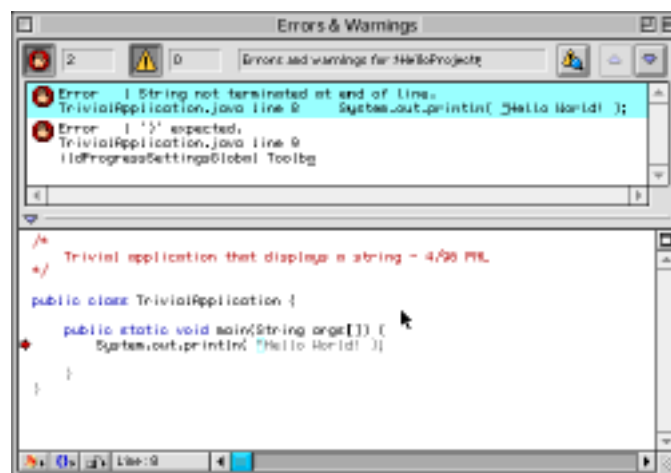


図2-17 CodeWarrior IDEによるエラー表示ダイアログ

1回の修正を行なって再度コンパイルしても、やはり更なる構文エラーが発見されることがあります。コンパイルが最後までうまくいくまで、根気良くプログラムを直していきましょう。

■間違いやすく発見しにくい構文エラーの例

今まで学生がよく起こしてしまう間違いで、特に自分自身では発見しにくいものの例をいくつか挙げてみます。

★ダブルクォーテーション（”）を閉じ忘れた場合です。後半の”が抜けています。

例： `System.out.println("This is a sample.);`

★最後のセミコロン（；）を付け忘れた場合です。

例： `System.out.println("This is a sample.")`

★記号が日本語で入力されている場合です。

記号は、半角の英字で入力しなければなりません。間違いやすい記号として、次の幾つがあります。

” 文字列が終了していないと見なされてしまいます。

□（スペース） character 0241と表示されていたら、どこかに全角の空白が入っています。

（ { } ） ; 括弧などが全角の記号になっているときは見分けにくいですね。

★その他の間違いやすいが見つけない例

- ・ピリオド（.）を入力していない
- ・Systemとout、およびoutとprintlnの間にピリオドを入れた後に空白をあけてしまった。
- ・括弧を閉じ忘れた。逆に余計に閉じてしまった。括弧は|や)あるいは、}等です。
- ・名前が違っている。println→println（小文字のエル）
- ・大文字にしていない。system→System

■エラーが発生したら、、、

特にエラーが発生したときに間違いやすいのは、エディタで修正した後に保存しなくても一度コンパイルしてしまうことです。「直したはずなのに同じエラーメッセージが表示されている。おかしいな、、、」ということになってしまいます。

また、実行までうまくいったときでも、改めてプログラムの修正をするときもあります。このときは、エディタで保存していなかったり、コンパイラでコンパイルをやり直さないと、新しく修正されたプログラムが実行されません。前のプログラムを再び実行することになります。

1. エディタで修正をして、保存する。
2. コンパイラを使って、コンパイラのやり直し。

この2つの過程を経て、はじめてプログラムの修正が反映されるのです。忘れないでおきましょう。

2.5 まとめと課題

■まとめ

★開発環境

- ・標準型の開発環境
- ・統合型の開発環境

★アプリケーションプログラムの作成から実行まで（CodeWarrior版）

- ・CodeWarrior IDEでNew（新規）で新しいプロジェクトを作る
- ・Java Stationeryを選び、プロジェクト名を指定する
- ・Java Applicationを選ぶ
- ・エディタでプログラムを修正して、ファイルに保存する
- ・プログラムファイルをコンパイルする（メニューやボタンを使う）
- ・ラインタイム・インタープリタでアプリケーションを実行する（メニューやボタンを使う）
- ・実行が終わったら、必ずアプリケーションを終了させる。

★アプレットプログラムの作成から実行まで（CodeWarrior版）

- CodeWarrior IDEでNew（新規）で新しいプロジェクトを作る
- Java Stationeryを選び、プロジェクト名を指定する
- Java Appletを選ぶ
- エディタでプログラムを記述して、ファイルに保存する
- エディタでHTMLのテキストを記述して、ファイルに保存する
- プログラムファイルをコンパイルする（メニューやボタンを使う）
- アプレットビューワで、実行する（メニューやボタンを使う）

★構文エラーの修正

- エディタで修正して保存する
- もう一度、コンパイラでコンパイルし直す

■課題

2-1

アプリケーションプログラムで、System.out.printlnを指定するとメッセージを表示した後に改行されます。printlnの部分をprintにすると、メッセージを表示した後に改行されません。そこで、System.out.printの行を複数書いてみてこれを確認なさい。

2-2

二重引用符 "（Double quotation mark）で囲まれた文字列はHello, World!という英語でしたが、これを日本語の文字列にしてみて、うまく表示できるかどうか確認なさい。アプリケーションでもアプレットでも両方やってみなさい。

2-3

アプリケーションプログラムで、System.out.printlnやSystem.out.printを複数使って、つぎのような文章を表示するようなプログラムを書いてみなさい。ファイル名は、Test.javaとしましょう（よってクラス名もTestとしてください）。

The java command executes Java bytecodes created by the Java compiler, javac. The JIT (just in time) compiler compiles these bytecodes into machine instructions by default in the Solaris Java VM when using java.

2-4

もし、自分の Web ページを持っている場合は、この章で作成したアプレットのクラスファイル（HelloWeb.class）とHTMLファイル（Tester.html）を、そのWebページのフォルダ（ディレクトリ）にコピーして、外部からNetscapeやInternet Explorerでアプレットの実行ができるかどうか確かめてみなさい。