

2. i アプリ helloWorld

ディスプレイに「HELLO」と表示するだけの簡単なiアプリを作成してみます。

- (1) 「i ppliTool for DoJa-3.5(FOMA)」を起動します。

スタート すべてのプログラム i ppli Development Kit for DoJa-3.5 i ppliTool for DoJa-3.5(FOMA)

- (2) 「プロジェクト新規作成」をクリックします。

<http://akioneer.sakura.ne.jp/kuu/iapl2-1.jpg>

- (3) プロジェクト名を入力します。

ゲーム名をプロジェクト名として、半角英数字で入力します。
今回はサンプルということでgame01とします。
「作成」ボタンをクリックすると、「C:¥iDKDoJa3.5¥apps」フォルダ内に
「game01」フォルダが作成されます。

<http://akioneer.sakura.ne.jp/kuu/iapl2-2.jpg>

- (4) 「C:\iDKDoJa3.5¥apps¥game01」フォルダに移動します。

「game01」フォルダ内は以下のような構成になっています。

<http://akioneer.sakura.ne.jp/kuu/iapl2-3.jpg>

(注)

拡張子の表示が許可されているかを確認してください。
1. 上の画面のときに「ツール」「フォルダオプション」「表示」タブを開きます。
2. 詳細設定欄にある「登録されている拡張子は表示しない」にチェックが
付いていないかを確認します。
チェックが付いている場合はチェックを外し、
「すべてのフォルダに適用」をクリックします。
3. ついでに「縮小版をキャッシュしない」にチェックが付いているかを確認します。
チェックが付いていない場合はチェックを付け、
「すべてのフォルダに適用」をクリックします。

下記のようになっていれば OK です。

<http://akioneer.sakura.ne.jp/kuu/iapl2-4.jpg>

- (5) ソースの作成をします。

ソースとはプログラムが記述されたファイルのことです。
ソースはプロジェクトフォルダの「src」フォルダに保存しますので、
「C:¥iDKDoJa3.5¥apps¥game01¥src」フォルダに移動してください。
そのフォルダ内でテキストファイルを2つ作成します。
テキストファイルの名前はKicker.javaとMainCanvas.javaに変更してください。

下記のようになっていれば OK です。

<http://akioneer.sakura.ne.jp/kuu/iapl2-5.jpg>

下記の内容をコピーし、Kicker.java と MainCanvas.java に貼り付け、上書き保存してください。

DoJa の Kicker.java

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

/**
 * ゲームの開始処理を行うクラスです。
 *
 * @author RGB
 * @version 1.0
 */
public class Kicker extends IApplication {

    /**
     * メインのキャンバスです。
     */
    private MainCanvas mc;

    /**
     * アプリケーションが起動したら呼ばれるメソッドです。
     */
    public void start() {
        mc = new MainCanvas(this);
        Display.setCurrent(mc);
        mc.start();
    }

}
```

DoJa の MainCanvas.java

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

/**
 * メインのキャンバスです。
 *
 * @author RGB
 * @version 1.0
 */
public class MainCanvas extends Canvas implements Runnable {

    // -----
    // 共通部：特に変更する必要はありません。
    // -----

    /**
     * 使用領域の幅です。
     */
    public static final int AREA_WIDTH = 240;

    /**
     * 使用領域の高さです。
     */
    public static final int AREA_HEIGHT = 240;

    /**
     * 画面の幅です。
     */
    public static final int DISPLAY_WIDTH = Display.getWidth();
```

```

/**
 * 画面の高さです。
 */
public static final int DISPLAY_HEIGHT = Display.getHeight();

/**
 * キー処理で使用する定数で、「上キー」を表します。
 */
public static final int KEY_UP = Display.KEY_UP;

/**
 * キー処理で使用する定数で、「下キー」を表します。
 */
public static final int KEY_DOWN = Display.KEY_DOWN;

/**
 * キー処理で使用する定数で、「右キー」を表します。
 */
public static final int KEY_RIGHT = Display.KEY_RIGHT;

/**
 * キー処理で使用する定数で、「左キー」を表します。
 */
public static final int KEY_LEFT = Display.KEY_LEFT;

/**
 * キー処理で使用する定数で、「決定キー」を表します。
 */
public static final int KEY_SELECT = Display.KEY_SELECT;

/**
 * 押されたキーの値です。
 */
private int keyeventPressed = 0;

/**
 * このキャンバスのグラフィックスオブジェクトです。
 */
private Graphics g = null;

/**
 * 現在のフレームカウントです。
 */
private int frameCount = 0;

/**
 * 現在のシーンです。
 */
private int scene = 0;

/**
 * シーンの切り替え状態を管理します。
 */
private boolean sceneSwitch = false;

/**
 * 設定された IApplication オブジェクトです。
 */
private IApplication kicker;

/**
 * ゲーム用のスレッドです。
 */
private Thread th;

/**
 * コンストラクタです。
 */
public MainCanvas(IApplication kicker) {
    this.kicker = kicker;
}

/**
 * キャンバス処理を開始します。
 */
public void start() {
    repaint();
}

```

```

/**
 * 画面の描画要求時に呼ばれます。
 *
 * @param g グラフィックスオブジェクトが渡されます。
 */
public void paint(Graphics g) {
    if(th == null) {
        this.g = g;
        th = new Thread(this);
        th.start();
        Thread.currentThread().yield();
    }
}

/**
 * キーイベント取得メソッドです。
 *
 * @param type イベントのタイプが渡されます。
 * @param param イベントのパラメータが渡されます。
 */
public void processEvent(int type, int param) {
    if(type == Display.KEY_PRESSED_EVENT) {
        keyeventPressed = param;
    }
    else if(type == Display.KEY_RELEASED_EVENT) {
        keyeventPressed = 0;
    }
}

/**
 * 指定されたキーが、押された状態であるかを判定します。
 *
 * @param key 検索するキーを設定します。
 */
public boolean keyPressing(int key) {
    if(keyeventPressed == key) {
        keyeventPressed = 0;
        return true;
    }
    else {
        return false;
    }
}

/**
 * 指定されたキーが、キーリピート状態であるかを判定します。
 *
 * @param key 検索するキーを設定します。
 */
public boolean keyRepeating(int key) {
    if(keyeventPressed == key) {
        return true;
    }
    else {
        return false;
    }
}

/**
 * アプリを終了します。
 */
public void exit() {
    Application.getCurrentApp().terminate();
}

/**
 * 画像を読み込みます。
 *
 * @param name 画像名称を設定します。
 * @return 読み込んだ画像を返します。
 */
public Image loadImage(String name) {
    MediaImage mImg = null;
    Image img = null;
    try {
        mImg = MediaManager.getImage("resource:/// " + name + ".gif");
    }
}

```

```

        mImg.use();
        img = mImg.getImage();
    }
    catch(ConnectionException ce) {
        if(mImg != null) {
            mImg.unuse();
            mImg.dispose();
        }
        throw new RuntimeException(ce.getMessage());
    }
    return img;
}

/**
 * 描画開始時・描画終了時の処理を行います。
 * @param state 描画開始は true、描画終了は false を設定します。
 */
public void draw(boolean state) {
    if(state) {
        g.lock();
    }
    else {
        g.unlock(false);
    }
}

/**
 * 描画に使用する色を設定します。
 * @param red 赤要素の輝度を指定します (0 ~ 255 )
 * @param green 緑要素の輝度を指定します (0 ~ 255 )
 * @param blue 青要素の輝度を指定します (0 ~ 255 )
 */
public void setColor(int red, int green, int blue) {
    g.setColor(Graphics.getColorOfRGB(red, green, blue));
}

/**
 * 矩形領域を塗りつぶします。
 * @param x 矩形の左上の X 座標を指定します。
 * @param y 矩形の左上の Y 座標を指定します。
 * @param width 矩形の幅を指定します。
 * @param height 矩形の高さを指定します。
 */
public void fillRect(int x, int y, int width, int height) {
    g.fillRect(x, y, width, height);
}

/**
 * 文字列を描画します。
 * @param str 描画する文字列を指定します。
 * @param x X 座標を指定します。
 * @param y Y 座標を指定します。
 */
public void drawString(String str, int x, int y) {
    g.drawString(str, x, y + Font.getDefaultFont().getBBoxHeight(str));
}

/**
 * イメージを描画します。
 * @param image 描画するイメージオブジェクトを指定します。
 * @param x X 座標を指定します。
 * @param y Y 座標を指定します。
 */
public void drawImage(Image image, int x, int y) {
    g.drawImage(image, x, y);
}

/**
 * ゲーム用スレッドの処理です。
 */
public void run() {

```

```

        g = getGraphics();
        try {
            changeScene(SC_INIT);
            while(true) {
                controlScene();
                Thread.currentThread().sleep(100);
                frameCount ++;
            }
        } catch(Throwable t) {
            System.out.println(t.getMessage());
            exit();
        }
    }
}

```

```

/**
 * シーンを変更します。
 *
 * @param scene 変更するシーンを設定します。
 */
public void changeScene(int scene) {
    this.scene = scene;
    sceneSwitch = true;
}

```

```

/**
 * シーンの初期化処理をすべきかを判断します。
 *
 * @return シーンが変更された直後は true を返します。
 *         それ以降は false を返します。
 */
public boolean firstScene() {
    boolean res = sceneSwitch;
    sceneSwitch = false;
    return res;
}

```

```

// -----
//
// 実装部：ゲームごとに変更します。
//
// -----

```

```

/** 「初期化」シーンです。*/
public static final int SC_INIT = 0;

/** 「タイトル」シーンです。*/
public static final int SC_TITLE = 1;

/** 「プレイ」シーンです。*/
public static final int SC_PLAY = 2;

/** 「結果」シーンです。*/
public static final int SC_RESULT = 3;

```

```

/**
 * シーンごとの処理を制御します。
 */
public void controlScene() {
    switch(scene) {
        case SC_INIT : sceneInit();
        break;
        case SC_TITLE : sceneTitle();
        break;
        case SC_PLAY : scenePlay();
        break;
        case SC_RESULT : sceneResult();
        break;
    }
}

```

```

/**
 * 「初期化」シーンを制御します。
 */

```

```

public void sceneInit() {
    if(firstScene()) {
    }

    draw(true);

    setColor(0, 0, 255);
    fillRect(0, 0, AREA_WIDTH, AREA_HEIGHT);

    setColor(255, 0, 0);
    drawString("HELLO", 10, 20);

    draw(false);
}

/**
 * 「タイトル」シーンを制御します。
 */
public void sceneTitle() {
}

/**
 * 「プレイ」シーンを制御します。
 */
public void scenePlay() {
}

/**
 * 「結果」シーンを制御します。
 */
public void sceneResult() {
}

}

```

(6) アプリ情報を設定します。

i アプリの情報を設定するには「ADF 設定」をクリックします。

<http://akioneer.sakura.ne.jp/kuu/iapl2-6.jpg>

以下の項目に値を設定します。

項目名	設定値	説明
AppName	GAME01	アプリの名称を設定します。好きな名前でも構いません。今回はこの名前にします。
AppClass	Kicker	最初に起動されるクラスです。先ほど説明した「Kicker」にします。左欄からコピーして貼り付けする場合は「Kicker」の右側に余分なスペースが入る場合がありますのでご注意ください。

<http://akioneer.sakura.ne.jp/kuu/iapl2-7.jpg>

最後に必ず「設定」をクリックします。

(7) 「ビルド」をクリックします。

<http://akioneer.sakura.ne.jp/kuu/iapl2-8.jpg>

ビルドをすると、「bin」フォルダにプログラム実行ファイル
(.jam ファイルと .jar ファイル) が作成されます。
エラーが出た場合は、もう一度 (4) からやり直してください。

(8) 「起動」をクリックします。

<http://akioneer.sakura.ne.jp/kuu/iapl2-9.jpg>

ディスプレイが青く塗りつぶされ、赤色の文字で「HELLO」と表示されたら OK です。

<http://akioneer.sakura.ne.jp/kuu/iapl2-10.jpg>

3. i アプリくうちゃん