iアプリ Java
ゲームプログラミング

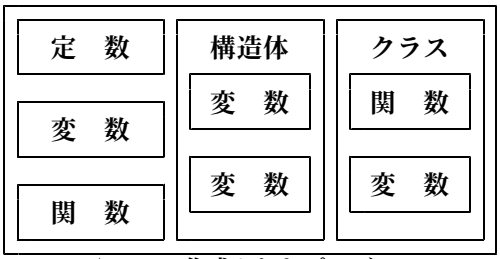
○　総集編　シューティングゲームの作成　○

Java

Javaとは、Sun Microsystems社が開発したオブジェクト指向プログラミング言語で、その実行環境も含めてJavaと呼びます。
Javaは、どのような環境でも実行できることを目標としており、仮想マシンが存在すれば、OSに依存することなくコードを実行できることが特徴です。OSに依存しないコードが書けることから、サーバー用のアプリケーションや、組み込みソフトの実行環境として携帯電話に普及しています。

Javaの特徴

Javaは、C++の優れた点だけを引き継ぎ、オブジェクト指向に不要なものや危険性の高いものを削り、シンプルで安全性の高い言語を目指しています。たいていの言語が文字とファイルの入出力しか言語仕様に含んでいないのに対し、Javaでは画像、GUI、ネットワークなどの膨大なクラスライブラリを言語仕様として標準サポートしています。

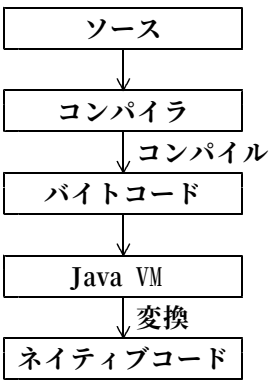


▲C++で作成されたプログラム
(構造化＋オブジェクト指向プログラミング)

▲Javaで作成されたプログラム
(完全なオブジェクト指向プログラミング)

Javaの仕組み

Javaで記述されたプログラムは、コンパイラによりバイトコードにコンパイルされます。バイトコードは、JavaVM(Java Virtual Machine)という仮想マシン用のコードです。JavaVMは、バイトコードを１つ１つ環境に合うように解釈しながら実行します(バイトコードインタプリタ方式)。直接実行できるネイティブコードに比べ実行速度は劣りますが、Windows, Macintosh, Linuxなどさまざまな環境のものが提供されており、同じJavaプログラムをOSやCPU、表示機能、ネットワークなどの違いを超えて、異なる環境で動かすことができるようになっています。



iアプリ

ｉアプリは、NTTドコモのｉモード端末機でダウンロードして利用できるアプリケーションです。ゲームや地図情報、株価情報などのコンテンツが携帯電話で楽しめます。
携帯電話は、PCと比較してハードウェアの性能上、厳しい制約があり、通常のJavaより機能は削られています。

ｉアプリ開発環境

ｉアプリを作成するには、最低限、以下のソフトウェアが必要になります。これは無償で入手することができます。

・Java2 SDK Standard Edition (1.3以降)
・i-appli Development Kit for DoJa

JDK

JDKは、Java言語でプログラミングを行なう際に必要な最低限のソフトウェアのセットです。Javaの開発元であるSun Microsystems社が開発、配布しています。コンパイラ、デバッガ、クラスライブラリ、JavaVMなどが含まれています。

JDKはOSごとに専用のものが存在し、どれも無償でダウンロードして入手することができます。なかでも、Windows, Linux, SolarisのJDKは、Sun Microsystems社(http://java.sun.com/)が提供しています。Macintoshや商用UNIXなども、OSのメーカーが提供しています。

iαppli Development Kit for DoJa (DoJa SDK)

DoJaとは、NTTドコモが提供するｉアプリ用のJava拡張ライブラリです。スクラッチパッドや携帯電話のボタンの処理機能など、携帯電話を制御するためのライブラリが定義されています。

DoJaは、ｉアプリ対応携帯電話では共通して使用できますが、KDDI系のezplusやボーダフォンのＶアプリでは使用できません。また、901i用のDoJaは900i用のDoJaを拡張したものとなっており、901iで追加されたライブラリは900iでは使用することができません。

DoJaは、NTTドコモのホームページ(http://www.nttdocomo.co.jp/p_s/imode/java/)からダウンロードすることができます。

Eclipse

Eclipseは、統合ソフトウェア開発環境(IDE)の一つで、Java開発者を中心に急速に普及しています。ソフトウェア開発ツールの共通プラットフォームの標準になるといわれています。ソースコードが公開されており、無償で入手・改変・再配布できるようになっています。大手ソフトウェアベンダの中には、自社の開発ツール製品にEclipseを組み込み、Eclipseに追加する形で自社独自部分を提供するという形の製品を発売するところも現れています。

携帯電話の仕様

携帯電話の各機種の仕様は、以下のようになっています。

機種	ﾌﾟﾛﾌｧｲﾙ	JAR容量	ｽｸﾗｯﾁﾊﾟｯﾄﾞ	Canvasｸﾗｽ解像度	浮動小数点対応
505i	DoJa3.0	30KB	200KB	240×240	×
506i	DoJa3.0	30KB	200KB	240×240	×
700i	DoJa4.0LE	30KB	200KB	240×240	×
900i	DoJa3.5	100KB	400KB	240×240	×
901i	DoJa4.0	100KB	400KB	240×240	○

容量と解像度は、２０年前のファミコン程度です。実行速度に関しても、PCとは比べものにならないぐらい非力なCPUを搭載しています(Pentium4の数十分の一程度)。さらに、その上でJavaVMを稼働し、バイトコードを解釈しながら実行するので、とても処理が重くなります。PC上で動かす場合はPCのCPUが使用されるので快適に動きますが、実機では遅くて使い物にならない、ということはよくあります。

変数

Javaの変数には、以下の８つの型があります。

型名	説明	値の範囲
boolean	真偽値を格納	true(成り立つ), false(成り立たない)
char	unicode文字を格納	\u0000 ～ \uffff
byte	１バイト整数	-128 ～ 127
short	２バイト整数	-32,768 ～ 32,767
int	４バイト整数	-2,147,483,648 ～ 2,147,483,647
long	８バイト整数	-92,233,372,036,854,775,808 ～ 92,233,372,036,854,775,807
float	単精度浮動小数点(４バイト)	±10の±38乗(有効桁数9桁)
double	倍精度浮動小数点(８バイト)	±10の±308乗(有効桁数16桁)

Javaで使用するおもな演算子は、以下のものです。

演算子名	記号	名　前	記述例	機　能
算術演算子	++	インクリメント	a++	値に1を加える
	--	デクリメント	a--	値から1を引く
	+	正符号	+a	正符号
	-	負符号	-a	負符号
	*	乗算	a * b	乗算
	/	除算	a / b	除算
	%	剰余	a % b	割り算の余りを求める。
	+	加算、文字列の連結	a + b	加算、両辺が文字列なら連結する
	-	減算	a - b	減算

演算子名	記号	名　前	記述例	機　能
ビット演算子	<<	左シフト	a << b	ビットを左にシフトする
	>>	算術右シフト	a >> b	ビットを右にシフトする
	>>>	論理右シフト	a >>> b	ビットを右にシフトする。先頭は0で埋められる

演算子名	記号	名　前	記述例	機　能
関係演算子	==	等値	a == b	両辺が等しければtrue、そうでなければfalseとなる
	!=	非等値	a != b	両辺が等しくなければtrue、そうでなければfalseとなる
	<	左不等	a < b	左辺より右辺が大きければtrue、そうでなければfalseとなる
	<=	等価左不等	a <= b	左辺が右辺以上ならtrue、そうでなければfalseとなる
	>	右不等	a > b	左辺より右辺が小さければtrue、そうでなければfalseとなる
	>=	等価右不等	a >= b	左辺が右辺以下ならtrue、そうでなければfalseとなる

演算子名	記号	名　前	記述例	機　能
論理演算子	&&	条件AND	a && b	両辺がtrueのときのみtrueとなる
	¦¦	条件OR	a ¦¦ b	少なくとも一方がtrueならtrueとなる

演算子名	記号	名　前	記述例	機　能
代入演算子	=	単純代入	a = b	右辺の式の値を左辺に代入する
	/=	除算代入	a /= b	a = a / bと同じ
	%=	余剰代入	a %= b	a = a % bと同じ
	*=	乗算代入	a *= b	a = a * bと同じ
	+=	加算代入	a += b	a = a + bと同じ
	-=	減算代入	a -= b	a = a - bと同じ

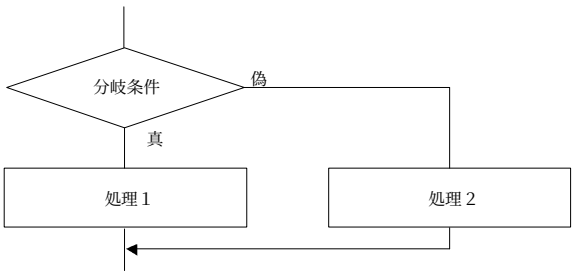
制御文

制御文とは、プログラムの流れを制御するための文です。Javaでは上から順番にプログラムが実行されますが、制御文を使うことにより、プログラムを分岐させたり、反復したりすることができます。
分岐させる制御文としてif-else、switch-caseがあり、反復(ループ)させる制御文としてwhile、do-while、forがあります。

1．if-else

if-else文は、条件が成り立つ場合は「処理１」が行われ、成り立たない場合は「処理２」が実行されます。

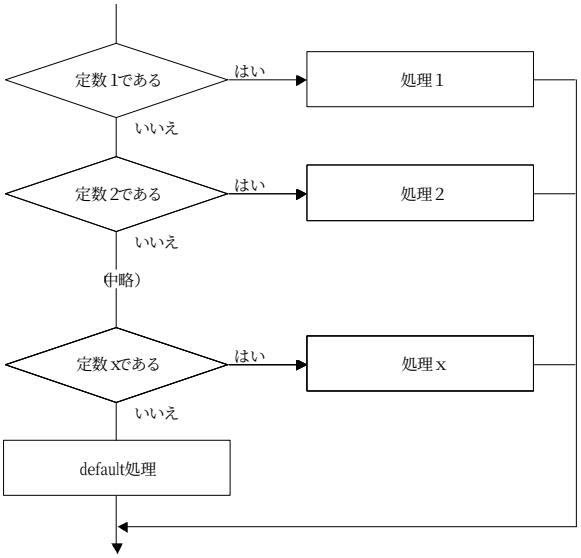
```
if(条件式) {
    条件式を満たしたときに実行する処理
} else {
    条件式を満たさなかったときに実行する処理
}
```



2．switch-case

switch文は、値によって実行する処理を多分岐させることができます。

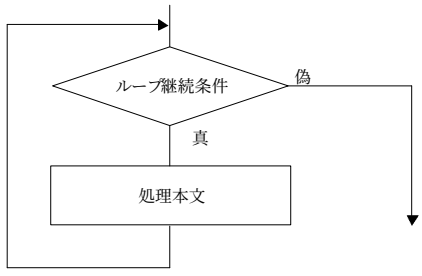
```
switch(変数) {
case 定数式１:
    処理１
    break;

case 定数式２:
    処理２
    break;
  (中略)
case 定数式ｘ:
    処理ｘ
    break;

default:
    すべての定数式を満たさなかったときの処理
    break;
}
```



3．while

while文は、ループに入る前に条件判定を行い、条件が成り立つ間処理を反復します。

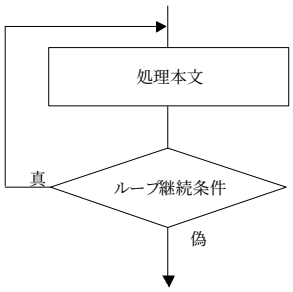
```
while(ループ継続条件) {
    処理本文
}
```



４．do-while

do-while文は、ループの最後に条件判定を行い、条件が成り立つ間処理を反復します。

```
do {
    処理本文
} while(ループ継続条件);
```

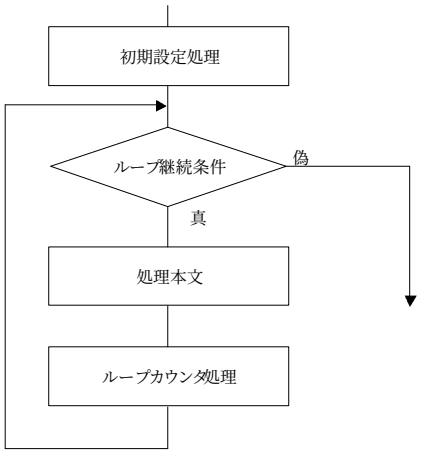


５．for

for文は、条件が成り立つ間、処理を反復させることができます。

```
for(初期設定処理; ループ継続条件; ループカウンタ処理) {
    反復させたい処理
}
```

for文は、指定した回数だけ反復させたいときに使用します。反復回数を管理するために、ループカウンタという変数を使用します。



配列

データは変数に入れて扱いますが、１つのデータにつき１つの変数を宣言する必要があります。しかし、この方法ではプログラム作成上、困難になる場合があります。
たとえば、複数の敵がいて、ぞれぞれの座標が、teki_x1, teki_y1, teki_x2, teki_y2…という変数に格納されているとしましょう。

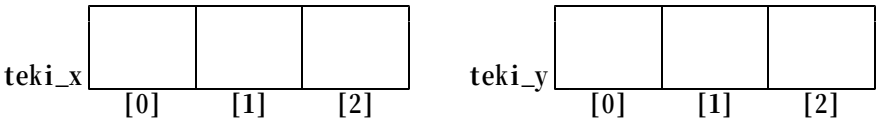
```
int   teki_x1 =  0, teki_y1 =  0;   // 敵No.1
int   teki_x2 = 10, teki_y2 =  0;   // 敵No.2
int   teki_x3 = 50, teki_y3 = 10;   // 敵No.3
```

これは、以下のように個々の変数を用意していることになります。

0	0	10	0	50	10
teki_x1	teki_y1	teki_x2	teki_y2	teki_x3	teki_y3

それでは、敵の数が多くなったらどうでしょう。たとえば、敵の数が1000になったとしたら、teki_x1000とteki_y1000まで２０００個もの変数を定義しなければならなくなり、それはかなり面倒です。

ここで、配列の出番です。配列は、「個々の変数をひとかたまりにしたもの」です。配列を使うと、同じ型のデータをまとめて保持できるようになり、プログラムの煩雑さから解放されます。配列を図で表現すると、以下のように変数が連続しているイメージになります。上の図との違いは、変数が連続している点にあります。



配列とは、まとめて箱を確保し、位置(先頭からの距離)を指定してアクセスできる変数です。

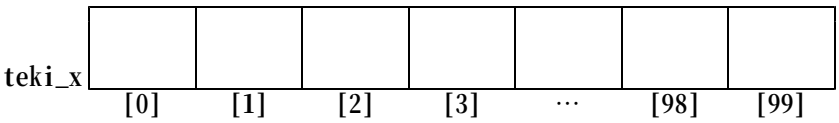
配列の定義

変数を使用するには定義が必要であったのと同じように、配列を使うためにも定義が必要です。

```
int[] teki_x = new int[100];     // int teki_x[] = new int[100]でも可
```

これは、teki_xという名前のint型の配列を定義しています。変数名の後に、new演算子と型名につづき、角括弧[]でくくって必要な個数を記述することで、配列の数(これを要素数といいます)を定義することができます。この例ではint型の１００個のデータが格納できる配列を定義しています。

配列を定義すると、メモリには以下の図のような変数の格納領域ができあがります。ちょうど、teki_x[0]，teki_x[1]，…，teki_x[99]という名前の１００個の変数が並んだ形になります。先頭がteki_x[1]ではなく、teki_x[0]になっていることに注意してください。



配列の添字

配列の個々の変数(これを配列の要素と呼びます)は、括弧[]を使ってアクセスすることができます。たとえば、上記の配列teki_xでは、teki_x[0]やteki_x[1]とすることで各要素にアクセスできます。この括弧でくくられた数字の部分を配列の添字と呼びます。Javaで配列の要素を扱う場合、次のことを注意してください。

配列の最初の要素は添字が０であり、最後の添字は「配列の要素数－１」である

配列の最初の要素には、添字を0にして(たとえば、teki_x[0])としてアクセスします。その次が1でアクセスします。0から始まることに注意してください。

１００個の要素数の場合、最後の１００番目の要素にアクセスするためには、添字を99とします。100にしてはいけません。0から始まるので、そこから100数えると最後は99になるわけです。また、添字には変数を使用することもとできます。たとえば、

```
int[]   array = new int[100];
int     i     = 9;
int     j     = array[i];    // array[9]と同じ意味
```

というように変数を使ってアクセスすることもできます。

プロジェクトの作成

Eclipseでは、プロジェクトという単位でプログラムを管理します。ｉアプリでは、ひとつのｉアプリに対してひとつのプロジェクトを作成します。
プロジェクトは、Javaソースファイルやリソースと呼ばれる各種画像や音楽ファイルなどをまとめて管理するものです。プロジェクトの実体は、プロジェクトフォルダ以下に作成される各種フォルダです。フォルダの構成は、以下のようになっています。

フォルダ	用途
bin	実行ファイル(.jar)が生成されるフォルダ
classes	コンパイルしたファイルが保存されるフォルダ
res	画像や音楽などのリソースを置くフォルダ
sp	ｉアプリでデータを保存するときに用いるスクラッチパッドの内容が保存されるフォルダ
src	ソースファイルを保存するフォルダ

課　題

Eclipseを起動し、ｉアプリ用のプロジェクトを作成しましょう。

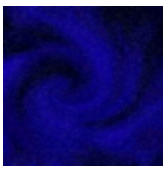
①デスクトップにあるEclipseのアイコンをダブルクリックして起動し、メニューから「ファイル」→「新規」→「プロジェクト」を選択します。

②「新規プロジェクト」の画面が表示されます。ｉアプリを作成する場合は、左側のツリーで「Java」を選んで、右側のプロジェクト一覧からは「DoJa-4.0プロジェクト」を選びます。そして「次へ」をクリックします。

③プロジェクト名を入力する画面が表示されるので、プロジェクトの名前「ShootingGame」を入力して「終了」をクリックします。ここで入力した名前のフォルダが作成されます。

画像の準備

今回作成するシューティングゲームでは、以下のような画像を使用します。

目　的	背　景	プレイヤー	敵	プレイヤー弾	敵の弾	おしまい
画　像						
ﾌｧｲﾙ名	BG1.jpg	PLY00.gif～PLY03.gif	ENM00.gif ENM01.gif	SHT00.gif～SHT03.gif	ENMSHT.gif	OSHI.gif
サイズ	240 × 240	42 × 55	113 × 168	24 × 22	18 × 32	157 × 37
説　明	背景となる画像です。	プレイヤーが操作するキャラクタの画像です。 アニメーション用に、４コマ分用意してあります。	敵の画像がENM00.gif、ダメージを与えたときの画像がENM01.gifです。	プレイヤーが発射する魔法弾です。 アニメーション用に、４コマ分用意してあります。	敵が発射する弾です。	ゲームが終了したときに表示する画像です。

課　題

シューティングゲームで使用する画像を作成し、プロジェクトのリソースを置くためのフォルダにコピーしましょう。

クラス

Javaでは、「クラス」という部品を設計し、クラスを組み合わせてプログラムを構築します。ｉアプリ対応ゲームの場合は、NTTドコモが開発したｉアプリケーションのIApplicationクラスと、画面に描画するためのDialogクラス、Panelクラス、Canvasクラスのうち最低１つをもとにクラスを作成しなければなりません。一般的なゲームでは、ｉアプリケーションクラスとキャンバスクラスを作成します。

クラスとは、オブジェクト指向プログラミングにおいて、データ(変数と定数)とその操作手順であるメソッド(関数)をまとめたものをいいます。オブジェクト指向プログラミングでは、クラスを定義することが基盤となっています。また、あるクラスを他のクラスに受け継がせることを「継承」といい、すでにあるクラスを再利用することにより、コードを大幅に削減することができます。

IApplicationクラスとは、ｉアプリの雛形を定義したクラスです。ｉアプリを作成するには、IApplicationクラスが持っているさまざまな能力を継承した新しいクラスを作成します。IApplicationクラスは、そのままでは使用できないようになっているので、必ずこのクラスを継承した新しいクラスを作成しなければならないのです。
Canvasクラスとは、ｉアプリで文字や図形、画像といったグラフィックを描画するためのクラスです。描画以外にも、キーの状態を取得する機能も提供しているので、まさにゲーム制作のためのクラスといえます。Canvasクラスも、そのままでは使用できないようになっているので、継承した新しいクラスを作成しなければなりません。このクラスに、ゲームの処理を記述します。

この２つのクラス以外にも、オブジェクト指向を用いたシューティングゲームでは、プレイヤークラスやエネミークラス、ショットクラスなどを作成します。しかし、ｉアプリではあえて作成せず、前述の２つのクラスだけにとどめます。なぜかというと、クラスを作成すればするほど、実行ファイルのサイズが大幅に増えていくので、容量の少ない現在の携帯電話では、実機にダウンロードすることすらできなくなってしまうからです。

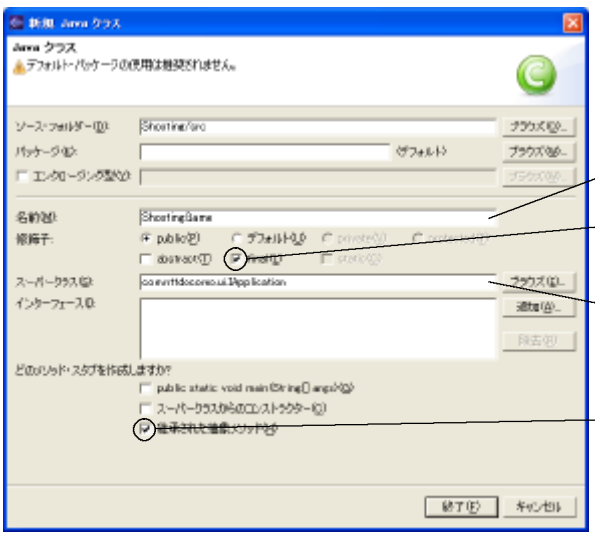
課　題

ｉアプリでゲームを作成するのに必要な２つのクラスを作成し、ｉアプリを実行してみましょう。

(１)ｉアプリケーションクラスを作成します。

①「パッケージ・エクスプローラー」の一番上のプロジェクト名「ShootingGame」を右クリックし、メニューから「新規(W)」→「クラス」を選びます。

②「新規 Javaクラス」ウィンドウが表示されるので、「名前(M)」と「スーパークラス(S)」を入力します。「名前」はクラス名です。「MyIAppli」とします。また、スーパークラス(もとになるクラス)には「com.nttdocomo.ui.IApplication」(NTTドコモのIApplicationクラス)と入力します。「final(L)」と「継承された抽象メソッド」にチェックを入れ、「終了」ボタンを押します。



クラス名「MyIAppli」を入力します

「final」にチェックを入れると、少しだけサイズを削減できます

継承するクラス名の正式名称「com.nttdocomo.ui.IApplication」を入力します。

「継承された抽象メソッド」にチェックを入れます。必要なメソッドが自動的に作成されます。

③MyIAppliクラスのソースファイルが作成されます。

(２)キャンバスクラスを作成します。

①「パッケージ・エクスプローラー」の一番上のプロジェクト名「JavaGame」を右クリックし、メニューから「新規(W)」→「クラス」を選びます。

②「新規 Javaクラス」ウィンドウが表示されるので、「名前(M)」と「スーパークラス(S)」を入力します。名前を「MyCanvas」、スーパークラスは「com.nttdocomo.ui.Canvas」と入力します。「final(L)」と「継承された抽象メソッド」にチェックを入れ、「終了」ボタンを押します。

③MyCanvasクラスのソースファイルが作成されます。

(３)ｉアプリの設定を行います。

①「パッケージ・エクスプローラー」で一番上のプロジェクト名をクリックし、メニューから「プロジェクト(P)」→「DoJa-4.0(S)」→「ADF/トラステッド動作設定(C)」を選択します。

②「プロジェクト設定」ウィンドウが表示されます。ここで入力しなければならないのは、一番上の「AppName」と中盤やや上の「AppClass」です。
「AppName」は、アプリケーションの名前「ShootingGame」を入力します。ここで指定した名前は、実際にｉアプリをダウンロードした際、ｉアプリ一覧に表示されます。
「AppClass」は、(１)で作成したアプリケーションのクラス名を入力します。今回は「ShootingGame」と入力します。

正しく入力できたら、「OK」ボタンを押します。

(４)ｉアプリを実行しましょう。

①メニューから「実行」→「実行」を選択します。

②「実行」ウィンドウが表示されるので、左側のツリーで「DoJa-4.0アプリケーション」を右クリックし、メニューから「新規」を選びます。

③新しい実行設定が作成されます。「名前」はそのままでも構いません。「実行」ボタンをクリックすると、実行が始まります。

④保存してないファイルがある場合は「リソースの保管」ウィンドウが表示されます。保存するファイルにチェックを付けて「OK」ボタンを押します。

⑤エミュレータのウィンドウが表示され、ｉアプリが実行されます。

⑥「PWR/HDL」ボタンを押すと終了します。

次回からは、アイコン▷をクリックするだけで実行できます。

メソッド

メソッドとは、クラスに備えられた機能のことをいいます。逆に言えば、クラスに機能を追加したいときはメソッドを定義する、ということです。
MyIAppliクラスには、アプリケーションを開始するためのstartメソッド、MyCanvasクラスには、画面の描画を行うpaintメソッドがすでに作成されています(ただし、処理は記述されていません)。言い換えれば、MyIAppliクラスには、アプリケーションを開始する機能、MyCanvasクラスには、画面の描画行う機能があると言えます。

ｉアプリでゲームを作成する場合は、キャンバスクラスに機能を追加していきます。つまり、MyCanvasクラスに必要なメソッドを追加していきます。
ゲームを初期化する機能、メインループ、プレイヤーや敵キャラクターの処理、キャラクターを描画する機能などは、メソッドとしてキャンバスクラスに記述します。

課　題

キャンバスクラスに、以下の機能を追加しましょう。

・実行……run メソッド
・初期化……initメソッド

・内部処理…procメソッド
・描画……drawメソッド

(１)MyCanvasクラスのソースコードを以下のように変更しましょう。

－MyCanvas.java－

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

/*
 * 作成日： 2005/06/20
 *
 * TODO この生成されたファイルのテンプレートを変更するには次を参照。
 * ウィンドウ ＞ 設定 ＞ Java ＞ コード・スタイル ＞ コード・テンプレート
 */

/**
 * @author waisemen
 *
 * TODO この生成された型コメントのテンプレートを変更するには次を参照。
 * ウィンドウ ＞ 設定 ＞ Java ＞ コード・スタイル ＞ コード・テンプレート
 */
public final class MyCanvas extends Canvas {

    // --- 定数の定義(ゲームに必要な定数を以下に定義します)

    // --- 変数の定義(ゲームに必要な変数を以下に定義します)

    // --- メソッドの定義(ゲームに必要なメソッドを以下に定義します)
    // --- 実行
    void run() {
    }

    // --- 初期化
    void init() {
    }

    // --- 内部処理
    void proc() {
    }

    // --- 描画
    void draw() {
    }

    /* (Javadoc なし)
     * @see com.nttdocomo.ui.Frame#paint(com.nttdocomo.ui.Graphics)
     */
    public void paint(Graphics arg0) {
        // TODO 自動生成したメソッド・スタブ

    }

}
```

Eclipseでは、プログラムの入力中に補完候補が現れることがあります。一覧から選択して「Enter」キーを押すと、自動的に選んだ項目が入力され便利です。このような補完機能を活用することで、単純な入力ミスを防ぐことが可能です。補完候補は「Ctrl」＋「Space」キーで手動表示させることもできます。
エラーが検出されたときには、画面左にエラーマーク表示されます。ここで、エラーマークをクリックすると、エラーの修正候補が表示されます。
このように、Eclipseは、非常に強力な入力支援機能を持っています。

各メソッドの役割は、以下のようになります。

・runメソッドは、ゲームを実行するメソッドです。このメソッドにゲームの中心となる処理を記述します。
・initメソッドは、ゲームを初期化するメソッドです。変数や画像など、ゲームを初期状態に設定するための処理を記述します。
・procメソッドは、ゲームの内部処理を行うメソッドです。キャラクターの座標や状態を変更するための処理を記述します。
・drawメソッドは、描画を行うメソッドです。内部処理で変更された状態を画面に表示するための処理を記述します。

(２)MyIAppliクラスのソースコードを以下のように変更しましょう。

－MyIAppli.java－

```
import com.nttdocomo.ui.*;
        :
   コメント省略
        :
public final class MyIAppli extends IApplication {

    /* (Javadoc なし)
     * @see com.nttdocomo.ui.IApplication#start()
     */
    public void start() {
        MyCanvas    canvas = new MyCanvas();      // キャンバスの生成
        Display.setCurrent(canvas);               // キャンバスをディスプレイに設定
        canvas.run();                             // キャンバスのrunを呼び出し、ゲームを実行する
        terminate();                              // 終了
    }

}
```

startメソッドは、ｉアプリが実行されたときに最初に実行されるメソッドです。startメソッドで行うのは、キャンバスの作成、ディスプレイへの設定、キャンバス実行です。

ｉアプリでは、詳細な処理はキャンバスクラスに記述するのが一般的なので、アプリケーションクラスには、プログラムの起動と終了のみを記述します(今回のプログラムには、終了はありません)。

(３)プログラムが入力できたら、「Ctrl」＋「s」キーを押して保存しましょう。この際、自動的にコンパイルされます。

(４)アイコン▷をクリックし、実行しましょう。

メインループ

ゲームプログラムは、「ものの状態の集まり」と考えることができます。キャラクターの座標やパラメータ、背景、画面エフェクトなどすべて、それぞれ「もの」の状態です。ゲームプログラムとは、このいろいろな「もの」の状態を時間に沿って更新し、それを画面に描画するプログラムと考えることができます。

つまり、ゲームプログラムの処理を非常に単純化すると、

- 「もの」の状態を更新(内部処理)
- 「もの」を画面に描画(描画処理)

という２つにまとめることができます。

この２つの処理の繰り返しがメインループになります。このループ１回ぶんを「１フレーム」と呼びます。１フレームごとに、キャラクタの位置や状態、背景やそのほかの表示状態などを更新し、その状態を反映して描画を行う、という処理を繰り返していきます。

フレームレート

ゲームプログラムでは、メインループ(正確には画面の更新)を一定の間隔で実行します。実行速度は環境によって異なるので、速い環境ほど高速に動作できます。

１秒間に何回画面を更新するのかをフレームレート(FPS:Frame Per Second)と呼びます。60FPSの場合、１秒間に60回画面を更新しているという意味になり、１フレームを約16.67ミリ秒の間隔で表示しています。

一般的な家庭用ゲームでは60FPSで、携帯電話では、505iで20FPS、900iで30FPS程度が限界となります。この数値は、表示する画像の大きさや数が増えるほど、または高性能な描画機能を使用すると目に見えて落ちていきます。

課　題

(1)キャンバスクラスにメインループを作成します。

メインループは、キャンバスクラスのrunメソッド(ゲームを実行するためのメソッド)に記述します。メインループに入る前に、initメソッドを呼び出して変数を初期化しておきます。メインループでは、内部処理と描画処理を交互に行い、ゲームを進めていきます。

「？」を埋め、runメソッドを以下のように変更しましょう。

```
    // --- 実行
    void run() {
        init();         // 初期化

        // メインループ
        while(true) {
            ????();     // 内部処理
            ????();     // 描画
        }
    }
```

(2)フレームレートを制御する機能を追加します。

フレームレートを保つには、メインループが一定の間隔で回るようにします。メインループの処理時間は、その都度変わるので、メインループ１回ごとに処理時間を計り、それが１フレームの間隔未満の場合は、時間に余裕があるので、その分だけプログラムを休止させます。
休止時間は、以下の式で計算するものとします。

休止時間 ＝ １フレームの間隔 － １フレームを処理するのにかかった時間
(※休止時間が負の場合は、処理落ちになります)

１フレームの間隔 ＝ １０００ ／ ＦＰＳ　(※１秒＝１０００ミリ秒)
１フレームを処理するのにかかった時間 ＝ 現フレーム終了時間 － 前回のフレーム終了時間
(※前回のフレーム終了時間＝現フレーム開始時間)

キャンバスクラスに、フレームレートを制御するための定数を定義します。定数FPSおよびINTERVALを「public final class MyCanvas extends Canvas {」の直下に追加しましょう。

```
public final class MyCanvas extends Canvas {

    // --- 定数の定義(ゲームに必要な定数を以下に定義します)
    public final static int   FPS      = 30;           // フレームレート
    public final static int   INTERVAL = 1000 / FPS;   // フレームの間隔(ミリ秒)

    // --- 変数の定義
```

定数FPSが１秒あたりのフレーム数、定数INTERVALが現フレームから次フレームまでの間隔です。

(３)フレームレートを制御するための変数を追加します。

(４)の方法でフレームレートを制御するには、「前回のフレーム終了時間」を変数に保存しておく必要があります。long型の変数「lastTime」を「// --- 変数の定義」の下に追加しましょう。

```
    // --- 定数の定義
    public final static int   FPS      = 30;           // フレームレート
    public final static int   INTERVAL = 1000 / FPS;   // フレームの間隔

    // --- 変数の定義
    long   lastTime = 0;                               // 前回のフレーム終了時間

    // --- 実行
    void run() {
```

(５)休止時間を求め、プログラムを一時休止させるsleepメソッドを作成します。以下のプログラムをキャンバスクラスに追加しましょう。

```
    // --- 描画
    void draw() {
    }

    // --- 休止
    void sleep() {
        long   wait = INTERVAL - (System.currentTimeMillis() - lastTime);   // ※1
        if(wait > 0) {
            try {
                Thread.sleep(wait);                                         // ※2
            } catch(Exception e) {
            }
        }
        lastTime = System.currentTimeMillis();                              // ※3
    }
```

※１は、(３)の方法で休止時間を求める部分です。休止時間がある場合、※２のThread.sleepメソッドでプログラムを休止させます。※３は、実質的にメインループの最後に行われる処理で、フレームの終了時間を保存する処理です。System.currenTimeMillisメソッドを呼び出し、現在の時間を取得しています。

(６)メインループの最後で休止するようにします。メインループを以下のように変更しましょう。

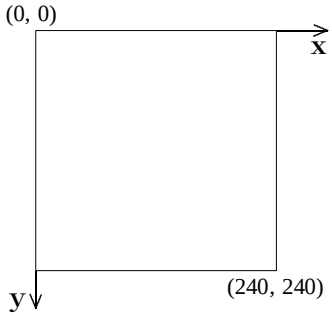
```
    // --- 実行
    void run() {
        init();         // 初期化

        // メインループ
        while(true) {
            ????();     // 内部処理
            ????();     // 描画
            sleep();    // 休止
        }
    }
```

Graphicsクラス

iアプリでグラフィックを描画する場合、Graphicsクラスを用います。このクラスは、文字を描画するdrawStringメソッド、線を引くdrawLineメソッド、矩形を描くdrawRectメソッド、画像を描画するdrawImageメソッドなど、描画に関するさまざまな機能を提供しています。
900iシリーズ以降は、ゲーム向けの機能が強化されました。2D-RPGのフィールドのような、さまざまな部品を敷き詰めて描画するdrawImageMapメソッド、画像の任意の部分を拡大縮小して描画するdrawScaledImageメソッドや、一昔前のゲーム機に搭載されていたスプライトを実現するためのSpriteクラスが導入されました。

スクリーン座標

2Dでグラフィックを描画する場合、座標の指定はスクリーン座標で行います。スクリーン座標とは、画面のピクセルに１対１に対応する座標系のことです。スクリーン座標の原点は、左上にあります。また、ｙ軸の向きが逆になっていることも大きな特徴です。



画像の読み込み

画像を読み込むには、以下のクラスライブラリを使用します。

・MediaManager
・MediaImage
・Image

画像は、MediaManagerをとおして取得します。MediaManager.getImageメソッドに画像が保存されている場所とファイル名を指定すると、形式のチェック、アクセス可能な場所かどうかのチェックを行います。誤りがなければ、画像を管理するためのMediaImage型の変数が返されます。
扱うことのできる形式は、GIF、透過GIF、JPEGです。指定できる場所は、リソース、URL、スクラッチパッド、byte配列ですが、大文字と小文字が区別されるので、注意が必要です。

MediaImageは、画像を管理します。MediaImageが指定された場所から画像を読み込み、各機種に合わせたデータに変換します。そのほか、画像の幅、高さ、イメージデータの取得、読み込まれた画像をメモリから解放する機能を提供します。

画像を描画するには、Image型の変数をGraphicsのdrawImageメソッドに渡します。Image型の変数は、MediaImageのgetImageメソッドで取得できます。

－画像読み込みのサンプルプログラム－

```
// 画像を読み込むのと描画するのに必要な変数の宣言
MediaImage   mi;
Image        img;

// 場所の指定(リソース)
mi = MediaManager.getImage("resource:///ファイル名");

// 画像の使用を宣言し、メモリに読み込む
try {
    mi.use();
} catch(Exception e) {
    // 例外処理(読み込めなかったときの処理をここに記述します)
}
```

```
// イメージの取得
img = mi.getImage();
```

useメソッドは、try-catch文で囲まないとエラーになります。try-catch文は、重大なエラーが発生したとき、どのようにするかを記述するためのものです。

画像を描画するには、GraphicsのdrawImageメソッドを呼び出します。このメソッドには、Image型の変数と描画座標(イメージの左上が基準)を渡します。

```
// 画像の描画(画面の(120, 80)に描画)
grp.drawImage(img, 120, 80);
```

画像の解放

携帯電話は容量がかなり少ないので、たくさんの画像をメモリに読み込むことはできません。不要になった画像を解放し、新しい画像を読み込むなどの工夫が必要になります。
画像の解放は、以下のような処理になります。

```
// 画像使用の終了を宣言(全機種共通)
try {
    img.dispose();
    img = null;

    mi.unuse();
    mi.dispose();
    mi = null;
} catch(Exception e) {
}

// ゴミ掃除(ガベージコレクション)
System.gc();
```

Image型の変数を取得したので、これをdisposeメソッドにより解放します。次に、MediaImageのunuseメソッドを呼び出してメモリから画像を解放します。最後に、disposeメソッドでMediaImageを解放します。最後のdisposeメソッドを呼ばない限り、メモリから解放されない機種もあります(プログラムを終了した場合は解放されます)。
Nシリーズなど一部の機種では、ImageをdisposeしただけでMediaImageもdisposeされてしまうという不具合があるので、上記のようにtry-catchで囲んでおきます。

課　題

(１)描画に必要なGraphicsクラスを取得します。Graphics型の変数「grp」を「// --- 変数の定義」の下に追加し、さらにグラフィックを取得するプログラムを追加しましょう。

```
// --- 定数の定義
public final static int   FPS      = 30;
public final static int   INTERVAL = 1000 / FPS;

// --- 変数の定義
Graphics  grp      = getGraphics();    // グラフィックの取得
long      lastTime = 0;                // 前回のフレーム終了時間

// --- 実行
void run() {
```

getGraphicsメソッドを呼び出すと、キャンバスに描画するためのグラフィック(正確にはグラフィックオブジェクト)が取得できます。このグラフィックに備えられているdrawStringメソッドやdrawImageメソッドを呼び出すことにより、キャンバスにグラフィックが描画されます。

(２)drawメソッドを次のように変更しましょう。

```
    // --- 描画
    void draw() {
        grp.lock();             // 描画開始

        // 以下に、描画処理を記述します

        grp.unlock(true);       // 描画終了
    }
```

iアプリでは、グラフィックのlockメソッドとunlockメソッドに囲まれた部分に描画処理を記述します。これ以外の部分では、正しく描画できなかったり、ちらつきが起こる場合があります。

(３)背景画像を保存するための変数を定義します。キャンバスクラスに以下の変数を追加しましょう。

```
    // 背景
    MediaImage   bgMi;
    Image        bgImg;
```

(４)背景画像を読み込みます。initメソッドを以下のように変更しましょう。

```
    //--- 初期化
    void init() {
        // 画像読み込み
        try {
            // 背景
            bgMi = MediaManager.getImage("resource:///BG.jpg");     // 読み込む画像の指定
            bgMi.use();                                             // 画像の読み込み
            bgImg = bgMi.getImage();                                // イメージを取得
        } catch(Exception e) {
        }
    }
```

「resource:///BG.jpg」が読み込む画像の指定です。大文字・小文字が区別されるので、正確に入力しましょう。「resource:///」は、場所がリソースという意味です。ここをhttp://ではじまるURLを入力することにより、インターネットから画像をダウンロードすることもできます。

(５)drawメソッドで背景を描画します。drawメソッドを以下のように変更しましょう。

```
    //--- 描画
    void draw() {
        grp.lock();                 // 描画開始

        grp.drawImage(bgImg, 0, 0); // 背景描画

        grp.unlock(true);           // 描画終了
    }
```

(６)プログラムを実行し、正しく描画されるかを確認しましょう。

(７)プレイヤーの画像を保存するための変数を定義します。キャンバスクラスに以下の変数を追加しましょう。

```
    // プレイヤー
    MediaImage   plyMi;
    Image        plyImg;
```

(８)キャラクター画像を読み込みます。initメソッドを以下のように変更しましょう。

```
    //--- 初期化
    void init() {
        // 画像読み込み
        try {
            // 背景
            bgMi = MediaManager.getImage("resource:///BG1jpg");
            bgMi.use();
            bgImg = bgMi.getImage();

            // プレイヤー
            plyMi = MediaManager.getImage("resource:///PLY00.gif");
            plyMi.use();
            plyImg = plyMi.getImage();
        } catch(Exception e) {
        }
    }
```

(７)drawメソッドでプレイヤーを描画します。drawメソッドを以下のように変更しましょう。

```
    //--- 描画
    void draw() {
        grp.lock();                     // 描画開始

        grp.drawImage(bgImg,  0, 0);    // 背景描画
        grp.drawImage(plyImg, 0, 0);    // プレイヤー描画

        grp.unlock(true);               // 描画終了
    }
```

(８)プログラムを実行し、正しく描画されるかを確認しましょう。

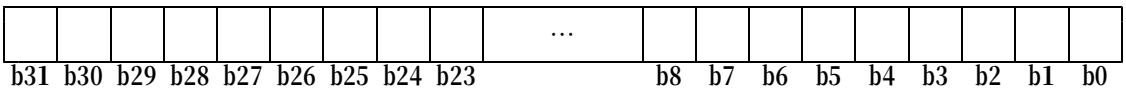
プレイヤーの移動

プレイヤーを移動しようとしてキーを押しても、まったく動きません。これは、移動のためのプログラムを記述していないためですが、もっとも問題なのは、つねに座標(０，０)に固定して描画しているという点です。このままでは、キーパッド入力処理のプログラムを記述したとしても、キャラクターは移動できません。

キャラクターは、固定された数値ではなく、キーパッドの入力によって数値を変更します。たとえば、右を１回押すたびに２つ移動する場合、(０，０)→(２，０)→(４，０)というように、ｘ座標の数値を増やしていかなければなりません。この動作を実現するために、座標に変数を導入します。

キーパッド状態の取得

携帯電話のボタン(キーパッド)の状態は、キャンバスクラスのgetKeypadStateメソッドで取得します。このメソッドを呼び出すと、int型の値を返します。Javaのint型は３２ビットあり、押されているキーに割り当てられたビットが１になり、押されていない場合は０になります。



たとえば、キー０に対応するビットは、ビット０(b0)で、キー１に対応するビットは、ビット１(b1)というように、それぞれのビットにキーが割り当てられています。

どのキーが押されているか(または押されていないか)を調べるには、「1 << (Display.キーの値)」を用います。たとえば、キー０は「Display.KEY_0」、キー１は「Display.KEY_1」となります。

キー	キーの値	値	備考
数字０	KEY_0	0x00	
数字１	KEY_1	0x01	
数字２	KEY_2	0x02	
数字３	KEY_3	0x03	
数字４	KEY_4	0x04	
数字５	KEY_5	0x05	
数字６	KEY_6	0x06	
数字７	KEY_7	0x07	
数字８	KEY_8	0x08	
数字９	KEY_9	0x09	
＊	KEY_ASTERISK	0x0a	
＃	KEY_POUND	0x0b	
◁	KEY_LEFT	0x10	
△	KEY_UP	0x11	
▷	KEY_RIGHT	0x12	
▽	KEY_DOWN	0x13	
選択/決定	KEY_SELECT	0x14	
ｸﾘｱ	KEY_CLEAR	0x20	getKeypadState(1)

▲よく使うキーとその値

課　題

キーパッドの入力状態によって、プレイヤーを移動させましょう。

(１)プレイヤーのｘ座標とｙ座標を変数で管理します。プレイヤーのｘ座標とｙ座標を格納する変数を以下のように宣言しましょう。

```
    // プレイヤー
    MediaImage    plyMi;
    Image         plyImg;
    int           plyX, plyY;
```

(２)initメソッドで変数を初期化し、プレイヤーの初期座標を設定します。initメソッドを以下のように変更しましょう。

```
    //--- 初期化
    void init() {
        // 変数初期化
        plyX = 0;
        plyY = 0;

        // 画像読み込み
            ⋮
```

ｘ座標(plyX)、ｙ座標(plyY)ともに０を設定しているので、プレイヤーは左上に描画されます。この値には、好きな数値を設定しましょう。

(３)drawメソッドを以下のように変更し、変数の座標にキャラクターが描画されるようにしましょう。

```
    //--- 描画
    void draw() {
        grp.lock();                             // 描画開始

        grp.drawImage(bgImg,  0, 0);            // 背景描画
        grp.drawImage(plyImg, plyX, plyY);  // プレイヤー描画

        grp.unlock(true);                       // 描画終了
    }
```

(４)この状態で実行すると、変数ply，plyYに設定した座標にプレイヤーが描画されます。

(５)プレイヤーを処理するメソッドを作成します。以下のプログラムを適切な場所に追加しましょう。

```
    //--- プレイヤー処理
    void player() {
    }
```

(６)内部処理procメソッドを以下のように変更し、playerメソッドが実行されるようにしましょう。

```
    //--- 内部処理
    void proc() {
        player();       // プレイヤー処理
    }
```

(７)キーパッドの状態を調べ、キャラクターを移動させます。移動は、変数plyX，plyYを増減させることで行います。playerメソッドを以下のように変更しましょう。なお、(ア)～(エ)は、各自考えてください。

```
    //--- プレイヤー処理
    void player() {
        // キー入力取得
        int   key = getKeypadState();

        // 移動
        if((1 << Display.KEY_LEFT  & key) != 0) // 左
            ※(ア)
        if((1 << Display.KEY_RIGHT & key) != 0) // 右
            ※(イ)
        if((1 << Display.KEY_UP    & key) != 0) // 上
            ※(ウ)
        if((1 << Display.KEY_DOWN  & key) != 0) // 下
            ※(エ)
    }
```

(８)実行して動作を確認しましょう。

(９)キャラクターの移動範囲を画面内だけにしましょう。

・ヒント１

座標は、画像の左上を指しています。

・ヒント２

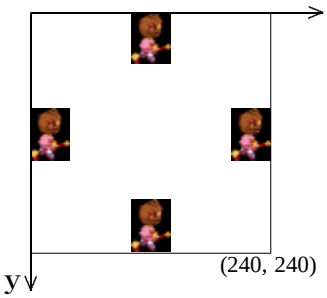
ｘ座標の最小値は、原点の0です。この値未満になるということは、原点より左側にいるということになり、画面外にいることになります。この場合は、ｘ座標を0にし、画面左に戻します。

・ヒント３

ｘ座標の最大値は、画面右端の240になりそうですが、実は違います。ｘ座標は、キャラクターの左端の座標なので、ここが240の場合は、すでに画面外に出ています。キャラクターの右端(左端 + 幅)が240を越えているかどうかを調べるか、キャラクターの左端が、240から幅を引いた値(240 - 幅)を越えているかを調べます。越えている場合は、ｘ座標を240から幅を引いた値にします。

・ヒント４

ｙ座標も同様に考えます。



敵の移動

敵は、プレイヤーと異なり、プログラムによって動かさなくてはなりません。動かし方はいろいろあり、一定のルートを通るだけのもの、一定のパターンで動くもの、プレイヤーや自分の状態によって異なる動作をするもの、時間によってパターンを変えるものなどさまざまな動作が考えられます。どのような動作をさせるかによって、ゲームの難易度や楽しさが変わってきます。

課　題

敵の画像を読み込んで描画し、さらにプログラムによって動かしましょう。

(１)敵の画像と座標を保存する変数を宣言します。以下のプログラムを「// プレイヤー」の下に追加しましょう。

```
    // プレイヤー
    MediaImage   plyMi;
    Image        plyImg;
    int          plyX, plyY;

    // 敵
    MediaImage   enmMi;
    Image        enmImg;
    int          enmX, enmY;

    //--- 初期化
    void init() {
        ⋮
```

(２)initメソッドで変数を初期化し、敵の初期座標を設定します。initメソッドを以下のように変更しましょう。

```
    //--- 初期化
    void init() {
        // 変数初期化
        plyX = 0;
        plyY = 0;

        enmX = 127;
        enmY = 0;

        // 画像読み込み
            ⋮
```

ｘ座標(enmX)に１２７、ｙ座標(enmY)に０を設定しているので、敵は右上に描画されます。この値には、好きな数値を設定しましょう。

(３)initメソッドで敵の画像を読み込みます。「// 画像読み込み」を以下のように変更しましょう。

```
        enmX = 127;
        enmY = 0;

        // 画像読み込み
        try {
            // 背景
            (省略)

            // プレイヤー
            (省略)

            // 敵
            enmMi = MediaManager.getImage("resource:///ENM00.gif");
            enmMi.use();
            enmImg = enmMi.getImage();
        } catch(Exception e) {
        }
    }
```

(４)drawメソッドで敵を描画します。drawメソッドを以下のように変更しましょう。

```
    //--- 描画
    void draw() {
        grp.lock();                         // 描画開始

        grp.drawImage(bgImg,  0, 0);        // 背景描画
        grp.drawImage(enmImg, enmX, enmY);  // 敵描画
        grp.drawImage(plyImg, plyX, plyY);  // プレイヤー描画

        grp.unlock(true);                   // 描画終了
    }
```

(５)プログラムを実行し、敵が描画されるかを確認しましょう。

(６)敵を動かす処理を作成します。

敵は、「動きデータ」を与えることによって動かすことにします。敵の行動は、８方向への移動と一時停止の９パターンとします。それぞれの行動に数字を割り当て、これを並べて動きデータとします。
動きデータは、以下の０～９の数字を並べることによって作成します。１フレームあたり１つの動きデータを処理します。

7 …↖(左上)	8 …↑(上)	9 …↗(右上)
4 …←(左)	5 …一時停止	6 …→(右)
1 …↙(左下)	2 …↓(下)	3 …↘(右下)

0 …動きデータ先頭へ戻る

たとえば、下に２０回、上に２０回移動するというパターンを繰り返すようにするには、

動きデータ = 22222222222222222222888888888888888888880

というデータになります。

まず、動きデータを保存する変数と、何文字目の動きデータを処理するかを保存する変数を宣言します。以下のプログラムを変数の宣言「// 敵」の下に追加しましょう。

```
    // 敵
    MediaImage   enmMi;
    Image        enmImg;
    int          enmX, enmY;

    // 敵動きデータ
    String       enmMove = "";
    int        enmMvCnt;        // 動きデータカウンタ
```

(7)initメソッドで変数を初期化します。initメソッドを以下のように変更しましょう。

```
        plyY    = 0;

        enmX    = 127;
        enmY    = 0;
        enmMvCnt = 0;
```

変数enmMoveは、あとで初期化します。

(8)敵を処理するenemyメソッドを作成します。以下のプログラムを適切な場所に追加しましょう。なお、プログラム中の'?'には、'+'または'-'のいずれか適切なものを入れましょう。

```
    //--- 敵処理
    void enemy() {
        // 敵移動
        if(enmMove.length() > enmMvCnt) {
            if(enmMove.charAt(enmMvCnt) == '0')
                enmMvCnt = 0;                               // 先頭に戻る
            switch(enmMove.charAt(enmMvCnt)) {
              case '1': enmX ?= 4;  enmY ?= 4;  break;  // 左下
              case '2':             enmY ?= 4;  break;  // 下
              case '3': enmX ?= 4;  enmY ?= 4;  break;  // 右下
              case '4': enmX ?= 4;              break;  // 左
              case '5':                         break;  // 一時停止
              case '6': enmX ?= 4;              break;  // 右
              case '7': enmX ?= 4;  enmY ?= 4;  break;  // 左上
              case '8':             enmY ?= 4;  break;  // 上
              case '9': enmX ?= 4;  enmY ?= 4;  break;  // 右上
            }
            enmMvCnt++;                                     // 次のデータへ
        }
    }
```

enemyメソッドでは、enmMove.charAtメソッドで動きデータを１文字取りだし、switch文によって適切な処理に分岐させて敵の座標を増減させています。

(9)procメソッドを以下のように変更し、内部処理としてenemyメソッドが実行されるようにしましょう。

```
    //--- 内部処理
    void proc() {
        player();       // プレイヤー処理
        enemy();        // 敵処理
    }
```

(10)動きデータを定義します。変数enmMoveの宣言を以下のように変更しましょう。

```
    // 敵動きデータ
    String      enmMove = "22222222222222222222888888888888888888880";
    int         enmMvCnt;       // 動きデータカウンタ
```

―敵動きデータサンプル―

```
String    enmMove = "22222222222222244444444445555566666666668888888888888880";
String    enmMove = "11111111111111111111111111111177777777777777777777777777777770";
```

(11)プログラムを実行すると、動きデータにそって敵が動きます。

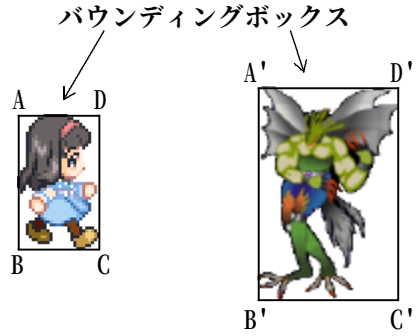
衝突検出

ゲームの中で、キャラクター同士の衝突検出(Collision Detection:衝突判定、当たり判定、ヒットチェックなどとも呼ばれます)は、ひんぱんに必要になります。シューティングゲームでは、プレイヤーと敵、プレイヤーの弾と敵、プレイヤーと敵の弾など、さまざまな種類の衝突検出が行われます。さらに、それらが１秒間に３０回の移動が行われるとすると、そのぶんだけ衝突検出が必要となります。

バウンディングボックス

ゲームのようにリアルタイム性を追求する場合、キャラクターの複雑な形状を完全に考慮した衝突検出をするのはたいへんです。実際には、バウンディングボックス(Bounding Box:衝突範囲)と呼ばれる架空の領域を用いて、その領域の交差判定を行います。

２次元の場合には、矩形や円を用いることが多く、この領域ともう一方のキャラクターの交差判定を行うことで、擬似的に衝突を検出します。たいていの場合、キャラクターをすっぽりと包むような矩形にします。そうでない場合には、キャラクターのめり込みが起こる可能性が出てきます。



バウンディングボックス

上の図のバウンディングボックスの頂点Ａの座標を(Ax, Ay)、頂点Ｃを(Cx, Cy)、頂点Ａ'を(A'x, A'y)、頂点Ｃ'を(C'x, C'y)と表します。このとき、２つのバウンディングボックスが交差しているかどうかを調べるプログラムは、次のようになります。

```
if(Ax <= C'x && Cx >= A'x && Ay <= C'y && Cy >= A'y)
    交差している
else
    交差していない
```

スクリーン座標系のｙ軸は、下方向に向いていることに注意しましょう。条件式をまとめると、

・頂点Ａが頂点Ｃ'の左側にあり、かつ、頂点Ｃが頂点Ａ'の右側にある
・頂点Ａが頂点Ｃ'の上側にあり、かつ、頂点Ｃが頂点Ａ'の下側にある

となります。この２つの条件を満たしたとき、２つの領域は交差しています。この条件を言い換えれば、四角形ＡＢＣＤの一部または全部が、四角形Ａ'Ｂ'Ｃ'Ｄ'の中にあるかどうかということです。

逆に言えば、

・頂点Ａが頂点Ｃ'の右側にある
・頂点Ｃが頂点Ａ'の左側にある
・頂点Ａが頂点Ｃ'の下側にある
・頂点Ｃが頂点Ａ'の上側にある

の条件を１つでも満たすと、絶対に交差していないということになります。これをJavaおよびＣ言語で記述すると、次のようなプログラムになります。

```
// 衝突判定([ax1, ay1]が頂点A、[ax2, ay2]が頂点C、[bx1, by1]が頂点A'、[bx2, by2]が頂点C')
if(ax2 < bx1 || ax1 > bx2 || ay1 > by2 || ay2 < by1)
    // 衝突していない
else
    // 衝突している
```

課　題

衝突を判定するメソッドを作成しましょう。

(１)矩形の交差を判定するisIntersectメソッドを作成します。以下のプログラムを適切な場所に追加しましょう。

```
    //--- 交差判定
    boolean isIntersect(int ax1, int ay1, int ax2, int ay2, int bx1, int by1, int bx2, int by2) {
        if(ax2 < bx1 || ax1 > bx2 || ay1 > by2 || ay2 < by1)
            return false;    // 衝突していない
        return true;         // 衝突している
    }
```

(２)キャラクターの衝突を検出し、衝突した場合のプログラムを記述するcollisionDetectionメソッドを作成します。以下のプログラムを適切な場所に追加しましょう。

```
    //--- 衝突検出
    void collisionDetection() {
    }
```

(３)内部処理で衝突検出を行うようにします。procメソッドを以下のように変更しましょう。

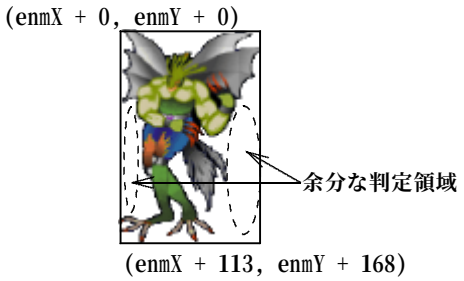
```
    //--- 内部処理
    void proc() {
        player();                // プレイヤー処理
        enemy();                 // 敵処理
        collisionDetection();    // 衝突検出
    }
```

この変更により、「プレイヤーの処理(playerメソッド)→敵の処理(enemyメソッド)→衝突検出(collisionDetectionメソッド)」という流れになり、キャラクターの移動後、衝突の検出ができるようになります。

(４)プレイヤーと敵の衝突を検出し、衝突した場合に「あたり」とコンソールに出力されるようにしてみます。collisionDetectionメソッドを以下のように変更しましょう。

```
    //--- 衝突検出
    void collisionDetection() {
        // プレイヤーと敵の衝突判定
        if(isIntersect(plyX + 0, plyY + 0, plyX + 42,  plyY + 55,
                       enmX + 0, enmY + 0, enmX + 113, enmY + 168)) {
            System.out.println("あたり");
        }
    }
```

プレイヤーと敵にバウンディングボックスを設定し、それが交差しているかどうかをisIntersectメソッドで調べます。バウンディングボックスは、以下のようにキャラクターを完全に覆うようになっているので、画像が存在しない場所も衝突しているとみなされます。



左上の頂点(enmX + 0, enmY + 0)の数値を増やすと、領域が右下に狭まります。同様に、右下の頂点(enmX + 113, enmY + 168)の数値を減らすと、頂点が左上に狭まります。
たとえば、左上の頂点を(enmX + 37, enmY + 34)、右下の頂点を(enmX + 50, enmY + 50)とすると、顔の部分だけが衝突領域になります。

課　題

プレイヤーと敵にライフを設定し、画面上部に表示しましょう。また、スコアも追加して表示しましょう。

(１)ライフは、変数で管理します。プレイヤーと敵のライフを格納する変数を宣言します。変数plyHPとenmHPを以下のように宣言しましょう。

```
    // プレイヤー
    MediaImage      plyMi;
    Image           plyImg;
    int             plyX, plyY;
    int             plyAnm;
    int             plyHP;                  // HP

    // 敵
    MediaImage      enmMi;
    Image           enmImg;
    int             enmX, enmY;
    int             enmHP;                  // HP
```

(２)initメソッドでライフの初期値を設定します。initメソッドを以下のように変更しましょう。

```
    //--- 初期化
    void init() {
        // 変数初期化
        plyX     = 0;
        plyY     = 0;
        plyAnm   = 0;
        plyHP    = 5;

        enmX     = 127;
        enmY     = 0;
        enmHP    = 10;
        enmMvCnt = 0;

        // 画像読み込み
```

(３)drawメソッドでライフを描画するようにします。drawメソッドを以下のように変更しましょう。

```
    //--- 描画
    void draw() {
        grp.lock();                                 // 描画開始

        grp.drawImage(bgImg, 0, 0);                 // 背景描画
        grp.drawImage(enmImg, enmX, enmY);          // 敵描画
        grp.drawImage(plyImg[plyAnm], plyX, plyY);  // プレイヤー描画

        // HP描画
        grp.setColor(Graphics.getColorOfRGB(255, 255, 255));
        grp.drawString("PLAYER " + plyHP,   0, 10);
        grp.drawString("ENEMY "  + enmHP, 180, 10);

        grp.unlock(true);                           // 描画終了
    }
```

drawStringメソッドは、文字を描画するメソッドです。描画の前にsetColorメソッドで色を設定しています。その中のGraphics.getColorOfRGBメソッドが色を作成するメソッドです。左から光の三原色の赤、緑、青の明るさで、これらが混合されて色が作成されます。色は、０～２５５で指定します。大きいほど明るくなります(黒からそれぞれの成分に近づきます)。

(４)スコアを格納する変数を宣言します。以下のプログラムを適切な場所に追加しましょう。

```
    int    score;
```

(５)initメソッドでスコアを初期化します。initメソッドの適切な場所に、以下のプログラムを追加しましょう。

```
score = 0;
```

(６)drawメソッドでスコアを描画するようにします。drawメソッドの適切な場所に、以下のプログラムを追加しましょう。

```
grp.drawString("SCORE "  + score,  90, 10);
```

課　題

プレイヤーまたは敵のライフが０以下になったらゲームが終了するようにしましょう。また、そのときに「おしまい」の画像を表示するようにしましょう。

(１)以下の方法(アルゴリズム)で作成します。

１．ゲームが進行している状態と、ゲームが終了した状態(ゲームオーバー)の２つの状態が存在することになります。２つの状態によって、処理を分岐させます。
２．ゲームの状態は、変数で管理します。
３．２の変数が１なら通常状態、２ならゲームオーバーとします。
４．ゲームオーバーなら「おしまい」と表示します。そのとき、ゲームは進行させません。

(２)ゲームオーバーかどうかを示す変数を宣言します。変数game0verを以下のように宣言しましょう。

```
    // 敵動きデータ
    String          enmMove = "";
    int             enmMvCnt;          // 動きデータカウンタ

    int             score;

    // ゲームモード
    int             gameMode;
```

変数gameModeには、通常状態なら「１」、ゲームオーバーなら「２」を設定します。ゲームを拡張する場合、０ならタイトル、３ならエンディングというようにしていきます。

(３)initメソッドで変数gameModeを初期化します。ゲームを起動したときは、通常状態から始まるようにします。initメソッドを以下のように変更しましょう。なお、「？」は各自考えてください。

```
    //--- 初期化
    void init() {
        // 変数初期化
        gameMode = ?;

        plyX      = 0;
        plyY      = 0;
```

(４)「おしまい」画像を読み込みます。画像を格納する変数osiImgを以下のように宣言しましょう。

```
    // ゲームオーバー
    int          gameMode;
    MediaImage   osiMi;
    Image        osiImg;
```

(５)画像を読み込みます。initメソッドの適切な場所に、以下のプログラムを追加しましょう。

```
    // おしまい
    osiMi = MediaManager.getImage("resource:///0SHI.gif");
    osiMi.use();
    osiImg = osiMi.getImage();
```

(６)内部処理をゲームの状態によって分岐させます。ゲームオーバーの場合、「おしまい」と表示するだけなので、内部処理を行う必要はありません。ゲームオーバーでない場合は、これまでどおりプレイヤー、敵、衝突検出を行います。
内部処理procメソッドを以下のように変更しましょう。なお、「？」は各自考えてください。

```
    //--- 内部処理
    void proc() {
        if(gameMode == ?)
            return;
        player();               // プレイヤー処理
        enemy();                // 敵処理
        collisionDetection();   // 衝突検出
    }
```

この変更により、ゲームオーバーの場合は内部処理が実行されなくなります。

(７)ゲームオーバーの場合、「おしまい」と表示するようにします。drawメソッドを以下のように変更しましょう。なお、「？」は各自考えてください。

```
    //--- 描画
    void draw() {
        grp.lock();                                 // 描画開始

        grp.drawImage(bgImg,  0, 0);                // 背景描画
        grp.drawImage(enmImg, enmX, enmY);          // 敵描画
        grp.drawImage(plyImg[plyAnm], plyX, plyY);  // プレイヤー描画

        // HP、スコア描画
        grp.setColor(Graphics.getColorOfRGB(255, 255, 255));
        grp.drawString("PLAYER " + plyHP,   0, 10);
        grp.drawString("ENEMY "  + enmHP, 180, 10);
        grp.drawString("SCORE "  + score,  90, 10);

        // おしまい描画
        if(gameMode == ?)
            grp.drawImage(osiImg, 42, 102);

        grp.unlock(true);                           // 描画終了
    }
```

(８)プレイヤーまたは敵のライフが０以下になったら、状態をゲームオーバーに変更します。この処理は、内部処理の最後に行います。procメソッドを以下のように変更しましょう。

```
    //--- 内部処理
    void proc() {
        if(gameMode == ?)       // この?は、(６)の問題になっています
            return;

        player();               // プレイヤー処理
        enemy();                // 敵処理
        collisionDetection();   // 衝突検出

        // ゲームオーバー判定
        if(plyHP <= 0 || enmHP <= 0)
            gameMode = 2;
    }
```

追加した判定により、どちらかのライフが０以下になると変数gameModeが２になり、ゲームオーバー状態となります。この状態になると、(６)により内部処理が実行されなくなり(＝ゲームが進行しない)、(７)により「おしまい」と表示されるようになります。

(９)プレイヤーと敵が衝突したら、プレイヤーのライフを０にします。以下のプログラムを適切な場所に追加しましょう。なお、「System.out.println("あたり")」は削除してかまいません。

```
plyHP = 0;
```

(10)プレイヤーと敵が衝突した場合、プレイヤーのライフが０になってゲームオーバーになるかどうか確認しましょう。

課　題

プレイヤーが弾を発射できるようにしましょう。

(１)以下の方法(アルゴリズム)で作成します。

・携帯電話の真ん中のボタン(SELECTキー)が押されたときに弾を発射
・発射できるのは１発のみ
・言い換えれば、画面に弾がないとき発射可能
・発射された弾は右に移動する
・画面の端または敵に当たると消滅(＝未発射状態となる)
・敵に当てるとダメージ１(敵のライフが１減る)、スコア1,000点加算
・弾は、「発射されている(画面内に存在する)状態」と「発射されていない(画面内に存在しない)状態」という２つの状態が存在する
・弾の状態は、変数で管理

(２)弾を管理する変数を宣言します。必要な変数は以下のものです。

・画像(pshMi:MediaImage型、pshImg:Image型)
・発射されているかどうか(pshLNC:boolean型…trueは発射中、falseは未発射)
・ｘ座標(pshX:int型)
・ｙ座標(pshY:int型)

以上の変数を宣言する以下のプログラムを適切な場所に追加に追加しましょう。

```
// プレイヤーショット
MediaImage      pshMi;
Image           pshImg;
boolean         pshLNC;
int             pshX;
int             pshY;
```

(３)変数を初期化します。initメソッドの適切な場所に、以下のプログラムを追加しましょう。

```
pshLNC = false;
```

弾の初期化は、発射状態をfalseにして未発射状態にしておくだけで十分です。座標は発射したときに、プレイヤーの座標をもとに決定します。

(４)画像を読み込みます。initメソッドの適切な場所に、以下のプログラムを追加しましょう。

```
// プレイヤーショット
pshMi = MediaManager.getImage("resource:///SHT00.gif");
pshMi.use();
pshImg = pshMi.getImage();
```

(５)弾が未発射状態でSELECTキーが押されたら、弾を発射するようにします。この処理は、プレイヤーに関する処理なので、playerメソッドに記述します。
以下のプログラムをplayerメソッドのに追加しましょう。なお、「？」は各自考えてください。

```
// ショット発射
if(pshLNC == false) {                           // 発射されていないか調べる
    if((1 << Display.KEY_?????? & key) != 0) {  // SELECTキーが押されているか調べる
        pshLNC = true;                          // 発射状態にする
        pshX   = plyX + 36;                     // 座標の設定(杖の先)
        pshY   = plyY + 22;
    }
}
```

弾が発射されていない状態(変数pshLNCがfalse)のとき、SELECTキーが押されたら、弾を発射状態(変数pshLNCをtrue)にし、弾の座標をプレイヤーの座標plyX，plyYをもとに変数pshX，pshYに設定します。

(６)弾を描画するようにします。弾を描画するのは、発射されているときだけです。発射されていないときに描画してはいけません。

以下のプログラムをdrawメソッドに追加しましょう。

```
        grp.drawImage(plyImg[plyAnm], plyX, plyY);  // プレイヤー描画

        // プレイヤーショット描画
        if(pshLNC == true)
            grp.drawImage(pshImg, pshX, pshY);

        // HP、スコア描画
        grp.setColor(Graphics.getColorOfRGB(255, 255, 255));
```

弾を描画するのは、発射されているとき(変数pshLNCがtrueのとき)だけなので、描画の前に判定を行い、条件を満たすときだけ描画します。

(７)とりあえず、弾を発射できるようになったので、プログラムを実行して確認してみましょう。

(８)弾を移動させる処理を作成します。必要な処理は、以下のものです。

・弾を移動させる
・弾が完全に画面外に出たら弾を未発射状態にする
・ただし、これらの処理を行うのは、弾が発射されているときだけ

この処理は、プレイヤーの弾を処理する専用のメソッドplayerShotを作成し、そこに記述します。以下のプログラムの「？」を埋め、適切な場所に追加しましょう。

```
    //--- プレイヤーショット処理
    void playerShot() {
        if(pshLNC == ????) {     // 弾が発射されているか調べる
            // 弾の移動
            pshX += 4;

            // 弾が画面内にあるか調べる
            // (弾の領域と画面の領域が交差しているか調べる＝交差していなければ、画面外にある)
            if(isIntersect(pshX, pshY, pshX + 24, pshY + 22,
                           0,    0,    240,        240) == false)
                pshLNC = ?????;      // 未発射状態にする
        }
    }
```

(９)内部処理procメソッドで、弾の処理が行われるようにします。procメソッドを以下のように変更しましょう。

```
    //--- 内部処理
    void proc() {
        if(gameMode == 2)
            return;

        player();                // プレイヤー処理
        enemy();                 // 敵処理
        playerShot();            // プレイヤーショット処理
        collisionDetection();    // 衝突検出

        // ゲームオーバー判定
        if(plyHP <= 0 || enmHP <= 0)
            gameMode = 2;
    }
```

(10)以上で弾が移動するようになったので、プログラムを実行して確認してみましょう。

(11)弾と敵の衝突検出を行います。衝突した場合、敵のライフを１減らし、スコアに1,000を加算します。以下のプログラムの「？」を埋め、collisionDetectionメソッドの適切な場所に追加しましょう。

```
    //--- 衝突検出
    void collisionDetection() {
        // プレイヤーショットと敵の衝突検出
        if(pshLNC == ????) {        // 発射されているときだけ判定する
            if(isIntersect(pshX,     pshY,     pshX + 24, pshY + 22,
                           enmX + 37, enmY + 34, enmX + 50, enmY + 50)) {
                pshLNC  = ?????;    // 弾を消す
                enmHP  -= 1;        // 敵のHPを減らす
                score  += 1000;     // スコアに1000を加算
            }
        }

        // プレイヤーと敵の衝突判定
```

プレイヤーの弾と敵の衝突検出も、弾が発射されているときだけ行います。弾は、敵の領域を調整し、顔の部分だけにしか当たらないようにしています。
衝突した場合、弾を消し(状態を未発射にする)、敵のライフから１を引き、スコアに1000を加算します。

課　題

敵が弾を発射するようにしましょう。

(１)以下の方法(アルゴリズム)で作成します。

・発射できるのは１発のみ
・言い換えれば、画面に弾がないとき発射可能
・発射する確率は、条件を満たしたときに１０％の確率
・発射された弾は左に移動する
・画面の端またはプレイヤーに当たると消滅(＝未発射状態となる)
・プレイヤーが当たるとダメージ１(プレイヤーのライフが１減る)
・プレイヤーの弾と敵の弾が当たった場合は、両方とも消滅し、スコアに100を加算
・弾は、「発射されている(画面内に存在する)状態」と「発射されていない(画面内に存在しない)状態」という２つの状態が存在する
・弾の状態は、変数で管理

(２)以下のプログラムを適切な場所に追加しましょう。なお、「？」は各自考えてください。

・乱数を扱うための変数の宣言

```
Random   rand = new Random();
```

※「import java.util.Random;」が必要

・敵の弾を管理する変数の宣言

```
    // 敵ショット
    MediaImage      eshMi;
    Image           eshImg;     // 画像
    boolean         eshLNC;     // 発射状態
    int             eshX;       // ｘ座標
    int             eshY;       // ｙ座標
```

・敵の弾を初期化するプログラム

```
eshLNC = false;
```

・敵の弾の画像を読み込むプログラム

```
    // 敵ショット
    eshMi = MediaManager.getImage("resource:///ENMSHT.gif");
    eshMi.use();
    eshImg = eshMi.getImage();
```

・敵が弾を発射するプログラム

```
    // ショット発射
    if(eshLNC == ?????) {                                       // 発射されていない
        if(Math.abs(rand.nextInt() % (100 / 10)) == 0) {        // １０%の確率
            eshLNC = true;                                      // 発射状態にする
            eshX   = enmX + 24;                                 // 座標の設定(口のあたり)
            eshY   = enmY + 32;
        }
    }
```

敵の弾が未発射のとき、０～９のランダムな数値を作成し、それが０のとき(＝１／１０の確率)に弾を発射します。なお、この処理は敵に関する処理なので、enemyメソッドに記述します。

・敵の弾を描画するプログラム

```
    // 敵ショット描画
    if(eshLNC == ????)
        grp.drawImage(eshImg, eshX, eshY);
```

プレイヤーの弾と同様に、発射されているときだけ描画します。

・敵の弾を移動するプログラム

```
    //--- 敵ショット処理
    void enemyShot() {
        if(eshLNC == ????) {
            // 弾の移動
            eshX -= 6;

            // 弾が画面内にあるか調べる
            // (弾の領域と画面の領域が交差しているか調べる＝交差していなければ、画面外にある)
            if(isIntersect(eshX, eshY, eshX + 18, eshY + 32, 0, 0, 240, 240) == false)
                eshLNC = ?????;     // 未発射状態にする
        }
    }
```

・内部処理procメソッドへ追加

```
    enemyShot();        // 敵ショット処理
```

・プレイヤーとの衝突を判定するプログラム

```
    // プレイヤーと敵の弾の衝突判定
    if(eshLNC == ????) {        // 発射されているときだけ判定する
        if(isIntersect(plyX + 0, plyY + 0, plyX + 42, plyY + 55,
                       eshX + 0, eshY + 0, eshX + 18, eshY + 32)) {
            eshLNC  = ?????;    // 弾を消す
            plyHP  -= 1;        // プレイヤーのHPを減らす
        }
    }
```

・プレイヤーの弾との衝突を判定するプログラム

```
    // プレイヤーの弾と敵の弾の衝突検出
    if(pshLNC == ???? && eshLNC == ????) {      // 両方が発射されているときだけ判定する
        if(isIntersect(pshX, pshY, pshX + 24, pshY + 22,
                       eshX, eshY, eshX + 18, eshY + 32)) {
            pshLNC  = ?????;                    // 弾を消す
            eshLNC  = ?????;
            score  += 100;                      // スコアに100を加算
        }
    }
```

課　題

プレイヤーをアニメーションさせましょう。

(1)プレイヤーの「画像を保存する変数」を配列にします。変数plyMiとplyImgの宣言を以下のように変更しましょう。

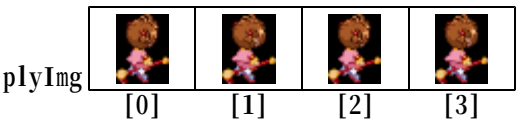
```
    // プレイヤー
    MediaImage[]    plyMi  = new MediaImage[4];
    Image[]         plyImg = new Image[4];
    int             plyX, plyY;
```

４コマ分の画像があるので、要素数４の配列を作成します。

(2)配列に画像を読み込みます。initメソッドの「// 画像読み込み」のプレイヤーの部分を以下のように変更しましょう。

```
        // プレイヤー
        for(int i = 0; i < plyMi.length; i++) {
            plyMi [i] = MediaManager.getImage("resource:///PLY0" + i + ".gif");
            plyMi [i].use();
            plyImg[i] = plyMi[i].getImage();
        }
```

for文で配列plyMiの要素の数(plyMi.length)だけ繰り返して画像を読み込みます。要素数は４なので、ループカウンタｉが０～３の間for文の内部が実行されます。プレイヤーのファイル名はPLY00.gif～PLY03.gifで、ちょうど"PLY0"とｉ(０～３)と".gif"を足した文字列と同等になります。これを利用して効率よくファイルを読み込んでいます。



(3)「何コマ目の画像を描画するのか」を保存する変数plyAnmを宣言します。「// プレイヤー」を以下のように変更しましょう。

```
    // プレイヤー
    MediaImage[]    plyMi  = new MediaImage[4];
    Image[]         plyImg = new Image[4];
    int             plyX, plyY;
    int             plyAnm;
```

(4)initメソッドで変数plyAnmを初期化します。initメソッドを以下のように変更しましょう。

```
        // 変数初期化
        plyX    = 0;
        plyY    = 0;
        plyAnm  = 0;
            :
```

(5)描画するコマを計算します。描画するコマは４コマなので、変数plyAnmを０から３まで増やし、３を超えたら０に戻します。playerメソッドを以下のように変更しましょう。

```
    //--- プレイヤー処理
    void player() {
        // キー入力取得
        int   key = getKeypadState();

        // 移動
        (省略)

        // 移動限界処理
        (省略)

        // アニメーション処理
        plyAnm = (plyAnm + 1) % plyImg.length;
    }
```

変数plyAnmを１ずつ増やし、要素数plyImg.length(今回は４)で割った余りを求めることにより、０から３の値しかとらなくなります(plyAnmが４になると０になります)。

(６)プレイヤーの描画をアニメーション対応にします。以下のように変更しましょう。

```
grp.drawImage(plyImg[plyAnm], plyX, plyY);   // プレイヤー描画
```

配列plyImgの添字に変数plyAnmを用いることにより、plyImg[0]～plyImg[3]を順に描画していきます。

ｉアプリのダウンロード

エミュレータでひととおりの動作確認ができたら、実機で最終確認を行います。電話がかかってきて、ｉアプリの動作が中断したときなど、エミュレータでは完全に再現できない部分もあり、また実機での動作速度など細かいところは微妙に異なるからです。

実機で動かすには、ｉアプリを実行するのに必要なファイルを携帯電話機に転送しなければなりませんが、PCから携帯電話機に直接転送することはできません。Web(http)サーバを用意し、ｉアプリをダウンロードできるようにする必要があります。

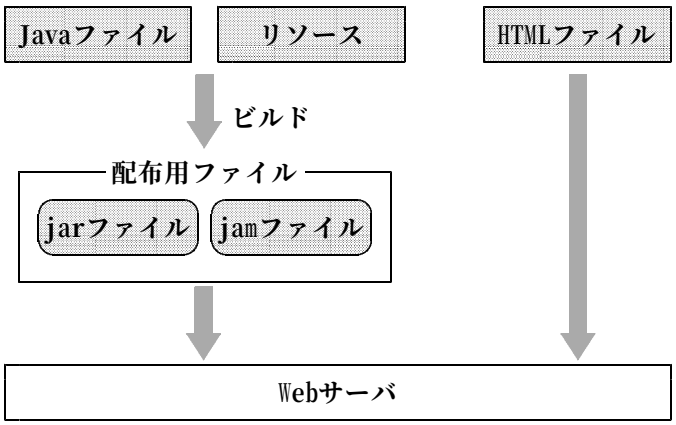
ｉアプリをビルドしたときに作成されるjamファイル(ｉアプリの情報が記述されているファイル)とjarファイル(複数のクラスファイルやリソースファイルをひとつにまとめたファイル)をWebサーバにftpなどで転送します。

次に、これらのファイルを携帯電話機にダウンロードできるように、以下のようなHTMLファイルを作成します。

```
<HTML>
<BODY>
<OBJECT declare id="application" data="ファイル名.jam" type="application/x-jam">
</OBJECT>
<A ijam="#application" href="iAppliNotSupported.html">ダウンロード</a>
</BODY>
</HTML>
```

<OBJECT>タグで指定したidと<A>タグで指定した「ijam="#」に続く文字列(上の例ではapplication)が一致している必要があります。また、<A>タグの「href="」に続く文字列(上の例ではiAppliNotSupported.html)は、ｉアプリ対応の携帯電話以外の機種やPC等から閲覧したときに、「ダウンロード」のリンクを選択したときに表示させるページへのリンクです。このページには、「ｉアプリがサポートされていません」などのメッセージを表示させるとよいでしょう。

上記のHTMLファイルもjamファイルとjarファイルがあるのと同じ場所に転送します。そして、HTMLファイルのURLを携帯電話機で指定すると、「ダウンロード」というリンクが表示されるので、[選択]キーを押してダウンロードします。ダウンロードが終われば、電話機を操作してｉアプリを起動することができます。



課　題

ｉアプリを携帯電話機にダウンロードし、実機で動作確認しましょう。