第5章　簡単なゲームを作ってみよう

この章では、今までで学んだ知識を元に、簡単なゲームを作っていきたいと思います。

今回は、以下のような、主人公をキー操作で動かすようなプログラムを作ってみましょう。
image::img/sample-result[sample]

新しくプロジェクトを作る

この章でも、新しくプロジェクトを作ってゲームを作っていきましょう。
先ほどと同じように、新しいプロジェクトを作ります。次に、先ほど作ったプログラム（３つのJavaファイル）をこのプロジェクトにコピーします。

主人公を描いてみる

早速、主人公を描画してみましょう。画像を使って描いてもいいですが、とりあえず、四角形を主人公として描いてみます。

以下のコードを追加してみましょう。追加する場所は、４つ`if`文をかいたところの直後に追加してください。

```
g.setColor(Color.GREEN);
g.fillRect(200,200,50,50);
```

このままでは、主人公が描画されただけで動きませんので、次は主人公を動かすコードをかいてみましょう。

主人公を動かす

主人公を動かすにはどうすればいいでしょうか。ずばり、主人公を描画する位置を変えるしかないですね。
位置を変えるには、位置の情報を毎回どこかに保存して、随時それの情報を見ることができるような、”箱”のようなものが必要です。

その”箱”とは、*変数*のことです！ 位置を格納する変数`x`と`ｙ`を宣言しましょう。

ここでは、`while`文の直前に宣言します。

```
//while文の上に宣言
int x = 200;
int y = 200;
```

変数を宣言したところで、今度は変数を値を変えられるようにしましょう。例えば、`Up`キーを押すと、上に動く、みたいな感じです。

これをするために、`if`文の中をかきかえましょう。

```
            if(Key.up){
                y -= 5;
            }
            if(Key.down){
                y += 5;
            }
            if(Key.right){
                x += 5;
            }
            if(Key.left){
                x -= 5;
            }
```

少し注意することとして、y座標は、上にいくほど値が減っていくので、`Up`キーが押された時は、yの値を減らしています。

最後に、主人公を描く位置を、xとyにすればOKです。

```
g.setColor(Color.GREEN);
g.fillRect(x,y,50,50);
```

これで無事、動くようになったはずです。

y -= 5って何？

`y -= 5`は、`y = y - 5`と同じ命令です。

では、`y = y - 5`はどういう命令なのかというと、`y - 5`した値を、再び`y`に代入するという処理です。 つまり、`y`の値を5だけ減らして`y`を更新します。

クラスを作ってみよう

このままでも、ゲームは実行できますが、実は、もっとわかりやすくプログラムを記述する方法があります。 クラスというものを使います。

クラスを説明する前に、次のコードを見てみましょう。

```
        while(true){
            g.setColor(Color.WHITE);
            g.fillRect(0,0,width, height);

            //追加！
            player.move();
            player.draw(g);

            wg.drawImage(img, 0, 0, this);
            try {
                Thread.sleep(10);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
```

さっきよりも、主人公が何をしているか見やすくなったと思いませんか？　playerがmoveして、さらにplayerをdrawしているのが一目でわかるようになったと思います。

このようなコードを書くためには、Player`クラスを新たに作る必要があります。　`Project Explorer`のところを右クリックして、`New`→`Class`で新しいクラスを作りましょう。クラス名は、`Player`で。

`move`メソッドを定義する。

次に考えるべきことは、`Player`が何をするかです。このゲームでプレイヤーは移動をします。つまり、`Player`は移動するということを定義すればいいわけです。

クラスの中で、移動するなどの動作の定義のことを、*メソッド*と言います。
では、移動の処理を行う、`move`というメソッドを定義してみましょう。

```
public class Player {
    public void move(){

    }
}
```

今度は、`move`メソッドの具体的な処理を記述していきましょう。
`move`メソッドは、キーが押されたら、その方向にプレイヤーが動けばいいわけです。

```
public class Player {
    public void move(){
        if(Key.up){
            y -= 5;
        }
        if(Key.down){
            y += 5;
        }
        if(Key.right){
            x += 5;
        }
        if(Key.left){
            x -= 5;
        }
    }
}
```

そう言えば、このコードどこかで見たような・・・。

そうです！whileループの中に記述していたコードと全く一緒です！　先ほどかいたwhileループの中のこの処理を、`move`メソッド内にコピペしましょう。

フィールドを追加する

先ほど`move`メソッドを定義した時に、xとyの変数の宣言はコピペせずにやりましたね。では、xとyはどこに宣言すればいいのでしょう？

結論からいうと、以下の場所に宣言します。

```
public class Player {
    int x;
    int y;

    public void move(){
        if(Key.up){
            y -= 5;
        }
        if(Key.down){
            y += 5;
        }
        if(Key.right){
            x += 5;
        }
        if(Key.left){
            x -= 5;
        }
    }
}
```

このxとyは、`move`メソッド内に宣言してはいけません。なぜなら、メソッド内に定義すると、その変数は*ローカル変数*となり、メソッドの実行が終了するたびに変数の状態が消えてしまうからです。

これを防ぐため、メソッドの外に定義します。このメソッドの外に定義された変数のことを、*フィールド*と呼びます。 こうすれば、`Player`はきちんと位置の情報を保持することが可能になります。

コンストラクタを作る

ここで、フィールドを見てみると、フィールドが初期化されていませんね。　　フィールドの初期化は、*コンストラクタ*というものを使います。

`Player`クラス内に以下のように定義します。

```
Player(int x, int y){
    this.x = x;
    this.y = y;
}
```

`this.x`は、フィールドのxを指し、ただのxは、コンストラクタの引数に渡ってきたxです。

他のクラスから使う時に、xとyの値を指定してPlayerを作ることができます。

`draw`メソッドを定義する。

`move`メソッドを定義したように、今度は`draw`メソッドも定義してみましょう。　　また、whileループの中の一部分をコピペすれば良いですね。

```
public class Player {
    int x;
    int y;

    Player(int x, int y){
        this.x = x;
        this.y = y;
    }

    public void move(){
        if(Key.up){
            y -= 5;
        }
        if(Key.down){
            y += 5;
        }
        if(Key.right){
            x += 5;
        }
        if(Key.left){
            x -= 5;
        }
    }

    // 追加！
    public void draw(Graphics g){
        g.setColor(Color.GREEN);
        g.fillRect(x,y,50,50);
    }
}
```

これで、`draw`メソッドが出来上がりました！

さあ、あとは、`MyJPanel`の方を少し直して完成です。

MyJPanelの修正

whileループの中を以下のように修正します。

```
        while(true){
            g.setColor(Color.WHITE);
            g.fillRect(0,0,width, height);

            //追加！
            player.move();
            player.draw(g);

            wg.drawImage(img, 0, 0, this);
            try {
                Thread.sleep(10);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
```

でも、このままでは実行できません。実は、クラスを使うには、クラスの*インスタンス*というものを作る必要があります。　　とりあえず、このクラスのフィールドに、`Player player = new Player(0,0);`のようなものを追加してみましょう。

```
import javax.swing.*;
import java.awt.*;

public class MyJPanel2 extends JPanel {
    Player player = new Player(0,0); //追加

    public void game(int width, int height){
        Image img = createImage(width, height);
        Graphics g = img.getGraphics();
        Graphics wg = getGraphics();

        while(true){
            g.setColor(Color.WHITE);
            g.fillRect(0,0,width, height);

            //追加！
            player.move();
            player.draw(g);

            wg.drawImage(img, 0, 0, this);
            try {
                Thread.sleep(10);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
    }
}
```

`new Player(0,0)`とすれば、初期座標(0,0)の`Player`を作ることができます。 もちろん、`new Player(200,200)`とすれば、初期座標(200,200)の`Player`を作ることができます。

こうすれば完成です！
あとは、自分で色々試してみるのもありですし、画像を読み込んで遊んでみても面白いと思います。