i アプリ実習マニュアル

第零版
2005 年 9 月 1 日（木）

目次

第 1 章　i アプリの基礎

1.1　i アプリとは何か

1.1.1　i モードと i アプリ

株式会社 NTT ドコモによって提供されている、携帯電話とインターネットとを接続するサービスは、「i モード」(i-mode) と呼ばれます。このサービスを利用すると、携帯電話を使ってメールを送受信したりウェブサイトを閲覧したりすることができます。

i モードに対応した携帯電話には、アプリケーションをウェブサイトからダウンロードして実行することができるという機能が搭載されています。その機能を使って実行することのできるアプリケーションは、「i アプリ」(i-application) と呼ばれます。

1.1.2　公式アプリと勝手アプリ

i モードに対応した携帯電話は、ウェブサイトへのリンクから構成されている「iMenu」と呼ばれるメニューを持っています。このメニューに掲載されているのは、NTT ドコモによる内容の審査を通過したウェブサイトで、「公式サイト」と呼ばれます。そして、公式サイトで配布されている i アプリは、「公式アプリ」と呼ばれます。

i モードを使って閲覧することのできるウェブサイトは、公式サイトだけではありません。NTT ドコモとは無関係に勝手に作られたウェブサイト、すなわち「勝手サイト」と呼ばれるウェブサイトを閲覧することも可能です。同様に、i アプリも、NTT ドコモとは無関係に勝手に配布されているものをダウンロードして実行することが可能です。そのような、NTT ドコモとは無関係に配布されている i アプリは、「勝手アプリ」と呼ばれます。

1.1.3　Java のクラスライブラリ

i アプリの開発には、Java というプログラミング言語が使われます。

Java を使ってプログラムを書くためには、そのためのクラスライブラリを使う必要があります。Java の標準的なクラスライブラリとしては、対象となる機器の規模によって選択することができるように、3 種類のものが準備されています。それらの種類の名称と、対象となる機器は、次のとおりです。

Java2 Standard Edition (J2SE)	デスクトップ向け
Java2 Enterprise Edition (J2EE)	企業サーバー向け
Java2 Micro Edition (J2ME)	組み込み機器向け

カーナビ、セットトップボックス、携帯電話、ネットワーク家電のような組み込み機器というのは、機能や能力が多岐に渡るため、J2ME は、「コンフィギュレーション」と「プロファイル」と呼ばれるものをいくつも持つことによって、多くの機器に対応できるように構成されています。

「コンフィギュレーション」(configuration) というのは、J2ME のうちで基本的な機能を提供する部分のことで、現時点では、

- Connected Device Configuration (CDC)
- Connected Limited Device Configuration (CLDC)

という二つのものがあります。CDC がカーナビやセットトップボックスなどを対象としているのに対して、CLDC は、携帯電話やネットワーク家電のような、CDC が対象とする機器よりも CPU が低速でメモリーも少ない機器を対象としています。

「プロファイル」(profile) というのは、J2ME のうちで、個々の組み込み機器に特有の機能を利用するための部分のことです。プロファイルの例としては、

- Mobile Information Device Profile (MIDP)
- DoCoMo Java (DoJa)

などがあります。

1.1.4　CLDC

Java の 3 種類のクラスライブラリのうちで、i アプリの開発には J2ME が使われます。そして、コンフィギュレーションとしては CLDC が使われます。

パッケージ	主要なクラス
java.lang	String、Integer、System、Runtime、Thread
java.util	Vector、Stack、Calendar、Date、Random
java.io	DataInputStream、DataOutputStream
javax.microedition.io	Connector

表 1.1: CLDC のパッケージと主要なクラス

パッケージ	主要なクラス
com.nttdocomo.ui	IApplication、Graphics、Canvas、Button
com.nttdocomo.util	Timer、ScheduleDate、Phone、Base64
com.nttdocomo.io	BufferdReader、PrintWriter
com.nttdocomo.system	ApplicationStore、PhoneBook、Schedule

表 1.2: DoJa の主要なパッケージと主要なクラス

表 1.1 は、CLDC を構成しているパッケージと、それぞれのパッケージに含まれているクラスのうちの主要なものを示しています。

1.1.5 DoJa

i アプリの開発には、プロファイルとして DoJa が使われます。

表 1.2 は、DoJa を構成しているパッケージのうちの主要なものと、それぞれのパッケージに含まれているクラスのうちの主要なものを示しています。

1.1.6 この文章について

この「i アプリ実習マニュアル」という文章は、i アプリを開発する方法について解説したものです。ただし、この文章は、読者はすでに Java が使えるようになっていると仮定して書かれています。この文章を読んでいて、Java に関してわからない点があった場合は、Java に関する書籍などを参照してください。

1.2 i アプリ開発ツールの使い方

1.2.1 開発ツールの準備

i アプリを開発するためには、Windows がインストールされているコンピュータと、次の二つのソフトが必要です。

- Java 2 SDK, Standard Edition
- iαppli Development Kit

これらのソフトはいずれも、インターネット上のサイトから無料でダウンロードすることができます。Java 2 SDK はサン・マイクロシステムズのサイトにあって、iαppli Development Kit は NTT ドコモのサイトにあります。

1.2.2 iαppliTool

iαppli Development Kit をインストールすると、スタートメニューの中に、iαppliTool というプログラムのファイルへのショートカットが作られます。

iαppliTool は、i アプリを開発する作業を支援するプログラムです。このプログラムには、携帯電話のエミュレーターも含まれています。このエミュレーターを使うことによって、携帯電話ではなくてコンピュータの上で i アプリを動作させることができます。

1.2.3 プロジェクトの新規作成

i アプリを開発する過程で扱われる各種のファイルは、「プロジェクト」(project) と呼ばれる単位で管理されます。これから新しい i アプリを開発する、というときには、プロジェクトを新

規作成する必要があります。

プロジェクトを新規作成したいときは、まず、iαppliTool のウィンドウの上にある「プロジェクト新規作成」というボタンをクリックします。そうすると、「新規作成」というダイアログボックスが開きますので、そこにプロジェクト名を入力して「作成」というボタンをクリックします。

それでは実際に、daikichi という名前のプロジェクトを新規作成してみてください。

1.2.4 ソースコードの作成

プロジェクトを新規作成すると、プロジェクトと同じ名前のフォルダが作られて、そのフォルダの下にもいくつかのフォルダが作られます。それらのフォルダのうちで、src という名前のものは、ソースコードのファイルを置くためのものです。

それでは、実際にソースコードのファイルを作ってみましょう。何らかのテキストエディターを使って、次のプログラムを入力してください。

プログラムの例 IDaikichi.java

```
import com.nttdocomo.ui.*;

public class IDaikichi extends IApplication {
    public void start() {
        Panel panel = new Panel();
        panel.add(new Label("あなたの運勢は大吉です。"));
        Display.setCurrent(panel);
    }
}
```

1.2.5 プログラムのビルド

ソースコードをコンパイルして実行可能なプログラムを作ることを、プログラムを「ビルドする」(build) と言います。

i アプリをビルドしたいときは、iαppliTool のウィンドウの上にある「ビルド」というボタンをクリックします。コンパイルの際にエラーが出た場合は、iαppliTool のウィンドウの上にエラーメッセージが表示されます。

i アプリをビルドすると、プロジェクトのフォルダの下にある bin というフォルダの下に、クラスファイルなどから構成される JAR ファイルが作られます。

1.2.6 エミュレーターによる実行

iαppliTool を起動したときに、その本体のウィンドウとは別のウィンドウで表示される、携帯電話の外観のように見えるものは、携帯電話のエミュレーターを操作するためのものです。

エミュレーターに i アプリを実行させたいときは、iαppliTool の本体のウィンドウの上にある「起動」というボタンをクリックします。

先ほど入力したソースコードからできた i アプリを実行すると、「あなたの運勢は大吉です。」という文字列がエミュレーターの画面に表示されます。

ちなみに、この i アプリには自分を終了させるという機能がありませんので、これを終了させたいときは強制終了の操作をする必要があります。i アプリは、終話キーを押すことによって強制的に終了させることができます。

1.2.7 プロジェクトの読み込み

iαppliTool を起動したのち、すでに存在しているプロジェクトに対して何らかの操作をしたいときは、そのプロジェクトを読み込むという操作をする必要があります。

プロジェクトを読み込みたいときは、iαppliTool のウィンドウの上にある「プロジェクト読込み」というボタンをクリックします。そうすると、プロジェクトを選択するダイアログボックスが表示されます。読み込みたいプロジェクトを選択して、「読込み」というボタンを押すと、そのプロジェクトが読み込まれます。

1.3 i アプリの配布

1.3.1 i アプリの配布に必要なファイル

インターネットを使って i アプリを配布するためには、ウェブサーバーの上に、次の三種類のファイルを置いておく必要があります。

- i アプリの本体（JAR ファイル）
- ADF
- HTML 文書

1.3.2 ADF

ADF というのは、application description file の略称で、i アプリの属性について記述したテキスト、またはそのテキストが格納されているファイルのことです。ADF の名前には、 `.jam` という拡張子を付けることになっています。

属性を設定する対象となる i アプリの個々の項目は、「エントリー」(entry) と呼ばれます。それぞれのエントリーは、「エントリー名」(entry name) と呼ばれる名前によって識別されます。そして、それぞれのエントリーに設定される具体的な属性は、「エントリー値」(entry value) と呼ばれます。

ADF を書くことによって、さまざまなエントリーにエントリー値を設定することができます。それらのうちの多くは、必要な場合だけ書けばいいのですが、次の五つのエントリーは、どんな場合でもかならず書かないといけないことになっています。

`AppName`	i アプリの名前。
`PackageURL`	JAR ファイルの URL。相対パスの起点は ADF の位置。
`AppSize`	JAR ファイルの大きさ。単位はバイト。
`AppClass`	i アプリの本体となるクラスの名前。
`LastModified`	最終更新日時。形式は、`Dow, DD Mon YYYY HH:MM:SS TimeZone`

ADF のそれぞれの行は、

エントリー名 = エントリー値 改行

という構文を持つ文字列です。この文字列は、「エントリー名」で指定されたエントリーに対して「エントリー値」を設定する、という意味を持っています。ですから、「大吉」という名前を i アプリに与えたいならば、

```
AppName = 大吉
```

という行を書けばいいわけです。

なお、ADF を書くときに使うことのできる日本語の文字コードは、シフト JIS のみです。

iαppliTool は、プロジェクトを新規作成したときに、ADF を生成して、プロジェクト名に `.jam` という拡張子を付けた名前のファイルにそれを格納して、`bin` というフォルダの下にそのファイルを置きます。そして、i アプリをビルドするたびに、その内容を修正します。

ADF はテキストデータですので、普通のテキストエディターを使って編集することもできますが、iαppliTool を使って編集することも可能です。iαppliTool を使って ADF を編集したいときは、iαppliTool のウィンドウの上にある「ADF 設定」というボタンをクリックします。そうすると、ADF を編集するための「ADF 設定」というダイアログボックスが開きます。

1.3.3 HTML 文書

iαppliTool は、i アプリをビルドするときに、その i アプリを配布するための HTML 文書の雛型を生成して、`Download.html` というファイルにそれを格納して、`bin` というフォルダの下にそのファイルを置きます。ですから、i アプリをダウンロードするための HTML 文書を作成するときは、その雛型を利用するといいでしょう。

iαppliTool は、プロジェクト名が `daikichi` だとすると、次のような HTML 文書の雛型を生成します。

```
<HTML>
<HEAD>
<TITLE>Download Page</TITLE>
```

```
</HEAD>
<BODY>
<OBJECT declare id="daikichi"
        data="daikichi.jam"
        type="application/x-jam">
</OBJECT>
<BR>
<A ijam="#daikichi" href="notapplicable.html">DOWNLOAD</A>
</BODY>
</HTML>
```

この中に書かれている notapplicable.html という URL は、i アプリに対応していない機種で i アプリをダウンロードしようとしたときに表示される HTML 文書を指示しています。その HTML 文書は、iαppliTool によって生成されるわけではありませんので、自分で書く必要があります。

1.4 DoJa の基礎

1.4.1 IApplication

DoJa の中にある com.nttdocomo.ui というパッケージには、ユーザーインターフェースを作るためのさまざまなクラスが含まれています。

それらのクラスの中で、まず最初に説明しなければならないのは、IApplication という抽象クラスです。i アプリの本体となるクラスは、かならず、この抽象クラスを実装したクラスでないといけません。そもそも i アプリというのは、IApplication クラスを実装したクラスのインスタンスのことなのです。

IApplication という抽象クラスを実装するためには、

```
public abstract void start() start
```

という抽象メソッドを実装する必要があります。start は、i アプリが起動されたときに最初に呼び出されるメソッドです。つまり、Java の普通のプログラムで main に相当するものが、i アプリでは start になるわけです。

1.4.2 i アプリの終了

i アプリは、terminate というメソッドを持っています。このメソッドは、引数を何も受け取らないで、i アプリを終了させるという動作をしますですから、i アプリが自分を終了させたいときは、このメソッドを呼び出せばいいわけです。

IApplication クラスを実装したクラスのインスタンスメソッドの中で i アプリを終了させたいときは、ただ単に、

```
terminate();
```

という文を書けばいいわけですが、それ以外の場所で i アプリを終了させるためには、動作中の i アプリを求めるという処理が必要になります。

IApplication クラスは、getCurrentApp というクラスメソッドを持っています。これは、動作中の i アプリを戻り値として返すクラスメソッドです（引数は何も受け取りません）。ですから、どんなメソッドの中でも、

```
IApplication.getCurrentApp().terminate();
```

という文を実行することによって、i アプリを終了させることができます。

1.4.3 フレーム

IApplication クラスは、i アプリを作るための土台のようなクラスです。しかし、このクラスには、ボタンやテキストボックスのような GUI の部品を表示する機能はありません。また、グラフィックスを描画する機能もありません。

Java では、GUI の部品は「コンポーネント」(component) と呼ばれます。コンポーネントを表示する i アプリを作りたいときは、「パネル」(panel) と呼ばれるものを使います。パネルというのは、その上にさまざまなコンポーネントを表示することのできる板のようなものだと考えることができます。

グラフィックスを描画するiアプリを作りたいときは、「キャンバス」(canvas) と呼ばれるものを使います。キャンバスというのも板のようなもので、その上にさまざまなグラフィックスを描画することができるようになっています。

パネルは Panel というクラスから生成され、キャンバスは Canvas という抽象クラスを実装したクラスから生成されます。これらのクラスはどちらも、 Frame という抽象クラスのサブクラスです。

Frame という抽象クラスを実装したクラスのインスタンスは、「フレーム」(frame) と呼ばれます。パネルとキャンバスは、どちらもフレームの一種です。

1.4.4 カレントフレーム

ひとつの時点で画面に表示することのできるフレームは、ひとつだけです。つまり、同時に二つ以上のフレームを画面に表示するということはできません。しかし、ひとつのiアプリが複数のフレームを切り替えて表示するということは可能です。

画面に表示されているフレームは、その時点での「カレントフレーム」(current frame) であると言われます。

カレントフレームを切り替えたいときは、 Display というクラスが持っている setCurrent というクラスメソッドを使います。このクラスメソッドは、引数としてフレームを受け取って、それをカレントフレームにします。たとえば、 panel という変数がパネルを指し示しているとするとき、

```
Display.setCurrent(panel);
```

という式文を実行すると、そのパネルがカレントフレームになります。

1.5 ダイアログ

1.5.1 ダイアログの生成

人間に対してメッセージを表示したり簡単な質問をしたり確認を求めたりするときにプログラムが表示するウィンドウは、「ダイアログ」(dialog) と呼ばれます。

DoJa では、ダイアログは Dialog というクラスから生成されます。このクラスのコンストラクタは、二つの引数を受け取ります。1 個目はダイアログの種類を指定する整数で、2 個目はダイアログのタイトルとして表示される文字列です。ダイアログの種類を指定する整数は、 Dialog クラスの中で次のような定数として定義されています。

DIALOG_INFO	メッセージの表示
DIALOG_WARNING	警告の表示
DIALOG_ERROR	エラーメッセージの表示
DIALOG_YESNO	「はい」か「いいえ」で答える質問
DIALOG_YESNOCANCEL	「はい」「いいえ」「取り消し」で答える質問

たとえば、

```
Dialog info = new Dialog(Dialog.DIALOG_INFO, "お知らせ");
```

という文を書くことによって、「お知らせ」というタイトルを持つ、メッセージを表示するためのダイアログを生成することができます。

1.5.2 メッセージの設定

ダイアログの上に表示するメッセージは、ダイアログが持っている setText というメソッドを呼び出すことによって設定することができます。このメソッドは、引数として 1 個の文字列を受け取って、その文字列をダイアログにメッセージとして設定します。たとえば、

```
info.setText("あなたの運勢は大吉です。");
```

という文を実行することによって、 info が指し示しているダイアログに対して、「あなたの運勢は大吉です。」という文字列をメッセージとして設定することができます。

1.5.3　ダイアログの表示

ダイアログはフレームの一種ですが、それを表示したいときは、Display.setCurrent ではなくて、ダイアログが持っている show というメソッドを呼び出します（引数は何も渡しません）。たとえば、

```
info.show();
```

という文を実行することによって、 info が指し示しているダイアログを画面に表示することができます。

1.5.4　押されたボタンの判定

ダイアログが持っている show メソッドは、ダイアログの上に表示されたボタンのうちで人間によって押されたものを識別する整数を、戻り値として返します。ボタンを識別する整数は、Dialog クラスの中で次のような定数として定義されています。

BUTTON_OK	「OK」「了解」
BUTTON_CANCEL	「CANCEL」「取り消し」
BUTTON_YES	「YES」「はい」
BUTTON_NO	「NO」「いいえ」

プログラムの例　IDialog.java

```
import com.nttdocomo.ui.*;

public class IDialog extends IApplication {
    public void start() {
        Dialog yesno = new Dialog(Dialog.DIALOG_YESNO,
            "質問");
        yesno.setText("あなたは地球の人間ですか。");
        int answer = yesno.show();
        String message;
        if (answer == Dialog.BUTTON_YES)
            message = "でしたら、特に問題はありません。";
        else
            message = "それはすごいですね。";
        Dialog info = new Dialog(Dialog.DIALOG_INFO,
            "メッセージの表示");
        info.setText(message);
        info.show();
        terminate();
    }
}
```

第 2 章　コンポーネント

2.1　コンポーネントの基礎

2.1.1　コンポーネントとは何か

GUI は、画面の上に表示されるさまざまな種類の部品から構成されます。GUI を構成する部品というものを意味する言葉としては、さまざまなものが使われていますが、Java ではそれを「コンポーネント」(component) と呼ぶのが普通です。したがって、DoJa でも、GUI の部品は「コンポーネント」と呼ばれます。

Java では、ひとつのコンポーネントはひとつのオブジェクトによって表示されます。「コンポーネント」という言葉は、コンポーネントを表示するオブジェクト、という意味でも使われます。

2.1.2　コンポーネントを生成するクラス

コンポーネントは、その種類に応じたクラスから生成されます。DoJa は、コンポーネントを生成するクラスとして次のようなものを持っています。

Label	文字列の表示

ImageLabel	画像の表示
Button	ボタン
ImageButton	画像を表示することのできるボタン
AnchorButton	ウェブページのアンカーのようなボタン
TextBox	文字列の読み込み
ListBox	項目の選択
Ticker	文字列のスクロール表示
VisualPresenter	アニメーションの再生

2.1.3 パネルのタイトル

第 1.4 節で説明したように、コンポーネントを表示する i アプリを作りたいときは、「パネル」と呼ばれる、その上にさまざまなコンポーネントを表示することのできる板のようなものを使います。

パネルの上端には、1 行の文字列を表示することのできる領域があります。その領域に表示される文字列を設定したいときは、パネルが持っている setTitle というメソッドを使います。このメソッドに引数として文字列を渡すと、その引数が、パネルの上端に表示される文字列として設定されます。

プログラムの例 ITitle.java

```
import com.nttdocomo.ui.*;

public class ITitle extends IApplication {
    public void start() {
        Panel panel = new Panel();
        panel.setTitle("私はパネルです。");
        Display.setCurrent(panel);
    }
}
```

2.2 ラベル

2.2.1 ラベルの基礎

Label というクラスから生成されたコンポーネントは、「ラベル」(label) と呼ばれます。ラベルは、1 行の文字列を表示するという機能を持っています。ちなみに、複数行のテキストを表示したいときは、もう少し先のところで紹介する、テキストボックスというコンポーネントを使います。

Label クラスのコンストラクタに引数として文字列を渡すと、その文字列を表示するラベルが生成されます。たとえば、

```
Label label = new Label("私はラベルです。");
```

という文を実行すると、「私はラベルです。」という文字列を表示するラベルが生成されて、そのラベルが label という変数に設定されます。

ラベルの上に表示される文字列は、実行時に変更することも可能です。表示される文字列を変更したいときは、ラベルが持っている setText というメソッドを呼び出します。このメソッドに引数として文字列を渡すと、その文字列がラベルの上に表示されます。

2.2.2 パネルへのコンポーネントの追加

コンポーネントを画面に表示するためには、そのコンポーネントをパネルに追加する必要があります。コンポーネントをパネルに追加したいときは、パネルが持っている add というメソッドを呼び出して、追加したいコンポーネントを引数として渡します。たとえば、panel という変数がパネルを指していて、compo という変数がラベルを指しているとするとき、

```
panel.add(compo);
```

という文を実行すると、そのコンポーネントがパネルに追加されます。

プログラムの例 ILabel.java

```
import com.nttdocomo.ui.*;

public class ILabel extends IApplication {
    public void start() {
        Panel panel = new Panel();
        panel.add(new Label("私はラベルです。"));
        Display.setCurrent(panel);
    }
}
```

2.2.3 コンポーネントの色

すべてのコンポーネントは、自分を表示するための色を設定する、

- `setBackground` 背景色を設定する。
- `setForeground` 前景色を設定する。

というメソッドを持っています。これらのメソッドは、設定する色をあらわす 1 個の整数を引数として受け取ります。

i アプリの場合、どんな整数がどんな色をあらわすのかというのは、それを実行する携帯電話の機種ごとに変化します。しかし、 Graphics というクラスが持っている、

- `getColorOfName` 定数で指定された色をあらわす整数を返す。
- `getColorOfRGB` RGB で指定された色をあらわす整数を返す。

というメソッドのいずれかを使うことによって、色をあらわす整数を求めることができます。

getColorOfName は、 Graphics クラスで定義されている、色を指定するための定数の値を引数として受け取って、その色をあらわす整数を返します。 Graphics クラスでは、

BLACK	BLUE	LIME	AQUA	RED	FUCHSIA	YELLOW	WHITE
GRAY	NAVY	GREEN	TEAL	MAROON	PURPLE	OLIVE	SILVER

という 16 個の定数が定義されています。

getColorOfRGB は、赤、緑、青のそれぞれの輝度をあらわす整数（0 から 255 まで）を引数として受け取って、それらによって示される色をあらわす整数を返します。

プログラムの例 IComponentColor.java

```
import com.nttdocomo.ui.*;

public class IComponentColor extends IApplication {
    public void start() {
        Panel panel = new Panel();
        Label label = new Label("背景は水色で前景は空色");
        panel.add(label);
        Display.setCurrent(panel);
        int aqua = Graphics.getColorOfName(Graphics.AQUA);
        int skyblue = Graphics.getColorOfRGB(0, 128, 255);
        label.setBackground(aqua);
        label.setForeground(skyblue);
    }
}
```

2.2.4 コンポーネントのフォント

コンポーネントの上に表示される文字の大きさは、そのコンポーネントにフォントを設定することによって変更することができます。

フォントは、 Font というクラスが持っている getFont というクラスメソッドを呼び出すことによって取得します。このメソッドは、フォントの大きさをあらわす整数を引数として受け取って、指定された大きさのフォントを戻り値として返します。フォントの大きさをあらわす整数は、 Font クラスの中で、次のような定数として定義されています。

- `SIZE_LARGE` 大
- `SIZE_MEDIUM` 中
- `SIZE_SMALL` 小

SIZE_TINY　極小

フォントをコンポーネントに設定したいときは、そのコンポーネントが持っている setFont というメソッドを使います。このメソッドは、引数としてフォントを受け取って、それをコンポーネントに設定します。

プログラムの例　IComponentFont.java

```
import com.nttdocomo.ui.*;

public class IComponentFont extends IApplication {
    public void start() {
        Panel panel = new Panel();
        Label label = new Label("大きなフォント");
        panel.add(label);
        Display.setCurrent(panel);
        label.setFont(Font.getFont(Font.SIZE_LARGE));
    }
}
```

2.3 ボタン

2.3.1 ボタンの基礎

Button というクラスから生成されたコンポーネントは、「ボタン」(button) と呼ばれます。ボタンは、自分が押されたときに、あらかじめ設定されている動作を実行する、という機能を持っています。

Button クラスのコンストラクタに引数として文字列を渡すと、その文字列が表示されたボタンが生成されます。たとえば、

```
Button button = new Button("私はボタンです。");
```

という文を実行すると、「私はボタンです。」という文字列が表示されたボタンが生成されて、そのボタンが button という変数に設定されます。

2.3.2 イベントリスナー

プログラムに対する人間の操作などによって発生する出来事は、「イベント」(event) と呼ばれます。

プログラムは、イベントが発生したときに、それに対応する動作を実行することができます。プログラムのそのような動作は、「イベント処理」(event processing) と呼ばれます。

i アプリでのイベント処理の方法は、パネルを使う場合とキャンバスを使う場合とで異なります。パネルを使う i アプリでは、イベント処理を実行するために、「イベントリスナー」(event listener) と呼ばれるオブジェクトが使われます。

イベントリスナーというのは、イベントが発生したときに、そのイベントに対応して自動的に呼び出されるメソッドを持っているオブジェクトのことです。

2.3.3 コンポーネントリスナー

イベントリスナーにはさまざまな種類があるのですが、それらのうちで、コンポーネントでイベントが発生したときに呼び出されるメソッドを持っているものは、「コンポーネントリスナー」(component listener) と呼ばれます。

コンポーネントリスナーを生成するクラスは、 ComponentListener というインターフェースを実装している必要があります。このインターフェースを実装するクラスを定義するときに実装しないといけないメソッドはひとつだけで、

```
public void componentAction(Component source, int type, int param)
```

というものです。このメソッドは、コンポーネントでイベントが発生したときに呼び出されるもので、1 個目の引数としてイベントが発生したコンポーネント、2 個目の引数としてイベントのタイプをあらわす整数、3 個目の引数としてイベントのパラメーターを受け取ります。

イベントのタイプをあらわす整数は、 ComponentLisner インターフェースの中で次のような定数として定義されています。

BUTTON_PRESSED	ボタンが押された。
SELECTION_CHANGED	リストボックスの項目の選択が変更された。
TEXT_CHANGED	テキストボックスへの文字列の入力が終了した。

2.3.4　パネルへのコンポーネントリスナーの登録

イベントの発生に対応して、コンポーネントリスナーの中のメソッドが呼び出されるようにするためには、そのコンポーネントリスナーをパネルに登録しておく必要があります。

パネルは、setComponentListener というメソッドを持っています。このメソッドを呼び出して、コンポーネントリスナーを引数として渡すことによって、それをパネルに登録することができます。

プログラムの例　IButton.java

```
import com.nttdocomo.ui.*;

class ButtonPanel extends Panel {
    int number;
    Label numberLabel;
    Button countup, exit;

    ButtonPanel() {
        setComponentListener(new ComponentListener() {
            public void componentAction(
                    Component source, int type, int param) {
                switch (type) {
                case BUTTON_PRESSED:
                    if (source == countup)
                        increaseNumber();
                    else if (source == exit)
                        terminate();
                    break;
                }
            }
        });
        number = 0;
        numberLabel = new Label("0");
        add(numberLabel);
        countup = new Button("増やす");
        add(countup);
        exit = new Button("終了");
        add(exit);
    }

    void increaseNumber() {
        number++;
        numberLabel.setText(Integer.toString(number));
    }

    void terminate() {
        IApplication.getCurrentApp().terminate();
    }
}

public class IButton extends IApplication {
    public void start() {
        Display.setCurrent(new ButtonPanel());
    }
}
```

2.4　リストボックス

2.4.1　リストボックスの基礎

ListBox というクラスから生成されたコンポーネントは、「リストボックス」(list box) と呼ばれます。リストボックスというのは、「項目」(item) と呼ばれる選択肢をいくつか持っていて、それらの項目のうちからひとつまたは複数を人間に選択させることができる、というコンポーネントです。

DoJa のリストボックスにはさまざまなタイプがあります。ListBox クラスのコンストラクタは、タイプをあらわす整数を引数として受け取って、指定されたタイプのリストボックスを生成します。タイプをあらわす整数は、ListBox クラスの中で次のような定数として定義されています。

SINGLE_SELECT	単一選択リスト
RADIO_BUTTON	ラジオボタンリスト
CHECK_BOX	チェックボックスリスト
NUMBERED_LIST	番号付き単一選択リスト
MULTIPLE_SELECT	複数選択リスト
CHOICE	ポップアップ形式の単一選択リスト

たとえば、

```
ListBox listbox = new ListBox(RADIO_BUTTON);
```

という文を実行すると、ラジオボタンリストが生成されて、そのラジオボタンリストが listbox という変数に設定されます。

2.4.2 リストボックスへの項目の追加

リストボックスは、生成された直後は項目をまったく持っていません。リストボックスに項目を追加したいときは、それが持っている次のようなメソッドを使います。

append	文字列を引数として受け取って、その文字列によってあらわされる項目をリストボックスに追加します。
setItems	文字列の配列を引数として受け取って、それを構成するそれぞれの文字列によってあらわされる項目をリストボックスに追加します。

2.4.3 項目の選択によるイベント

人間がリストボックスを操作して、その中の項目のどれかを選択すると、イベントが発生して、コンポーネントリスナーの中の componentAction というメソッドが呼び出されます。その場合、そのメソッドは 2 個目の引数（イベントのタイプをあらわす整数）として、

```
SELECTION_CHANGED
```

という定数であらわされる整数を受け取ります。

2.4.4 選択された項目の取得

リストボックスに追加されたそれぞれの項目は、「インデックス」(index) と呼ばれる整数によって識別されます。

リストボックスが持っている getSelectedIndex というメソッドは、選択された項目のインデックスのうちでもっとも小さいものを返します（引数は何も受け取りません）。

項目をあらわしている文字列を取得したいときは、getItem というメソッドを使います。このメソッドは、インデックスを引数として受け取って、そのインデックスによって識別される項目をあらわしている文字列を返します。

プログラムの例 IListBox.java

```
import com.nttdocomo.ui.*;

class ListBoxPanel extends Panel {
    ListBox listbox;
    Label selectedItem;
    Button exit;

    ListBoxPanel() {
        setComponentListener(new ComponentListener() {
            public void componentAction(
```

```
                    Component source, int type, int param) {
                switch (type) {
                case SELECTION_CHANGED:
                    if (source == listbox)
                        setSelectedItem();
                    break;
                case BUTTON_PRESSED:
                    if (source == exit)
                        terminate();
                    break;
                }
            }
        });
        createListBox();
        selectedItem = new Label("");
        add(selectedItem);
        exit = new Button("終了");
        add(exit);
    }

    void createListBox() {
        listbox = new ListBox(ListBox.CHOICE);
        for (int i = 0; i < 40; i++)
            listbox.append("項目" + i);
        add(listbox);
    }

    void setSelectedItem() {
        selectedItem.setText(
            listbox.getItem(listbox.getSelectedIndex()) +
            "が選択されました。");
    }

    void terminate() {
        IApplication.getCurrentApp().terminate();
    }
}

public class IListBox extends IApplication {
    public void start() {
        Display.setCurrent(new ListBoxPanel());
    }
}
```

2.5　テキストボックス

2.5.1　テキストボックスの基礎

TextBox というクラスから生成されたコンポーネントは、「テキストボックス」(list box) と呼ばれます。テキストボックスというのは、人間によって入力された文字列を読み込むためのコンポーネントです。

文字列を表示したいときはラベルを使えばいいわけですが、ラベルに表示することができるのは 1 行の文字列だけです。2 行以上の文字列を表示したいときは、ラベルの代わりにテキストボックスを使うことになります。

2.5.2　テキストボックスの生成

TextBox クラスのコンストラクタは、4 個の引数を受け取ります。1 個目は最初に表示する文字列、2 個目は 1 行に表示できる半角文字の個数、3 個目は表示する行数、そして 4 個目は文字列の表示モードをあらわす整数です。

文字列の表示モードというのは、普通に表示するか、それともパスワードとしてアスタリスク (*) で表示するか、ということです。それらをあらわす整数は、TextBox クラスの中で、次の定数として定義されています。

DISPLAY_ANY　普通に表示する。
DISPLAY_PASSWORD　パスワードとしてアスタリスクで表示する。

たとえば、

```
TextBox textbox = new TextBox("入力", 30, 4, TextBox.DISPLAY_ANY);
```

という文を実行することによって、最初に「入力」と表示されている、1 行が 30 文字で 4 行を普通に表示するテキストボックスを生成して、それを textbox という変数に設定することができます。

2.5.3 編集の可否の設定

文字列を読み込むという目的ではなくて、2 行以上の文字列を表示することを目的としてテキストボックスを使う場合は、文字列の編集ができないように設定するといいでしょう。

テキストボックスは、デフォルトでは編集が可能な状態に設定されています。その状態を変更したいときは、それが持っている setEditable というメソッドを呼び出します。このメソッドは、引数として 1 個の真偽値を受け取って、それがあらわしている状態を設定します。真は編集が可能な状態という意味で、偽は編集が不可能な状態という意味です。

2.5.4 入力の終了によるイベント

人間がテキストボックスへの文字列の入力を終了させる操作をすると、イベントが発生して、コンポーネントリスナーの中の componentAction というメソッドが呼び出されます。その場合、そのメソッドは 2 個目の引数（イベントのタイプをあらわす整数）として、

TEXT_CHANGED

という定数であらわされる整数を受け取ります。

2.5.5 入力された文字列の取得

人間によって入力された文字列をテキストボックスから取り出したいときは、テキストボックスが持っている getText というメソッドを呼び出します。このメソッドは、引数を何も受け取らないで、テキストボックスに入力されている文字列を戻り値として返します。

プログラムの例　ITextBox.java

```
import com.nttdocomo.ui.*;

class TextBoxPanel extends Panel {
    TextBox textbox1, textbox2;
    Button exit;

    TextBoxPanel() {
        setComponentListener(new ComponentListener() {
            public void componentAction(
                    Component source, int type, int param) {
                switch (type) {
                case TEXT_CHANGED:
                    if (source == textbox1)
                        setInputText();
                    break;
                case BUTTON_PRESSED:
                    if (source == exit)
                        terminate();
                    break;
                }
            }
        });
        textbox1 = new TextBox(
            "", 36, 8, TextBox.DISPLAY_ANY);
        add(textbox1);
        textbox2 = new TextBox(
            "", 36, 8, TextBox.DISPLAY_ANY);
        textbox2.setEditable(false);
        add(textbox2);
        exit = new Button("終了");
```

```
        add(exit);
    }

    void setInputText() {
        textbox2.setText(textbox1.getText());
    }

    void terminate() {
        IApplication.getCurrentApp().terminate();
    }
}

public class ITextBox extends IApplication {
    public void start() {
        Display.setCurrent(new TextBoxPanel());
    }
}
```

2.6　キーリスナー

2.6.1　キーリスナーの基礎

第 2.3 節で説明したように、パネルには、コンポーネントで発生したイベントを処理する、「コンポーネントリスナー」と呼ばれるイベントリスナーを登録することができるわけですが、パネルに登録することのできるイベントリスナーは、それだけではありません。

パネルには、「キーリスナー」(key listener) と呼ばれるイベントリスナーを登録することもできます。キーリスナーは、携帯電話のキーが押されたときに呼び出されるメソッドと、キーが離されたときに呼び出されるメソッドを持っているイベントリスナーです。

キーリスナーは、KeyListener というインターフェースを実装したクラスから生成されます。このインターフェースを実装するクラスは、キーが押されたときと離されたときのそれぞれに呼び出される、次の二つのメソッドを実装する必要があります。

```
public void keyPressed(Panel panel, int key)    押されたとき
public void keyReleased(Panel panel, int key)   離されたとき
```

これらのメソッドは、2 個の引数を受け取ります。1 個目はイベントが発生したときに表示されていたパネルで、2 個目はイベントを発生させたキーをあらわす整数です。

キーをあらわす整数は、Display クラスの中で定数として定義されています。たとえば、数字の 0 は KEY_0、数字の 9 は KEY_9、アスタリスクは KEY_ASTERISK、……という感じです。

2.6.2　パネルへのキーリスナーの登録

キーリスナーをパネルに登録したいときは、パネルが持っている setKeyListener というメソッドを呼び出します。このメソッドは、引数としてキーリスナーを受け取って、それをパネルに登録します。

プログラムの例　IKeyListener.java

```
import com.nttdocomo.ui.*;

class KeyListenerPanel extends Panel {
    Label keyInteger;

    KeyListenerPanel() {
        setKeyListener(new KeyListener() {
            public void keyPressed(Panel panel, int key) {
                setKeyInteger(key);
            }
            public void keyReleased(Panel panel, int key) { }
        });
        keyInteger = new Label("");
        add(keyInteger);
    }

    void setKeyInteger(int key) {
```

```
        keyInteger.setText(
            Integer.toString(key) +
            "番のキーが押されました。");
    }
}

public class IKeyListener extends IApplication {
    public void start() {
        Display.setCurrent(new KeyListenerPanel());
    }
}
```

2.6.3 ソフトキーリスナーの基礎

i モードに対応している携帯電話は、「ソフトキー」(soft key) と呼ばれる二つのキーを持っています。ソフトキーは左右に並んでいますので、ソフトキーのそれぞれは、「左ソフトキー」、「右ソフトキー」と呼ばれます。

ソフトキーの操作によって発生したイベントは、キーリスナーではなくて、「ソフトキーリスナー」(soft key listener) と呼ばれるイベントリスナーによって処理されます。

ソフトキーリスナーは、SoftKeyListener というインターフェースを実装したクラスから生成されます。このインターフェースを実装するクラスは、ソフトキーが押されたときと離されたときのそれぞれに呼び出される、次の二つのメソッドを実装する必要があります。

`public void softKeyPressed(int softKey)`	押されたとき
`public void softKeyReleased(int softKey)`	離されたとき

これらのメソッドは、押されたソフトキーをあらわす整数を引数として受け取ります。ソフトキーをあらわす整数は、Frame クラスの中で次のように定数として定義されています。

SOFT_KEY_1	左ソフトキー
SOFT_KEY_2	右ソフトキー

2.6.4 パネルへのソフトキーリスナーの登録

ソフトキーリスナーは、setSoftKeyListener というメソッドを呼び出すことによってパネルに登録することができます。このメソッドは、引数としてソフトキーリスナーを受け取って、それをパネルに登録します。

2.6.5 ソフトキーのラベル

ソフトキーに何らかの機能を割り当てている i アプリは、普通、その機能をあらわす文字列を画面の最下部に表示します。画面の最下部に表示される、ソフトキーの機能をあらわす文字列は、ソフトキーの「ラベル」(label) と呼ばれます。

パネルは、ソフトキーのラベルを自分に設定する、setSoftLabel というメソッドを持っています（Frame クラスから継承したものです）。このメソッドは、2 個の引数を受け取ります。1 個目はソフトキーをあらわす整数（Frame クラスで定数として定義されているもの）で、2 個目はソフトキーのラベルとして設定する文字列です。

プログラムの例 ISoftKeyListener.java

```
import com.nttdocomo.ui.*;

class SoftKeyListenerPanel extends Panel {
    int number;
    Label numberLabel;

    SoftKeyListenerPanel() {
        setSoftKeyListener(new SoftKeyListener() {
            public void softKeyPressed(int softKey) {
                switch (softKey) {
                case SOFT_KEY_1:
                    terminate();
                    break;
                case SOFT_KEY_2:
                    increaseNumber();
```

```
                    break;
                }
            }
            public void softKeyReleased(int softKey) { }
        });
        number = 0;
        numberLabel = new Label("0");
        add(numberLabel);
        setSoftLabel(Frame.SOFT_KEY_1, "終了");
        setSoftLabel(Frame.SOFT_KEY_2, "増やす");
    }

    void increaseNumber() {
        number++;
        numberLabel.setText(Integer.toString(number));
    }

    void terminate() {
        IApplication.getCurrentApp().terminate();
    }
}

public class ISoftKeyListener extends IApplication {
    public void start() {
        Display.setCurrent(new SoftKeyListenerPanel());
    }
}
```

2.7　タイマー

2.7.1　タイマーの基礎

GUI を持つプログラムは、基本的には、人間の操作によるイベントが発生したときにそれに対応する処理をするわけですが、場合によっては、人間による操作とは無関係に何らかの処理を実行させたいということもあります。そのような、人間による操作とは無関係に実行される処理のことを、「バックグラウンド処理」(background processing) と呼びます。

バックグラウンド処理を実行する方法のひとつは、「タイマー」(timer) と呼ばれるものを使うというものです。タイマーというのは、一定の時間間隔でイベントを発生させるオブジェクトのことです。タイマーがイベントを発生させるたびにメソッドが呼び出されるようにしておくことによって、そのメソッドにバックグラウンド処理を実行させることができます。

2.7.2　タイマーの生成

i アプリでは、パネルを使う場合とキャンバスを使う場合とで、扱うことのできるタイマーが異なっています。パネルを使う i アプリで扱うことができるのは、

```
com.nttdocomo.util.Timer
```

というクラスから生成されるタイマーです。これまでに紹介したクラスとは異なるパッケージにあるという点に注意してください。

`Timer` クラスには、引数を受け取らないコンストラクタしかありません。ですから、`Timer` クラスからタイマーを生成して、その参照を `timer` という変数に代入したいという場合は、

```
timer = new Timer();
```

という文を書くことになります。

なお、タイマーを使うためには、それに対して次のような初期設定を実行する必要があります。

- イベントを発生させる時間間隔の設定
- イベントの発生を繰り返すかどうかの設定
- タイマーリスナーの登録

2.7.3 イベントを発生させる時間間隔の設定

イベントを発生させる時間間隔は、タイマーが持っている setTime というメソッドを使って設定します。このメソッドに時間の長さ（単位はミリ秒）を引数として渡すと、その長さが時間間隔として設定されます。

2.7.4 イベントの発生を繰り返すかどうかの設定

タイマーは、デフォルトでは、イベントを 1 回だけ発生させて終了するという動作をするように設定されています。バックグラウンド処理のためにタイマーを使う場合は、この設定を変更して、イベントを何回も発生させるようにする必要があります。

イベントの発生を繰り返すかどうかは、タイマーが持っている setRepeat というメソッドを使って設定します。このメソッドに引数として true を渡せばイベントの発生を繰り返すようになり、 false を渡せば 1 回だけで終了するようになります。

2.7.5 タイマーリスナーの登録

タイマーにイベントリスナーを登録すると、そのイベントリスナーの中のメソッドが、タイマーが発生させたイベントに応じて呼び出されることになります。タイマーのイベントに応じて呼び出されるメソッドを持っているイベントリスナーは、「タイマーリスナー」(timer listener) と呼ばれます。

タイマーリスナーは、タイマーが持っている setListener というメソッドを使って設定します。このメソッドに引数としてタイマーリスナーを渡せば、それがタイマーに設定されます。

2.7.6 タイマーリスナーのインターフェース

タイマーリスナーは、 TimerListener というインターフェースを実装したクラスから生成されます。

TimerListener を実装するクラスは、

```
public void timerExpired(Timer source)
```

というメソッドを実装する必要があります。タイマーがイベントを発生させると、このメソッドが呼び出されます。

なお、 timerExpired が受け取る引数は、イベントを発生させたタイマーです。

2.7.7 タイマーの動作の開始

生成された直後のタイマーは、停止した状態になっています。その動作を開始させたいときは、タイマーが持っている start というメソッドを呼び出します。このメソッドは、引数を何も受け取りません。

プログラムの例 ITimer.java

```
import com.nttdocomo.ui.*;
import com.nttdocomo.util.*;

class TimerPanel extends Panel {
    int number;
    Label numberLabel;
    Timer timer;

    TimerPanel() {
        timer = new Timer();
        timer.setTime(1000);
        timer.setRepeat(true);
        timer.setListener(new TimerListener() {
            public void timerExpired(Timer source) {
                increaseNumber();
            }
        });
        timer.start();
        number = 0;
        numberLabel = new Label("0");
        add(numberLabel);
    }
```

```
    void increaseNumber() {
        number++;
        numberLabel.setText(Integer.toString(number));
    }
}

public class ITimer extends IApplication {
    public void start() {
        Display.setCurrent(new TimerPanel());
    }
}
```

第 3 章　キャンバス

3.1　キャンバスの基礎

3.1.1　キャンバスの基礎の基礎

第 1.4 節でも説明しましたが、i アプリは、「フレーム」(frame) と呼ばれるものを携帯電話の画面に表示することができます。フレームというのは、Frame という抽象クラスを実装したクラスのインスタンスのことです。たとえば、コンポーネントを表示するために使われるパネルというのは、フレームの一種です。そしてまた、「キャンバス」(canvas) と呼ばれるものも、フレームの一種です。キャンバスは、その上にグラフィックスを描画することのできるフレームです。

キャンバスを作りたいときは、Canvas というクラスを使います。ただし、Canvas は抽象クラスですので、このクラスからいきなりインスタンスを生成することはできません。それを実装したサブクラスを定義して、そのサブクラスのインスタンスを生成すると、それがキャンバスになります。

3.1.2　画面へのキャンバスの表示

キャンバスを画面に表示したいときは、パネルの場合と同じように、カレントフレームを設定するメソッドを使います。たとえば、canvas という変数がキャンバスを指し示しているとするならば、

```
Display.setCurrent(canvas);
```

という式文を実行することによって、そのキャンバスをカレントフレームとして設定することができます。

3.1.3　paint

Canvas という抽象クラスを実装したサブクラスを定義するためには、

```
public abstract void paint(Graphics g)
```

という抽象メソッドを実装する必要があります。これは、キャンバスに対する描画が必要になったときに自動的に呼び出されるメソッドです。ですから、グラフィックスを描画する処理をこのメソッドの動作として書いておくことによって、キャンバスにグラフィックスを描画することができます。

paint は、「グラフィックスオブジェクト」(graphics object) と呼ばれるものを引数として受け取ります。グラフィックスオブジェクトというのは、Graphics というクラスのインスタンスのことで、グラフィックスを描画するためのさまざまな状態やメソッドを持っています。

3.1.4　座標系

キャンバスの上にグラフィックスを描画するためには、その位置を指定する座標を、そのためのメソッドに引数として渡す必要があります。

グラフィックスの描画に使われる座標系は、キャンバスの左上の隅に原点があって、x 軸は右を向いていて、y 軸は下を向いている、というものです。

3.1.5 文字列の描画

グラフィックスオブジェクトは、グラフィックスを描画するさまざまなメソッドを持っています。それらのメソッドの詳細については次の節で説明しますが、ここでは、それらのうちの一例として、文字列を描画するメソッドを紹介しておきたいと思います。

文字列は、drawStringというメソッドを使うことによって描画することができます。このメソッドは、3個の引数を受け取ります。1個目は描画する文字列で、2個目は描画する位置のx座標で、3個目は描画する位置のy座標です。

プログラムの例 ICanvas.java

```
import com.nttdocomo.ui.*;

class DaikichiCanvas extends Canvas {
    public void paint(Graphics g) {
        g.drawString("あなたの運勢は大吉です。", 50, 120);
    }
}

public class ICanvas extends IApplication {
    public void start() {
        Display.setCurrent(new DaikichiCanvas());
    }
}
```

3.1.6 キャンバスの色

キャンバスは、背景色を設定するsetBackgroundというメソッドを持っています（Frameクラスから継承したものです）。このメソッドは、色をあらわす整数を引数として受け取って、その色を背景色として設定します。

キャンバスの上に描画するグラフィックスの色を設定したいときは、グラフィックスオブジェクトが持っているsetColorというメソッドを使います。このメソッドは、色をあらわす整数を引数として受け取って、その色を、グラフィックスを描画するための色として設定します。

プログラムの例 ICanvasColor.java

```
import com.nttdocomo.ui.*;

class ColorCanvas extends Canvas {
    int lightaqua, navy;

    ColorCanvas() {
        lightaqua = Graphics.getColorOfRGB(128, 255, 255);
        navy = Graphics.getColorOfName(Graphics.NAVY);
        setBackground(lightaqua);
    }

    public void paint(Graphics g) {
        g.setColor(navy);
        g.drawString(
            "背景は明るい水色で前景はネイビー。", 20, 120);
    }
}

public class ICanvasColor extends IApplication {
    public void start() {
        Display.setCurrent(new ColorCanvas());
    }
}
```

3.1.7 キャンバスのフォント

drawStringが文字列を描画するときに使うフォントは、グラフィックスオブジェクトが持っているsetFontを使うことによって設定することができます。このメソッドは、引数としてフォントを受け取って、そのフォントを、文字列を描画するためのフォントとして設定します。

プログラムの例 ICanvasFont.java

```
import com.nttdocomo.ui.*;

class FontCanvas extends Canvas {
    Font font;

    FontCanvas() {
        font = Font.getFont(Font.SIZE_LARGE);
    }

    public void paint(Graphics g) {
        g.setFont(font);
        g.drawString("大きなフォント。", 10, 120);
    }
}

public class ICanvasFont extends IApplication {
    public void start() {
        Display.setCurrent(new FontCanvas());
    }
}
```

3.2　グラフィックスの描画

3.2.1　グラフィックスの描画の基礎

描画の必要性が生じたときに自動的に呼び出される paint というメソッドは、 Graphics というクラスのインスタンスを引数として受け取ります。このインスタンスは、グラフィックスを描画するためのさまざまなメソッドを持っています。

この節では、 Graphics クラスのインスタンスが持っている、グラフィックスを描画するメソッドのうちで、主要なものをいくつか紹介したいと思います。

3.2.2　ピクセル

setPixel というメソッドは、ピクセルに前景色を設定します。このメソッドは 2 個の引数を受け取ります。1 個目はピクセルの x 座標で、2 個目はピクセルの y 座標です。

3.2.3　直線

drawLine というメソッドは、直線を描画します。このメソッドは 4 個の引数を受け取ります。1 個目は始点の x 座標、2 個目は始点の y 座標、3 個目は終点の x 座標、4 個目は終点の y 座標です。

3.2.4　長方形

drawRect というメソッドは、長方形を描画します。このメソッドは 4 個の引数を受け取ります。1 個目は左上の頂点の x 座標、2 個目は左上の頂点の y 座標、3 個目は横の長さ、4 個目は縦の長さです。

fillRect と clearRect というメソッドは、長方形の領域を塗りつぶします。それらの相違点は、塗りつぶすために使う色です。 fillRect は前景色を使うのに対して、 clearRect は背景色を使います。ですから、 clearRect を使うことによって、グラフィックスを消去することができます。

なお、 fillRect と clearRect とが受け取る引数は、 drawRect と同じです。

3.2.5　楕円弧

drawArc というメソッドは、楕円弧を描画します。このメソッドは 6 個の引数を受け取ります。1 個目から 4 個目までの引数は drawRect と同じで、それらの引数によって楕円弧が内接する長方形を指定します。5 個目の引数は楕円弧の開始角度です。角度は、時計の 3 時の方向が 0 度で、プラスならば反時計回り、マイナスならば時計回りになります。6 個目の引数は始点から終点までの角度の差です。6 個目の引数についても、プラスならば反時計回り、マイナスならば時計回りになります。

fillArc というメソッドは、楕円弧の領域を扇形に塗りつぶします。このメソッドが受け取る引数は drawArc と同じです。

3.2.6 折れ線

drawPolyline というメソッドは、任意の個数の頂点を直線でつなぐことによって折れ線を描画します。このメソッドは 3 個の引数を受け取ります。1 個目は頂点の x 座標を要素とする配列、2 個目は頂点の y 座標を要素とする配列、3 個目は頂点の個数です。

3.2.7 多角形

fillPolygon というメソッドは、任意の個数の頂点を直線でつなぐことによってできる多角形の領域を塗りつぶします。このメソッドが受け取る引数は drawPolyline と同じです。

プログラムの例　IGraphics.java

```
import com.nttdocomo.ui.*;

class GraphicsCanvas extends Canvas {
    int green, lightgreen;

    GraphicsCanvas() {
        green = Graphics.getColorOfName(Graphics.GREEN);
        lightgreen = Graphics.getColorOfRGB(128, 255, 128);
        setBackground(lightgreen);
    }

    public void paint(Graphics g) {
        g.setColor(green);
        g.setPixel(10, 10);
        g.drawLine(120, 10, 220, 50);
        g.drawRect(10, 60, 100, 40);
        g.fillRect(120, 60, 100, 40);
        g.drawArc(10, 110, 100, 40, 45, 270);
        g.fillArc(120, 110, 100, 40, 45, 270);
        int[] x = { 10, 80, 110, 30};
        int[] y = { 160, 160, 180, 210 };
        g.drawPolyline(x, y, 4);
        int[] x2 = { 120, 190, 220, 140 };
        g.fillPolygon(x2, y, 4);
        g.clearRect(140, 80, 20, 100);
    }
}

public class IGraphics extends IApplication {
    public void start() {
        Display.setCurrent(new GraphicsCanvas());
    }
}
```

3.2.8 画面の大きさ

携帯電話の画面の大きさは、その携帯電話の機種によって異なります。画面の大きさに合わせた適切な位置や大きさでグラフィックスを描画したいという場合は、プログラムの実行時に画面の大きさを取得する必要があります。

画面の大きさは、Display クラスが持っている、次のようなクラスメソッドを呼び出すことによって取得することができます。

getWidth	画面の横幅を返します。
getHeight	画面の高さを返します。

これらのメソッドは引数を受け取りません。また、戻り値の単位はピクセルです。

プログラムの例　IDisplaySize.java

```
import com.nttdocomo.ui.*;

class SizeCanvas extends Canvas {
```

```
    final static int DIAMETER = 20;
    int width, height;

    SizeCanvas() {
        width = Display.getWidth();
        height = Display.getHeight();
    }

    public void paint(Graphics g) {
        g.drawString(" width: " + width, 30, 40);
        g.drawString("height: " + height, 30, 60);
        g.fillArc(width - DIAMETER, height - DIAMETER,
                  DIAMETER, DIAMETER, 0, 360);
    }
}

public class IDisplaySize extends IApplication {
    public void start() {
        Display.setCurrent(new SizeCanvas());
    }
}
```

3.3 キャンバスでのイベント処理

3.3.1 キャンバスでのイベント処理の基礎

パネルを使うiアプリと、キャンバスを使うiアプリとでは、イベントを処理する方法が異なります。パネルを使うiアプリでは、コンポーネントリスナーやキーリスナーなどのイベントリスナーを使ってイベントを処理したわけですが、キャンバスを使うiアプリではイベントリスナーは使えません。

キャンバスを使うiアプリでイベントを処理したいときは、Canvasクラスで定義されている、イベントが発生したときに自動的に呼び出されるメソッドを、そのクラスを継承するクラスでオーバーライドします。

イベントが発生したときに自動的に呼び出されるのは、

```
public void processEvent(int type, int param)
```

というメソッドです。このメソッドをオーバーライドすれば、イベントが発生したときに、その動作が実行されることになります。

3.3.2 イベントのタイプとパラメーター

processEventメソッドは、2個の引数を受け取ります。1個目はイベントのタイプをあらわす整数で、2個目はイベントのパラメーターです。

イベントのタイプをあらわす整数は、Displayクラスの中で、定数として定義されています。たとえば、キーが押されたというイベントのタイプをあらわす整数は、

```
KEY_PRESSED_EVENT
```

という定数を書くことによって記述することができます。

processEventメソッドは、キーが押されたというイベントが発生した場合、2個目の引数、つまりイベントのパラメーターとして、押されたキーをあらわす整数を受け取ります。

キーをあらわす整数は、第2.6節でも説明しましたが、Displayクラスの中で定数として定義されています。たとえば、数字の0はKEY_0、数字の9はKEY_9、アスタリスクはKEY_ASTERISK、……という感じです。

3.3.3 キャンバスでのソフトキーの処理

パネルを使うiアプリでは、普通のキーとソフトキーとでは異なるイベントリスナーを使わないといけなかったわけですが、キャンバスを使うiアプリでは、普通のキーもソフトキーも扱い方は同じです。つまり、どちらの場合もprocessEventが呼び出されるということです。

ソフトキーが押された場合にprocessEventが受け取る2個目の引数は、そのソフトキーをあらわす整数です。ソフトキーをあらわす整数は、Displayクラスの中で、次のように定数として

定義されています。

KEY_SOFT1　左ソフトキー
KEY_SOFT2　右ソフトキー

なお、ソフトキーのラベルを設定する setSoftLabel メソッドは、Frame クラスで定義されているものですので、Canvas クラスにも継承されています。

3.3.4 再描画の実行

何らかのイベントが発生したときに画面上のグラフィックスを変化させたいというときは、プログラムにグラフィックスの再描画を実行させる必要があります。

グラフィックスの再描画は、キャンバスが持っている repaint というメソッドを呼び出すことによって実行させることができます。

repaint には、引数をまったく受け取らないものと、長方形を指定する 4 個の整数を受け取るものとがあります。後者を使うと、長方形で指定した領域だけを再描画することができます。

プログラムの例　ICanvasEvent.java

```
import com.nttdocomo.ui.*;

class EventCanvas extends Canvas {
    final static int RADIUS = 10;
    final static int STEP = 10;
    int width, height, x, y;

    EventCanvas() {
        width = Display.getWidth();
        height = Display.getHeight();
        x = width / 2;
        y = height / 2;
        setSoftLabel(Frame.SOFT_KEY_2, "終了");
    }

    public void processEvent(int type, int param) {
        if (type == Display.KEY_PRESSED_EVENT) {
            switch (param) {
            case Display.KEY_UP:
                moveCharacter(0, -STEP);
                break;
            case Display.KEY_DOWN:
                moveCharacter(0, STEP);
                break;
            case Display.KEY_LEFT:
                moveCharacter(-STEP, 0);
                break;
            case Display.KEY_RIGHT:
                moveCharacter(STEP, 0);
                break;
            case Display.KEY_SOFT2:
                terminate();
                break;
            }
        }
    }

    void moveCharacter(int mx, int my) {
        x += mx;
        y += my;
        repaint();
    }

    public void paint(Graphics g) {
        g.clearRect(0, 0, width, height);
        g.fillArc(x - RADIUS, y - RADIUS,
                  RADIUS * 2, RADIUS * 2, 0, 360);
    }
```

```
    void terminate() {
        IApplication.getCurrentApp().terminate();
    }
}

public class ICanvasEvent extends IApplication {
    public void start() {
        Display.setCurrent(new EventCanvas());
    }
}
```

3.4　ショートタイマー

3.4.1　ショートタイマーの基礎

第 2.7 節で説明したように、パネルを使う場合とキャンバスを使う場合とでは、扱うことのできるタイマーが異なります。キャンバスを使う場合に扱うことのできるのは、「ショートタイマー」(short timer) と呼ばれるタイマーです。

3.4.2　ショートタイマーの生成

ショートタイマーは、

```
com.nttdocomo.ui.ShortTimer
```

というクラスから生成されます（Timer が util にあるのに対して、ShortTimer は ui にあります)。

ショートタイマーの生成には、new ではなくて、getShortTimer という、ShortTimer クラスのクラスメソッドを使います。このメソッドは 4 個の引数を受け取ります。1 個目はタイマーイベントを通知する対象となるキャンバス、2 個目はタイマーを識別するための整数、3 個目はイベントを発生させる時間間隔（単位はミリ秒)、そして 4 個目は、イベントの発生を繰り返すかどうかを指定する真偽値（真だと繰り返し、偽だと 1 回だけ）です。このメソッドは、生成したショートタイマーを戻り値として返します。

3.4.3　ショートタイマーの開始

ショートタイマーは、生成された直後は停止した状態になっています。ショートタイマーの動作を開始させたいときは、それが持っている start というメソッドを呼び出します。このメソッドは、引数を何も受け取りません。

なお、start は、イベントの通知を受け取るキャンバスをカレントフレームにしたのちに呼び出す必要があります。

3.4.4　ショートタイマーのイベント処理

ショートタイマーがイベントを発生させて、それがキャンバスに通知されると、キャンバスは、processEvent メソッドを呼び出します。そのとき、受け取る引数の 1 個目、つまりイベントのタイプをあらわす整数は、

```
Display.TIMER_EXPIRED_EVENT
```

という定数で記述されるものです。そして引数の 2 個目は、イベントを発生させたショートタイマーを識別する整数です。

3.4.5　実行再開時の処理

電話を掛けたり電話が掛かってきたりして i アプリの動作が中断した場合、ショートタイマーは動作を停止します。その場合、通話が終了しても、ショートタイマーの動作は自動的には再開されません。

ショートタイマーの動作は、start メソッドを呼び出すことによって開始させることができるわけですが、i アプリの動作が再開された時点でそれを呼び出すためには、どうすればいいのでしょうか。

IApplication クラスで定義されている、

```
    public void resume()
```

というメソッドは、通話によって中断されていたiアプリの動作が再開されたときに自動的に呼び出されます。

ですから、IApplicationのサブクラスでresumeをオーバーライドして、その中でショートタイマーのstartメソッドを呼び出すことによって、iアプリの動作が再開されたときにショートタイマーの動作を再開させることができます。

プログラムの例　IShortTimer.java

```
import com.nttdocomo.ui.*;

class ShortTimerCanvas extends Canvas {
    final static int RADIUS = 10;
    final static int STEP = 10;
    int width, x, y, direction;
    ShortTimer timer;

    ShortTimerCanvas() {
        width = Display.getWidth();
        x = width / 2;
        y = Display.getHeight() / 2;
        direction = 1;
        timer = ShortTimer.getShortTimer(this, 0, 100, true);
    }

    void start() {
        timer.start();
    }

    public void processEvent(int type, int param) {
        if (type == Display.TIMER_EXPIRED_EVENT) {
            moveCharacter();
        }
    }

    void moveCharacter() {
        if (direction == 1)
            if (x < width - RADIUS)
                x += STEP;
            else
                direction = 2;
        else
            if (x > RADIUS)
                x -= STEP;
            else
                direction = 1;
        repaint();
    }

    public void paint(Graphics g) {
        g.clearRect(0, y - RADIUS, width, y + RADIUS);
        g.fillArc(x - RADIUS, y - RADIUS,
                  RADIUS * 2, RADIUS * 2, 0, 360);
    }

}

public class IShortTimer extends IApplication {
    ShortTimerCanvas canvas;

    public void resume() {
        canvas.start();
    }

    public void start() {
        canvas = new ShortTimerCanvas();
```

```
        Display.setCurrent(canvas);
        canvas.start();
    }
}
```

第 4 章　データの保存

4.1　ブックマーク

4.1.1　ユーザーデータ領域

携帯電話は、「ユーザーデータ領域」(user data area) と呼ばれる永続的な記憶領域を持っています。この記憶領域には、ブックマーク、電話帳、スケジュールなどのデータを登録することができるようになっています。

なお、ユーザーデータ領域にデータを登録する i アプリを実行するためには、その i アプリの ADF の中に、その i アプリがユーザーデータ領域にアクセスするということを宣言する、

```
AccessUserInfo=yes
```

というエントリーを書いておく必要があります。

4.1.2　データを登録するメソッド

電話帳、ブックマーク、スケジュールのデータを携帯電話に登録したいときは、

`com.nttdocomo.system.Bookmark`	ブックマーク
`com.nttdocomo.system.PhoneBook`	電話帳
`com.nttdocomo.system.Schedule`	スケジュール

というクラスのそれぞれが持っている `addEntry` というクラスメソッドを使います。

なお、`addEntry` は、実行時に何らかのエラーが発生した場合は例外を発生させますので、それを呼び出す文は `try` 文の中に書く必要があります。

4.1.3　`Bookmark.addEntry` に渡す引数

`Bookmark` クラスの `addEntry` には、2 個の文字列を引数として渡します。そうすると、引数の 1 個目を URL、2 個目をタイトルとするブックマークが登録されます。

プログラムの例　`IBookmark.java`

```
import com.nttdocomo.ui.*;
import com.nttdocomo.system.*;

class BookmarkPanel extends Panel {
    TextBox url, title;
    Button regist, exit;

    BookmarkPanel() {
        setComponentListener(new ComponentListener() {
            public void componentAction(
                    Component source, int type, int param) {
                if (type == BUTTON_PRESSED)
                    if (source == regist)
                        registToBookmark();
                    else if (source == exit)
                        terminate();
            }
        });
        add(new Label("URL"));
        url = new TextBox("", 36, 1, TextBox.DISPLAY_ANY);
        add(url);
        add(new Label("タイトル"));
        title = new TextBox("", 36, 1, TextBox.DISPLAY_ANY);
        add(title);
        regist = new Button("登録");
```

```
        add(regist);
        exit = new Button("終了");
        add(exit);
    }

    void showException(Exception e) {
        Dialog dialog = new Dialog(Dialog.DIALOG_ERROR,
                                   "例外");
        dialog.setText(e.toString());
        dialog.show();
    }

    void registToBookmark() {
        try {
            Bookmark.addEntry(url.getText(), title.getText());
        } catch (Exception e) {
            showException(e);
        }
    }

    void terminate() {
        IApplication.getCurrentApp().terminate();
    }
}

public class IBookmark extends IApplication {
    public void start() {
        Display.setCurrent(new BookmarkPanel());
    }
}
```

4.2 電話帳

4.2.1 PhoneBook.addEntry に渡す引数

PhoneBook クラスの addEntry というメソッドを呼び出すことによって、ユーザーデータ領域に電話帳のデータを登録することができます。

PhoneBook.addEntry には、5 個の引数を渡します。

引数の 1 個目と 2 個目は文字列です。1 個目は名前として登録されて、2 個目はフリガナとして登録されます。姓と名を区別して登録する携帯電話の場合、それらの文字列は姓として登録されます。

引数の 3 個目と 4 個目は文字列の配列です。3 個目の配列の要素は電話番号として登録されて、3 個目の配列の要素はメールアドレスとして登録されます。

引数の 5 個目は、電話帳のグループを指定するグループ ID です。グループを指定しない場合は、−1 を渡します。

プログラムの例　IPhoneBook.java

```
import com.nttdocomo.ui.*;
import com.nttdocomo.system.*;

class PhoneBookPanel extends Panel {
    TextBox name, kana, phone, mail;
    Button regist, exit;

    PhoneBookPanel() {
        setComponentListener(new ComponentListener() {
            public void componentAction(
                    Component source, int type, int param) {
                if (type == BUTTON_PRESSED)
                    if (source == regist)
                        registToPhoneBook();
                    else if (source == exit)
                        terminate();
            }
```

```
        });
        add(new Label("名前"));
        name = new TextBox("", 36, 1, TextBox.DISPLAY_ANY);
        add(name);
        add(new Label("フリガナ"));
        kana = new TextBox("", 36, 1, TextBox.DISPLAY_ANY);
        add(kana);
        add(new Label("電話番号"));
        phone = new TextBox("", 36, 1, TextBox.DISPLAY_ANY);
        add(phone);
        add(new Label("メールアドレス"));
        mail = new TextBox("", 36, 1, TextBox.DISPLAY_ANY);
        add(mail);
        regist = new Button("登録");
        add(regist);
        exit = new Button("終了");
        add(exit);
    }

    void showException(Exception e) {
        Dialog dialog = new Dialog(Dialog.DIALOG_ERROR,
                                   "例外");
        dialog.setText(e.toString());
        dialog.show();
    }

    void registToPhoneBook() {
        String[] phones = { phone.getText() };
        String[] mails = { mail.getText() };
        try {
            PhoneBook.addEntry(name.getText(), kana.getText(),
                               phones, mails, -1);
        } catch (Exception e) {
            showException(e);
        }
    }

    void terminate() {
        IApplication.getCurrentApp().terminate();
    }
}

public class IPhoneBook extends IApplication {
    public void start() {
        Display.setCurrent(new PhoneBookPanel());
    }
}
```

4.2.2　トラステッド i アプリ

通常の i アプリは、電話帳のデータを登録することはできますが、それを読み込むことはできません。しかし、「トラステッド i アプリ」(trusted i-application) と呼ばれる特殊な i アプリは、電話帳のデータを読み込むことも可能です。

トラステッド i アプリというのは、通常の i アプリには禁止されている、

- 電話帳や発着信履歴などのデータの読み込み
- i アプリのダウンロード元サーバー以外のサーバーとの通信
- i モードメールとの連携

などの機能を利用することのできる i アプリのことです。ただし、トラステッド i アプリを開発することができるのは公式アプリの開発者のみです。

4.3 スケジュール

4.3.1 Schedule.addEntry に渡す引数

Schedule クラスの addEntry というメソッドを呼び出すことによって、ユーザーデータ領域にスケジュールのデータを登録することができます。

Schedule.addEntry には、3 個の引数を渡します。引数の 1 個目はスケジュールの内容として登録される文字列です。引数の 2 個目は、スケジュールの日付と時刻をあらわす、

```
com.nttdocomo.util.ScheduleDate
```

というクラスのインスタンスです。そして引数の 3 個目は、アラームを鳴らすかどうかを指定する真偽値です。鳴らしたい場合は真を渡し、鳴らしたくない場合は偽を渡します。

4.3.2 日付と時刻をあらわすオブジェクト

スケジュールの日付と時刻をあらわす ScheduleDate クラスのインスタンスを生成するときは、そのクラスのコンストラクタに、日付と時刻のタイプをあらわす整数を引数として渡す必要があります。その整数は、 ScheduleDate クラスの中で次のような定数として定義されています。

ONETIME	1 回限り
DAILY	毎日
WEEKLY	毎週
MONTHLY	毎月
YEARLY	毎年

ScheduleDate クラスのインスタンスに、日付や時刻などの数値を設定したいときは、それが持っている set というメソッドを呼び出します。このメソッドは、2 個の引数を受け取ります。引数の 1 個目は数値を設定する対象を指定する整数で、引数の 2 個目は設定する数値（整数）です。

数値を設定する対象を指定する整数は、

```
java.util.Calendar
```

というクラスの中で定数として定義されています。そこで定義されている定数としては、たとえば次のようなものがあります。

YEAR	年（西暦）
MONTH	月 $(0-11)$
DAY_OF_MONTH	日 $(1-31)$
HOUR_OF_DAY	時間 $(0-23)$
MINUTE	分 $(0-59)$

プログラムの例　ISchedule.java

```
import com.nttdocomo.ui.*;
import com.nttdocomo.system.*;
import com.nttdocomo.util.*;
import java.util.*;

class SchedulePanel extends Panel {
    TextBox year, month, day, hour, minute, content;
    Button regist, exit;

    SchedulePanel() {
        setComponentListener(new ComponentListener() {
            public void componentAction(
                    Component source, int type, int param) {
                if (type == BUTTON_PRESSED)
                    if (source == regist)
                        registToSchedule();
                    else if (source == exit)
                        terminate();
            }
        });
        add(new Label("年（西暦）"));
```

```
        year = new TextBox("", 36, 1, TextBox.DISPLAY_ANY);
        add(year);
        add(new Label("月 (1-12)"));
        month = new TextBox("", 36, 1, TextBox.DISPLAY_ANY);
        add(month);
        add(new Label("日 (1-31)"));
        day = new TextBox("", 36, 1, TextBox.DISPLAY_ANY);
        add(day);
        add(new Label("時 (0-23)"));
        hour = new TextBox("", 36, 1, TextBox.DISPLAY_ANY);
        add(hour);
        add(new Label("分 (0-59)"));
        minute = new TextBox("", 36, 1, TextBox.DISPLAY_ANY);
        add(minute);
        add(new Label("内容"));
        content = new TextBox("", 36, 1, TextBox.DISPLAY_ANY);
        add(content);
        regist = new Button("登録");
        add(regist);
        exit = new Button("終了");
        add(exit);
    }

    void showException(Exception e) {
        Dialog dialog = new Dialog(Dialog.DIALOG_ERROR,
                                   "例外");
        dialog.setText(e.toString());
        dialog.show();
    }

    int parseInt(TextBox textbox) {
        int n = 0;
        try {
            n = Integer.parseInt(textbox.getText());
        } catch (NumberFormatException e) {
            showException(e);
        }
        return n;
    }

    ScheduleDate createScheduleDate() {
        ScheduleDate date =
            new ScheduleDate(ScheduleDate.ONETIME);
        date.set(Calendar.YEAR, parseInt(year));
        date.set(Calendar.MONTH, parseInt(month) - 1);
        date.set(Calendar.DAY_OF_MONTH, parseInt(day));
        date.set(Calendar.HOUR_OF_DAY, parseInt(hour));
        date.set(Calendar.MINUTE, parseInt(minute));
        return date;
    }

    void registToSchedule() {
        try {
            Schedule.addEntry(content.getText(),
                              createScheduleDate(), false);
        } catch (Exception e) {
            showException(e);
        }
    }

    void terminate() {
        IApplication.getCurrentApp().terminate();
    }
}

public class ISchedule extends IApplication {
    public void start() {
```

```
        Display.setCurrent(new SchedulePanel());
    }
}
```

4.4 スクラッチパッド

4.4.1 スクラッチパッドの基礎

iモードに対応した携帯電話は、ユーザーデータ領域とは別に、「スクラッチパッド」(scratch pad) と呼ばれる永続的な記憶領域を持っています。

トラステッドiアプリではない通常のiアプリは、ユーザーデータ領域に対してデータを書き込むことはできますが、そこからデータを読み込むことはできません。それに対して、スクラッチパッドに対しては、通常のiアプリでも書き込みと読み込みの両方が可能です。ただし、読み込むことができるのは、同じiアプリが書き込んだデータだけです。言い換えれば、スクラッチパッドを使って複数のiアプリがデータを共有することはできない、ということです。

スクラッチパッドは、それを最大 16 個までの部分に分割して利用することができます。

4.4.2 使用する記憶領域の大きさの宣言

スクラッチパッドを利用するiアプリを実行するためには、そのiアプリの ADF の中に、そのiアプリが使用するスクラッチパッドの記憶領域の大きさを宣言する、

`SPsize=`大きさ`,・・・`

というエントリーを書いておく必要があります。エントリー値は、スクラッチパッドを分割したそれぞれの記憶領域の大きさ（単位はバイト）をコンマで区切って並べたものです（コンマの前後に空白を書くことはできません)。たとえば、

```
SPsize=2000,3000,4000
```

というエントリーを書くことによって、スクラッチパッドを、2000 バイト、3000 バイト、4000 バイトの 3 個の記憶領域に分割して利用する、ということを宣言することができます。

なお、スクラッチパッドのそれぞれの記憶領域には、ADF の中に大きさが書かれている順番のとおりに、0 番から始まる番号が与えられます。

4.4.3 データストリームの基礎

スクラッチパッドに対する読み書きは、「データストリーム」(data stream) と呼ばれるオブジェクトが持っているメソッドを呼び出すことによって実行することができます。

データストリームは、データを読み込むために使われるものと、データを書き込むために使われるものに分類されます。読み込みのためのデータストリームは「データ入力ストリーム」(data input stream) と呼ばれ、書き込みのためのデータストリームは「データ出力ストリーム」(data output stream) と呼ばれます。

4.4.4 データストリームの生成

データ入力ストリームとデータ出力ストリームは、それぞれ、次のクラスから生成されます。

`java.io.DataInputStream` データ入力ストリーム
`java.io.DataOutputStream` データ出力ストリーム

これらのクラスのインスタンスを生成したいときは、普通、`new`ではなくて、

`javax.microedition.io.Connector`

というクラスが持っている、

`openDataInputStream` データ入力ストリームを生成する。
`openDataOutputStream` データ出力ストリームを生成する。

というクラスメソッドを使います。これらのメソッドは、読み書きの対象となるものを指定する URL を引数として受け取って、それに対する読み書きを実行するためのデータストリームを戻り値として返します。これらのクラスメソッドは、データストリームを生成することができなかった場合は、`IOException`というクラスの例外を発生させます。

スクラッチパッドの記憶領域は、その記憶領域の番号を n とすると、

scratchpad:///n

という URL によって指定されます。

4.4.5　データの読み込み

データ入力ストリームは、データを読み込むためのメソッドとして、次のようなものを持っています。

readInt	4 バイトだけ読み込んで int で返す。
readBoolean	1 バイトだけ読み込んで boolean で返す。
readUTF	最後まで読み込んで string で返す。

これらのメソッドは、引数は何も受け取らないで、読み込んだデータを戻り値として返します。読み込みができなかった場合は、 IOException の例外を発生させます。

4.4.6　データの書き込み

データ出力ストリームは、データを書き込むためのメソッドとして、次のようなものを持っています。

writeInt	int のデータ（4 バイト）を書き込む。
writeBoolean	boolean のデータ（1 バイト）を書き込む。
writeUTF	string のデータを書き込む。

これらのメソッドは、引数としてデータを受け取って、書き込みの対象となっているものにその引数を書き込みます。書き込みができなかった場合は、 IOException の例外を発生させます。

4.4.7　書き込みの後始末

ストリームを使って読み書きを実行した場合は、かならず、その処理の後始末を実行する必要があります。

読み書きの後始末は、データストリームが持っている close というメソッドを呼び出すことによって実行することができます。このメソッドは、引数を何も受け取りません。後始末ができなかった場合は、 IOException の例外を発生させます。

プログラムの例　IScratchPad.java

```
import com.nttdocomo.ui.*;
import javax.microedition.io.*;
import java.io.*;

class ScratchPadPanel extends Panel {
    void setCurrent(Panel panel) {
        Display.setCurrent(panel);
    }

    void showException(Exception e) {
        Dialog dialog = new Dialog(Dialog.DIALOG_ERROR,
                                   "例外");
        dialog.setText(e.toString());
        dialog.show();
    }

    void terminate() {
        IApplication.getCurrentApp().terminate();
    }
}

class SelectionPanel extends ScratchPadPanel {
    Button write, read, exit;

    SelectionPanel() {
        setComponentListener(new ComponentListener() {
            public void componentAction(
```

```
                    Component source, int type, int param) {
                if (type == BUTTON_PRESSED)
                    if (source == write)
                        setCurrent(new WritePanel());
                    else if (source == read)
                        setCurrent(new ReadPanel());
                    else if (source == exit)
                        terminate();
            }
        });
        write = new Button("書き込み");
        add(write);
        read = new Button("読み込み");
        add(read);
        exit = new Button("終了");
        add(exit);
    }
}

class WritePanel extends ScratchPadPanel {
    TextBox textbox;
    Button write, cancel;

    WritePanel() {
        setComponentListener(new ComponentListener() {
            public void componentAction(
                    Component source, int type, int param) {
                if (type == BUTTON_PRESSED)
                    if (source == write) {
                        writeScratchPad();
                        setCurrent(new SelectionPanel());
                    } else if (source == cancel)
                        setCurrent(new SelectionPanel());
            }
        });
        textbox = new TextBox(
            "", 36, 8, TextBox.DISPLAY_ANY);
        add(textbox);
        write = new Button("書き込む");
        add(write);
        cancel = new Button("キャンセル");
        add(cancel);
    }

    void writeScratchPad() {
        try {
            DataOutputStream dos =
                Connector.openDataOutputStream(
                    "scratchpad:///0");
            dos.writeUTF(textbox.getText());
            dos.close();
        } catch (IOException e) {
            showException(e);
        }
    }
}

class ReadPanel extends ScratchPadPanel {
    TextBox textbox;
    Button back;

    ReadPanel() {
        setComponentListener(new ComponentListener() {
            public void componentAction(
                    Component source, int type, int param) {
                if (type == BUTTON_PRESSED)
                    if (source == back)
```

```
                        setCurrent(new SelectionPanel());
                }
            });
            textbox = new TextBox(
                "", 36, 8, TextBox.DISPLAY_ANY);
            textbox.setEditable(false);
            add(textbox);
            back = new Button("戻る");
            add(back);
            readScratchPad();
        }

        void readScratchPad() {
            try {
                DataInputStream dis =
                    Connector.openDataInputStream(
                        "scratchpad:///0");
                String data = dis.readUTF();
                textbox.setText(data);
                dis.close();
            } catch (IOException e) {
                showException(e);
            }
        }
}

public class IScratchPad extends IApplication {
    public void start() {
        Display.setCurrent(new SelectionPanel());
    }
}
```

参考文献

[muray,2004] muray、志村拓、山口威、山田英樹ほか、『i モード Java プログラミング・FOMA 対応版』、アスキー、2004、ISBN 4-7561-4503-5。

[NTT ドコモ,2005] 株式会社 NTT ドコモ、『i アプリコンテンツ開発ガイド for DoJa–4.0/4.0LE ～詳細編～第 2.00 版』、2005。

[布留川,2005] 布留川英一、『i アプリゲーム開発テキストブック』、毎日コミュニケーションズ、2005、ISBN 4-8399-1736-1。

[山崎,2001] 山崎由喜憲、exilis、『i アプリの作り方』、ソフトバンクパブリッシング、2001、ISBN 4-7973-1654-3。

[ユーエヌアイ,2001] 有限会社ユーエヌアイ研究所、『i アプリ API リファレンス』、翔泳社、2001、ISBN 4-7981-0064-1。

索引