

2 初めての Java

2.1 Java をインストール

2.1.1 ダウンロード&インストール

JAVA でプログラミングするためには、まず JavaSDK というものをインストールしなければいけません。簡単に言えば、Java の素ですね。JavaSDK はインターネットからダウンロードすることができます。早速ダウンロードしてみましょう。

ここでは、ダウンロードの方法を説明しますが、なにぶんにも Java はバージョンアップのスピードが速いです。1,2年たつとあっというまにダウンロードの方法が変更されてしまったりします。そのため、本章の内容はあくまでもこれは参考程度です。インストール方法については、いろいろ参考になる HP がありますので、調べてみてください。

さて、Java のプログラミングを行うためには、Java の **JDK (Java Development Kit)** と呼ばれるキットをダウンロードする必要があります。JDK は Sun のホームページからダウンロードできます。ちなみに、JavaSDK の最新バージョンは JAVA6 です。(2007 年 5 月現在)

ダウンロードは、

<http://java.sun.com/javase/ja/6/download.html>

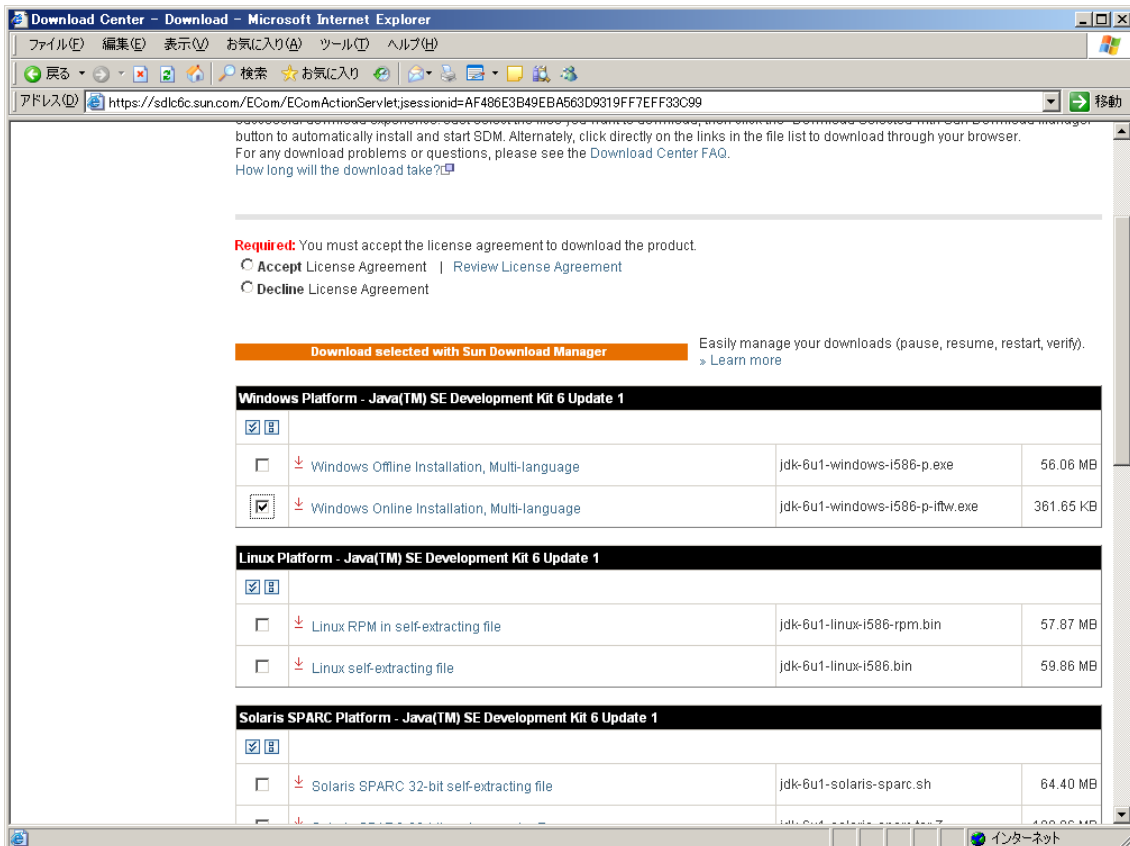
からできます (これまた 2007 年 5 月現在)。

ページ中央付近に「JDK 6 のダウンロード」というリンクがありますので、ここをクリック。



JDK ダウンロード・その 1

ライセンスに同意する必要がありますので、Accept にチェックを入れて Continue をクリックしましょう。そして、Windows をお使いならば Windows Platform の Windows Offline Installation または Windows Installation をクリックしてください。Linux ご利用の方は Linux Platform です。まあ、Linux をお使いの方なら Java のインストールくらいは自力でできると思いますが。



JDK ダウンロード・その 2

さて、ダウンロード終了後、ファイルをダブルクリックすればインストールが開始されます。通常はデフォルトの設定のまま進めていけば大丈夫でしょう。あまり難しい設定変更はしないほうが後々楽チンです。「続き」をどんどん押していけばインストール完了です。

ここでは、C:\Program Files\Java\jdk1.6.0_01 にインストールされたと仮定して話を進めていきます。別のフォルダにインストールされた場合は、今後インストールフォルダを読み替えてください。

さあ、これでもう、いつでも JAVA を使うことが出来ます。やったね・・・とは行きません。

次に、Java の設定をしなければいけません。

コラム

JDK というのは、Java Development Kit の略です。ようするに、Java ソフトウェア開発者向けのツールのセットという意味です。JDK には、Java のプログラムをコンピュータが理解できるように翻訳するコンパイラ(javac)と、Java を実行するための JavaVM(Java バージャルマシン)それに、作ったプログラムの仕様書を自動作成する JavaDoc など色々なツールが含まれています。

2.1.2 設定

Java の設定で一番大切なのが **PATH** と **CLASSPATH** の設定です。

Java を初めてインストールする人(何度もインストールした人でも)間違いやすいポイントですので気をつけてください。

この二つは、**環境変数**の一種です。そのため、Windows と Linux など設定方法が異なりますので、注意が必要です。ですが、ここでは Windows での設定方法だけを解説します。

まずは、コントロールパネルを開きます。そこにある「システム」を開いてください。システムのプロパティというウィンドウが開いたと思います。

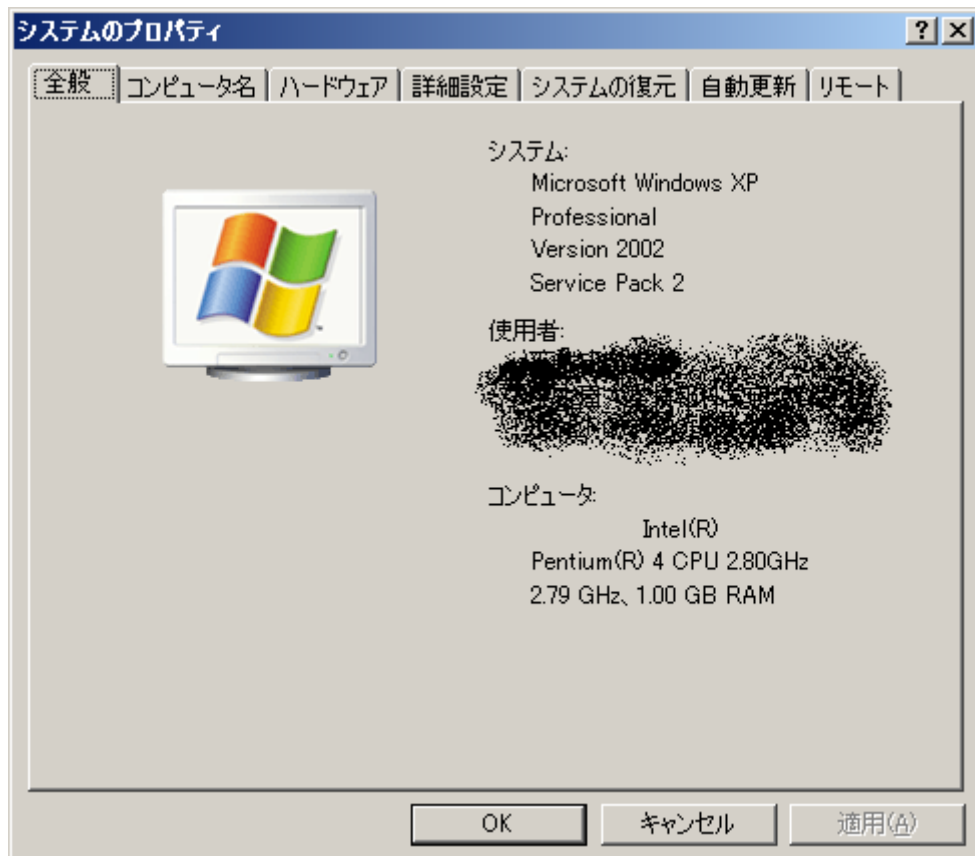


図 1 システムのプロパティ

次に、詳細設定タブをクリックします。

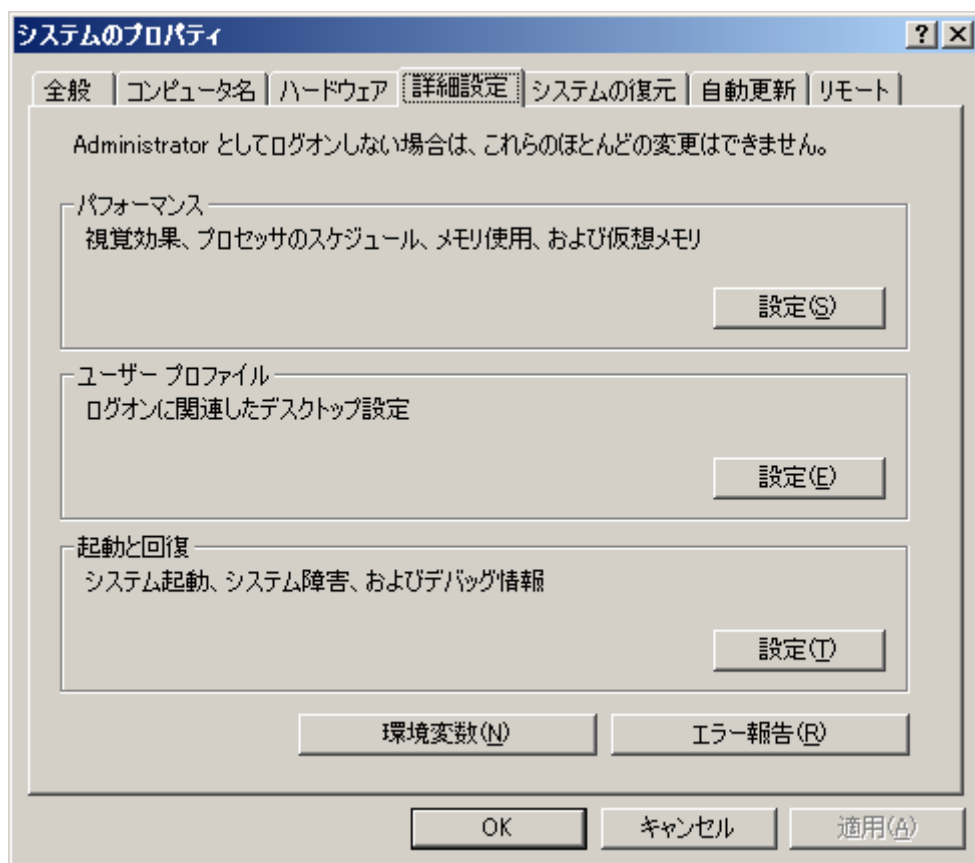


図 2 詳細設定

この下のほうに環境変数というボタンがあるのがわかると思います。このボタンをクリックしてください。

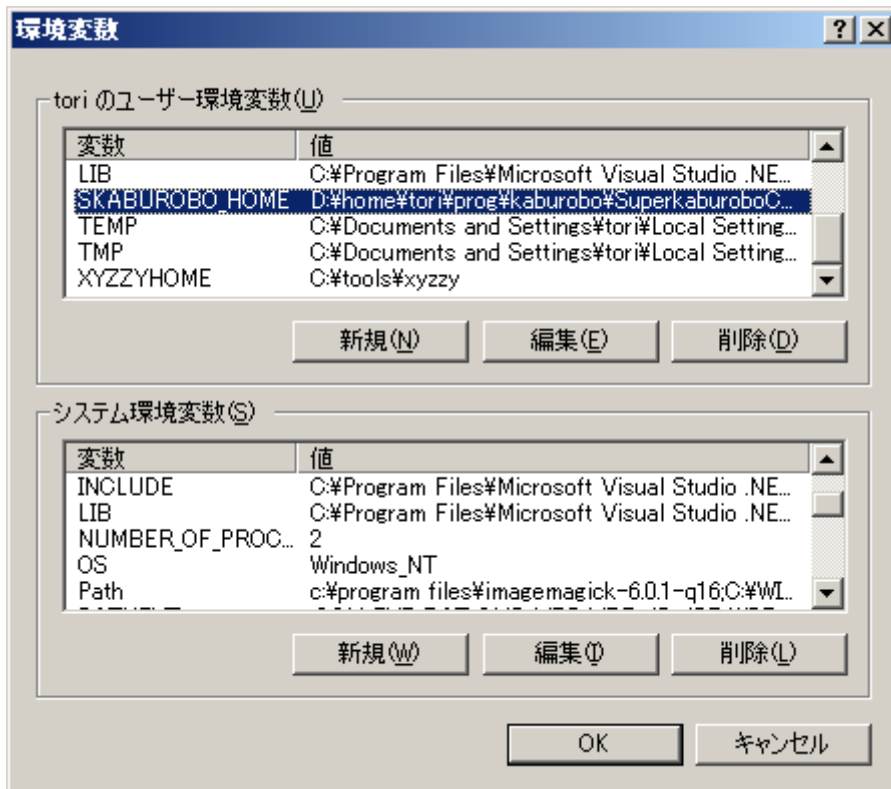


図 3 環境設定

このような、環境設定ウィンドウが開きます。

ここで、ユーザー環境変数とシステム環境変数をみてください。この中に、変数 PATH（または Path, path）はあるでしょうか？もしあれば、そこをクリックして「編集」ボタンをおしてください。もしなければ、「新規」ボタンがありますので、これをクリックします。

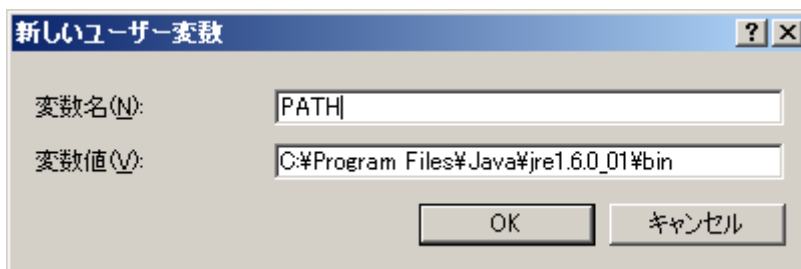


図 4 PATH の設定

そして、

変数名=PATH

変数値=Java をインストールしたフォルダ¥bin

と記述しましょう。

ここでは、C:\Program Files\Java\jre1.6.0_01 に Java をインストールしたので、変数値は

C:\Program Files\Java\jre1.6.0_01\bin

となります。

すでにあって、編集する場合はこれまでの値と;で区切ります。

これまでの変数値:C:\Program Files\Java\jre1.6.0_01\bin

次に、CLASSPATH を設定します。同様に、すでに CLASSPATH があれば編集を、なければ新規をクリックします。

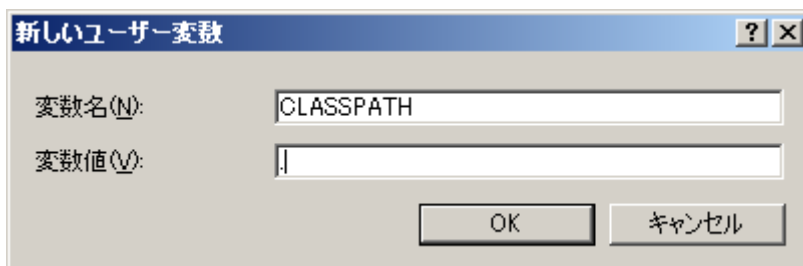


図 5 CLASSPATH の設定

そして、

変数名=CLASSPATH

変数値=.

とします。これで設定完了です。

さっそく Java を動かしてみ・・・たいところですが、そのまえに本書の教材をダウンロードしてください。

それをつかって Java を楽しんできたいと思います。

2.2 Java を動かしてみる

2.2.1 教材のインストール

教材をインストールしてみましょう。教材は秀和システムのホームページページ (URL) からダウンロードすることができます。ファイルは zip 形式で圧縮されていますので、ダウンロードしたら、適当な場所に解凍してください。ここでは、C:\Java に展開したものと話を進めていきたいと思います。

中には、Maze.jar というファイルと、Sample.java そして、User.java というファイルが入っています。これらを C:\Java に解凍します。

別のフォルダに解凍する場合は、今後 C:\Java を解凍したフォルダに読み替えてください。

2.2.2 教材のコンパイル

さて、ここまでできたら、次はこの教材をコンパイルします。

コンパイルというのは、あるプログラムをコンピュータが実行できる形に変更することを言います。

コンパイルはまず、コマンドプロンプトを開くところからはじめます。

スタートメニューからプログラム⇒アクセサリ⇒コマンドプロンプトと開きます。

コマンドプロンプトが開いたら、

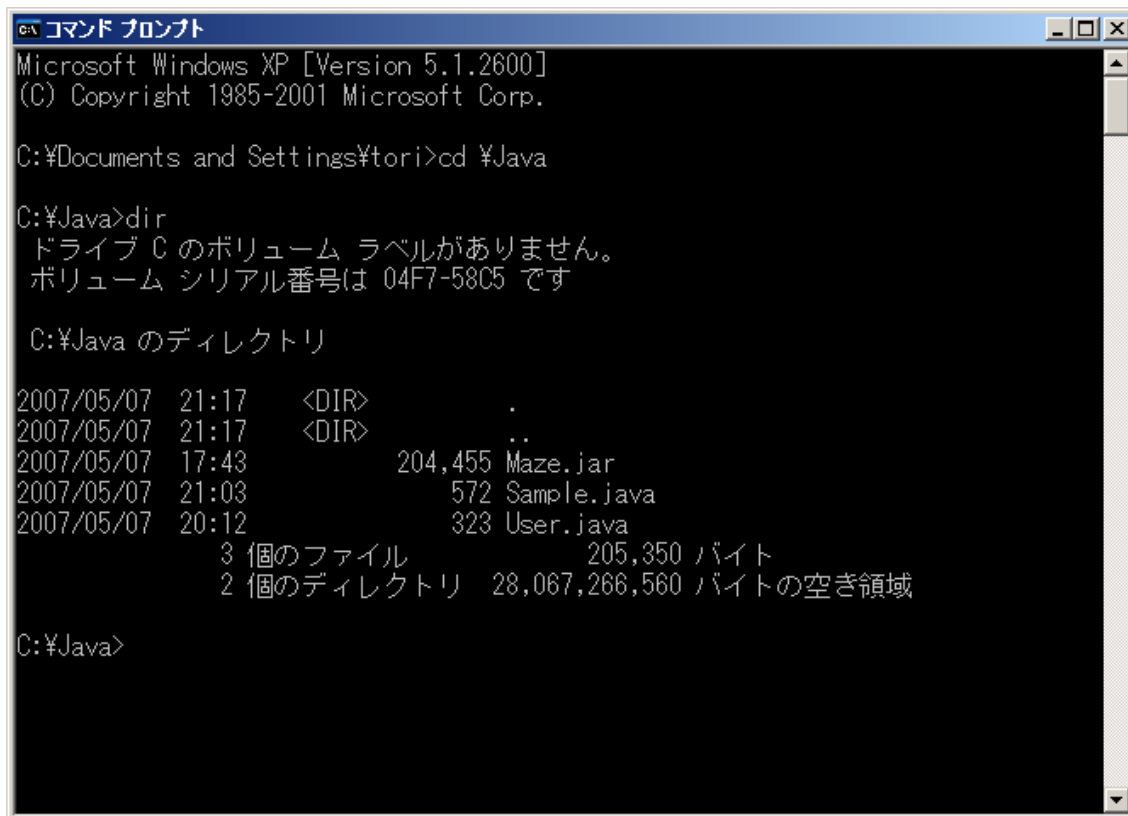
```
C:> cd %Java
```

と打ってください。教材を展開したフォルダに移動するはずですが、C:%Java 以外の場所に展開した方は展開したフォルダに移動してください。

ここで、

```
C:%Java%> dir
```

と打つと、Maze.jar などのファイルがあることが確認されると思います。



```
コマンド プロンプト
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:%Documents and Settings%tori>cd %Java

C:%Java>dir
ドライブ C のボリューム ラベルがありません。
ボリューム シリアル番号は 04F7-58C5 です

C:%Java のディレクトリ

2007/05/07  21:17    <DIR>          .
2007/05/07  21:17    <DIR>          ..
2007/05/07  17:43                204,455 Maze.jar
2007/05/07  21:03                572 Sample.java
2007/05/07  20:12                323 User.java
               3 個のファイル                205,350 バイト
               2 個のディレクトリ  28,067,266,560 バイトの空き領域

C:%Java>
```

図 6 コマンドプロンプト

これを確認したら、

```
C:%Java> javac -cp Maze.jar Sample.java
```

とコマンドをうちます。

なにやら複雑なコマンドですが、あまり気にしないでください。こういうものを打ち込むことで、教材を実行する準備が出来るんだと思ってもらえれば結構です。

これを行うと、

```
Sample.class
```


というファイルができるはずですが、これが **Java** を実行するために必要なファイルになります。

もしエラーメッセージが出た場合は、**Java** が正しくインストールされていないか、**Path** の設定がされていない可能性があります。

```
C:¥Java¥> javac -cp Maze.jar Sample.java
'javac' は、内部コマンドまたは外部コマンド、
操作可能なプログラムまたはバッチ ファイルとして認識されていません。
```

などと出た場合は、前節の **Path** の設定を確認してください。

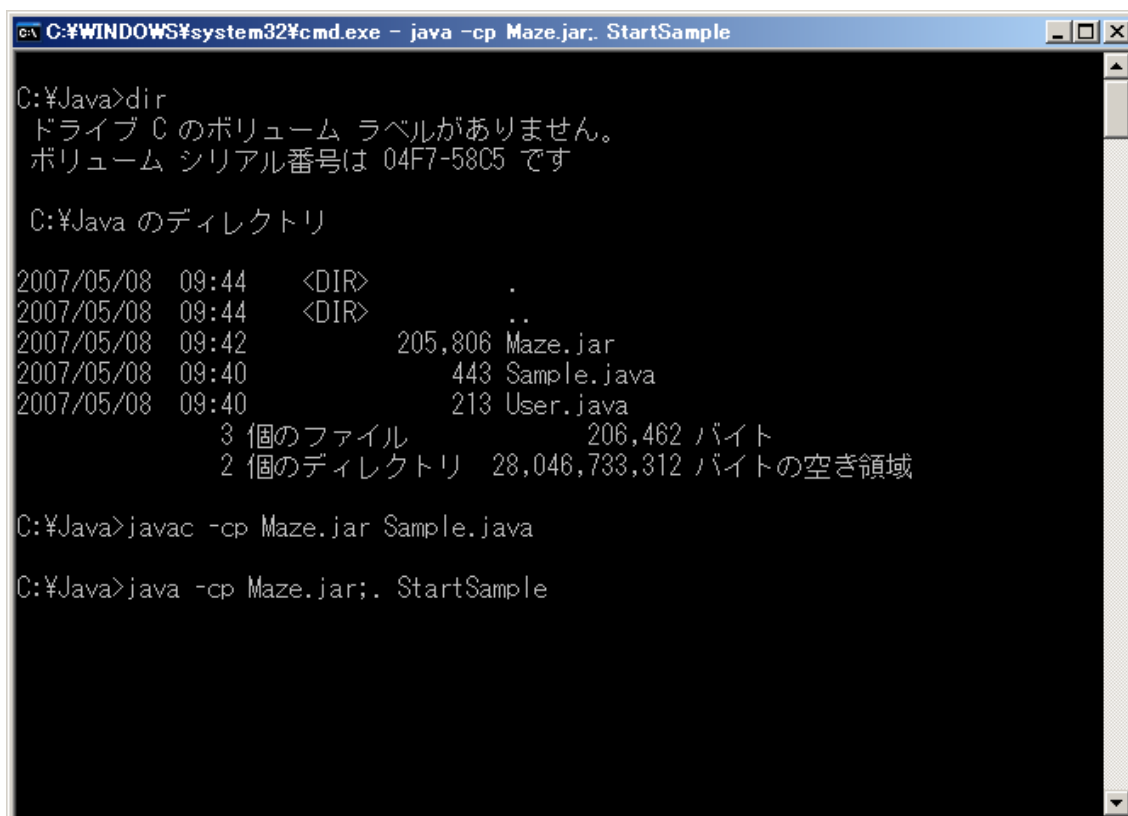
2.2.3 教材の実行

コンパイルが終わったら、**Sample.class** というファイルが出来ているのが確認できるはずです。

ここまできたら、あとは実行するだけ。ためしに実行してみましょう。

```
C:¥Java¥> java -cp Maze.jar;. Sample
```

これもまた奇妙なコマンドですが、これが **Java** のプログラムを動かすためのコマンドです。ここも、こういうものだと思って覚えてください。では、教材を実行してみましょう。



```
C:\WINDOWS\system32\cmd.exe - java -cp Maze.jar;. StartSample

C:¥Java>dir
ドライブ C のボリューム ラベルがありません。
ボリューム シリアル番号は 04F7-58C5 です

C:¥Java のディレクトリ

2007/05/08  09:44    <DIR>          .
2007/05/08  09:44    <DIR>          ..
2007/05/08  09:42             205,806 Maze.jar
2007/05/08  09:40              443 Sample.java
2007/05/08  09:40              213 User.java
               3 個のファイル             206,462 バイト
               2 個のディレクトリ 28,046,733,312 バイトの空き領域

C:¥Java>javac -cp Maze.jar Sample.java

C:¥Java>java -cp Maze.jar;. StartSample
```

図 7 教材を実行したコマンドプロンプトの決定的瞬間

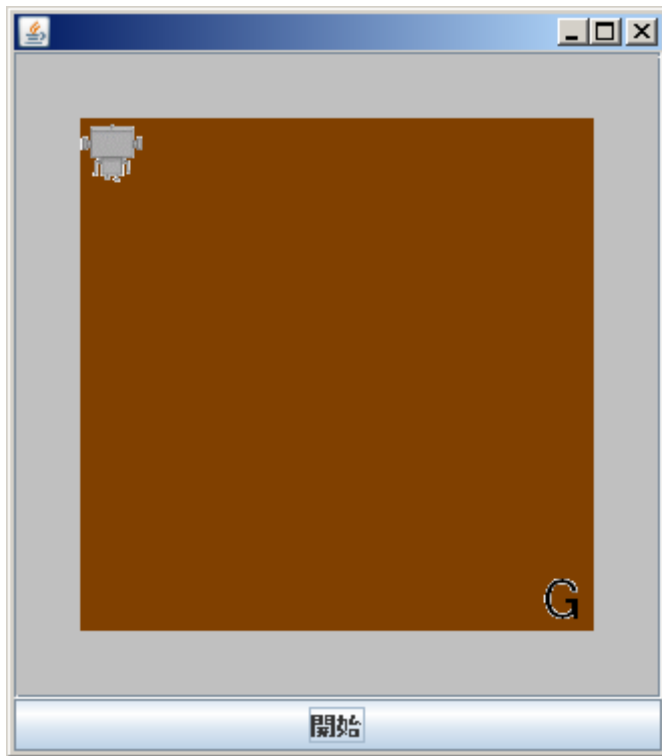


図 8 サンプルプログラム

なにやらウィンドウとロボットが登場しました。

ここで、開始ボタンを押すと、左上にいたロボットがうのように動いて右下の G まで移動します。

これは、ロボット君をゴールまで誘導するプログラムなんですね。なんのこっちゃよくわからないプログラムですが、とにかく動きました。万歳。

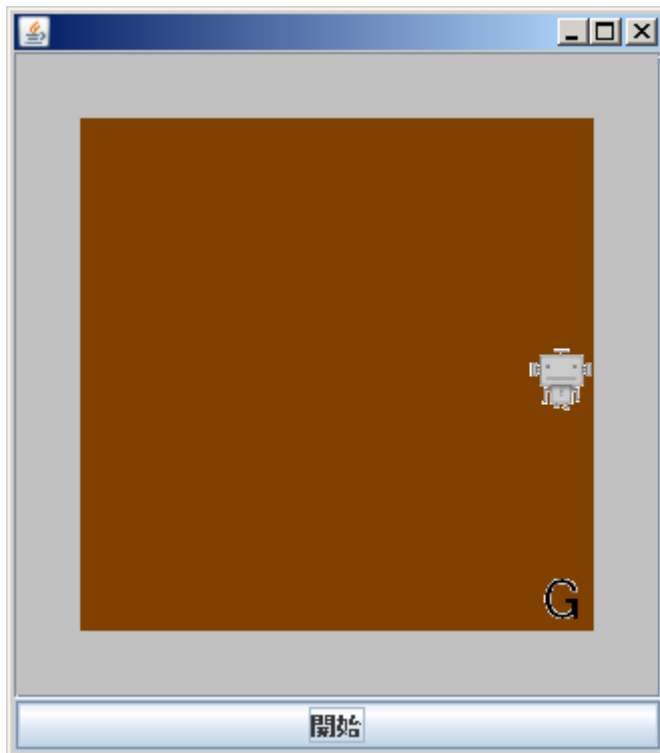


図 9 ロボット君がゴールに向かって猛進中

2.2.4 教材の確認

せっかく動いたプログラムです。このプログラムがいったいどうやってロボット君をゴールに導いているか興味ありますよね。その謎を解くために、**Sample.java** をメモ帳などのテキストエディタで開いてみてください。テキストエディタじゃないと駄目ですよ。ワードなどで開いたら正しく動かなくなってしまいますので、注意してください。テキストエディタが良くわからない人はメモ帳で開いとけば無難です。

すると、こんなものが見えたと思います。

```
import jp.co.shuwasystem.java.maze.map.MazeManager;
import jp.co.shuwasystem.java.maze.map.Robot;

public class Sample extends MazeManager{

    protected void action(Robot robot) {

        robot.right();
        robot.right();
        robot.right();
        robot.down();
    }
}
```

```
        robot.down();
        robot.down();
        robot.right();
        robot.right();
        robot.right();
        robot.right();

        robot.down();
        robot.down();
        robot.down();
        robot.down();
    }
}
```

ここで重要なのは太字の部分だけです。他の部分はおまじないなので気にしないでください。

ポイントは、

```
        robot.right();
        robot.down();
```

です。

このように書いてあると、ロボットがウィンドウの中で右や下へ移動するのです。

でも、こんな決められた動きをしているロボットを見ても面白くないですよ？自分で自由にロボットを動かしてみたいことと思います。そこで、次は、いよいよ **Java** プログラミングによって自分でロボットを動かしてみたいと思います。

2.2.5 自分で作成

自分でロボットを動かすために利用するのは、**User.java** というファイルです。ここに、**Java** プログラムを記述してコンパイルすることによってプログラムを動かすことが出来るようになります。

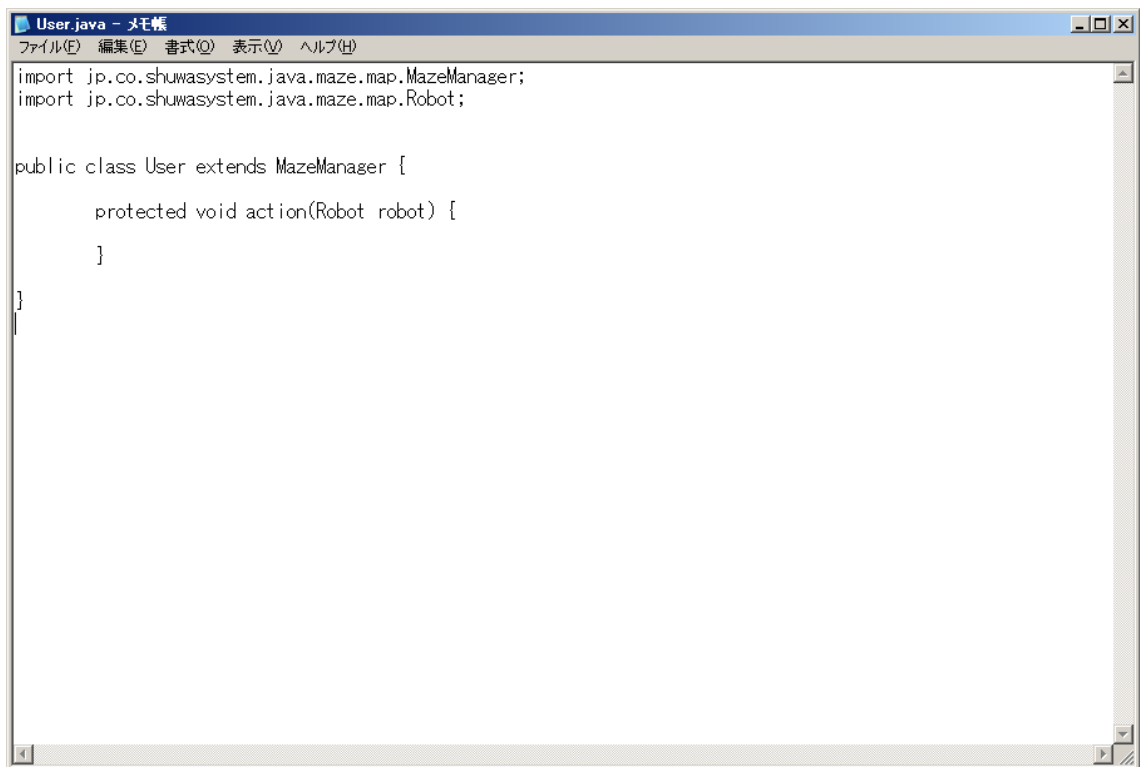
では、まずは **User.java** を開いてみましょう。図のようなものが見えたかと思います。これがこれから皆さんがプログラムを記述していくひな形となるものです。

まあ、何はともあれまずは動かしてみるのが一番ですよ。

早速、**User.java** をコンパイルしてください。コンパイルは、

```
C:¥Java¥> javac -cp Maze.jar User.java
```

とします。またまた変なコマンドですが、難しいことは余り考えないでこの通りに入れてください。



```
import jp.co.shuwasystem.java.maze.map.MazeManager;
import jp.co.shuwasystem.java.maze.map.Robot;

public class User extends MazeManager {
    protected void action(Robot robot) {
    }
}
```

図 10 ついに User.java を開く

コンパイルが無事終了したら、次は実行です。実行するためのコマンドは以下のとおりです。

```
C:¥Java>java -cp Maze.jar;. StartMaze
```

これで自分が考えたとおりにロボットが動きます！まあ、まだ何も作っていないので、まったく動かないのですが。

試してみましょう。

```
C:\WINDOWS\system32\cmd.exe - java -cp Maze.jar;. StartMaze
C:\Java>javac -cp Maze.jar User.java
C:\Java>java -cp Maze.jar;. StartMaze
ゴールに到達できませんでした
```

図 11 コンパイル, そして実行・・・そして失敗.

どうでしょうか? ウィンドウが開いた後, 「開始」 ボタンを押すと, 何もせずにゴールにたどり着けなかったメッセージが表示されると思います. ウィンドウ上には, FAIL の文字が悲しく流れていきます. ちょっと寂しいですね.

あまりにも寂しいので, 少し歩くようにしましょうか.

User.java を以下のように変更してください.

```
import jp.co.shuwasystem.java.maze.map.MazeManager;
import jp.co.shuwasystem.java.maze.map.Robot;

public class User extends MazeManager {

    protected void action(Robot robot) {

        robot.right();

    }

}
```

ポイントは, robot.right();です. これで, ロボットを一歩右に歩かせるプログラムになります.



図 12 ゴールできず

このとき、`robot.right()`は必ず半角で書いてください。全角では駄目ですよ。また、一字一句間違えずに打ちましょう。少しくらいの間違いならいいかな？と思って書くと失敗します。コンピュータは融通が利きません。きっちりかきましょう。

さて、Java のプログラムは一回書き換えたら、必ずコンパイルをしなければいけません。ちょっと面倒ですが、一種の手続ききですので必ず行うようにしてください。コンパイルのコマンドは、

```
C:¥Java¥> javac -cp Maze.jar User.java
```

ですよ。

このとき、

```
C:¥Java>javac -cp Maze.jar User.java
```

```
User.java:8: 文ではありません。
```

```
    robot,right();
```

```
    ^
```

```
User.java:8: ';' がありません。
```

```
    robot,right();
```

```
    ^
```

エラー 2 個

こんなメッセージがでることがあります。ポイントは、最後の「エラー」という部分で

すね. このようなメッセージが出たら, どこか間違えているよ, という意味になります.

Java のプログラムには書き方の文法のようなものがありますが, その文法が間違えている場合, コンパイルするときに教えてくれるのです. この例の場合だと,

```
robot.right();
```

の . が , になっていて,

```
robot,right();
```

になってしまっています. そこで, そこを修正してコンパイルしなおしてください.

コンパイルが無事終われば, なんのメッセージも表示されませんので, 安心して実行コマンドを入力してください.

```
C:¥Java>java -cp Maze.jar;. StartMaze
```

今度はどうなったでしょうか?

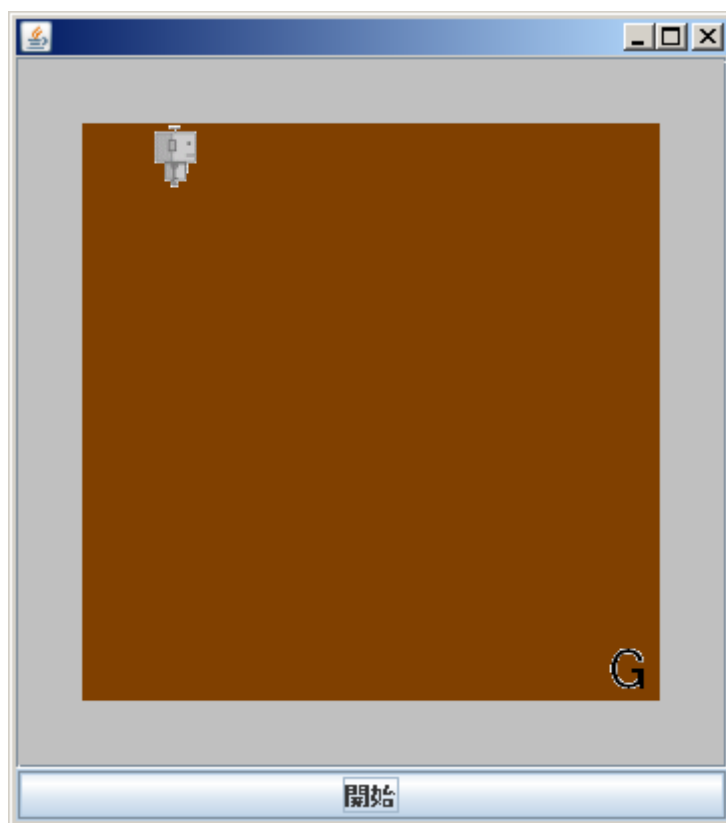


図 13 ついに移動に成功!

開始ボタンを押すと, ロボットが一步だけ右に動きます. まあ, それで終わりなので, ゴールに到達はできないのですが...

それでも, 自分の手でロボットを動かすことに成功したんですよ. これはすごい.

ロボット君を動かすためには, User.java に以下のいずれかを書き込んでいけばいいだけ

です。どんどん好きなところへ動かして行ってあげましょう。

移動方向	プログラム
右	robot.right();
左	robot.left();
上	robot.up();
下	robot.down();

例えば、二歩右へ行って、三歩下へ移動し、さらに二歩左へ移動して、最後に三歩上に移動してもともといた位置に戻りたいなら、

```
import jp.co.shuwasystem.java.maze.map.MazeManager;
import jp.co.shuwasystem.java.maze.map.Robot;

public class User extends MazeManager {

    protected void action(Robot robot) {

        robot.right();
        robot.right();

        robot.down();
        robot.down();
        robot.down();

        robot.left();
        robot.left();

        robot.up();
        robot.up();
        robot.up();

    }

}
```

これを実行すれば、ぐるぐるっとロボット君が回ってくれることが確認できるはずです。

これを利用して、ロボット君が無事ゴールにたどり着くようにプログラムを修正してみ

てください。

無事ゴールしたら、きっと喜ぶロボット君の姿が見られるはずです。

2.2.6 迷路を移動に挑戦！

ところで、単に四角い画面上を縦横無尽にロボットを走らせてゴールしても、イマイチ達成感がわきませんよね。わからないことにしてください。

そこで、せっかく **StartMaze** という名前のプログラムですので、ロボット君が走り回るフィールドを迷路にしてみましょう。

ロボット君が走り回るフィールドを迷路にするのは非常に簡単。

実行するときのコマンドを、

```
C:\Java>java -cp Maze.jar;. StartMaze 1
```

にしてください。

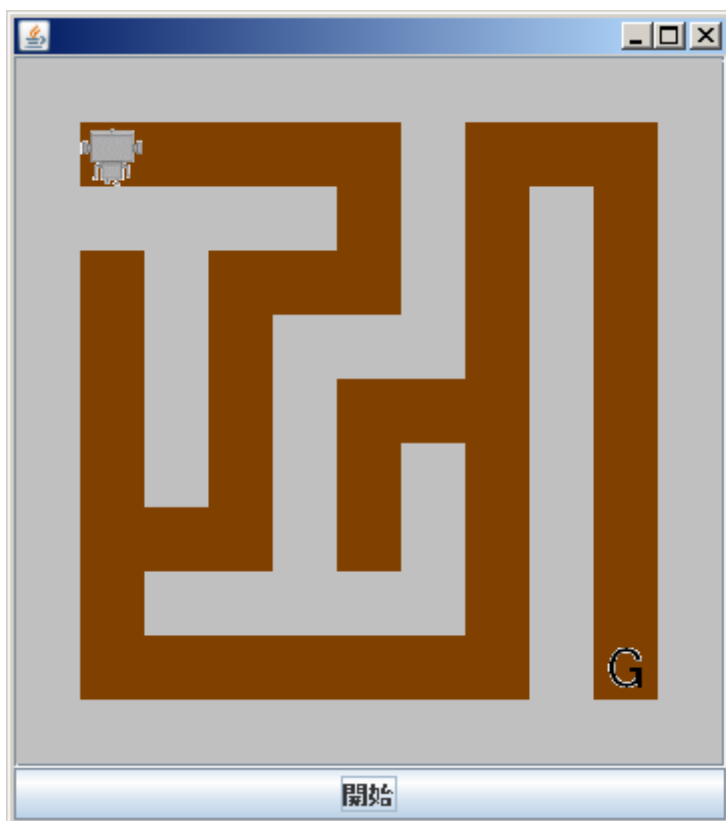


図 14 迷路と化したぞ！

先ほどまでは違って、壁がいっぱいあるフィールドです。ゴールにたどり着くには、かなり複雑な経路を通らなければいけませんね。

さあ、この迷路をクリアするためのプログラムを書いて見ましょう。

ロボット君は壁に当たるとその先には進めずに、そのまま次の命令を受け付けてしまいます。例えば、右側が壁のときに

```
robot.right();
```

などと書くと、何もしないことになります。この辺に注意してロボット君をゴールへ誘導してください。

コラム

実は、

```
java -cp Maze.jar;. StartMaze
```

の後に書くのは、1でなくても2でも10でも *hoge* でもなんでもかまいません。StartMazeの後にスペースで区切って適当な文字を書くと、ロボット君がチャレンジするフィールドが変化します。いろんなフィールドでロボット君をゴールに導いてみてください。

また、狭いフィールドで満足できないあなたは、

```
java -cp Maze.jar;. StartBigMaze 1
```

などとすると、巨大な迷路にチャレンジできます。ゴールへの誘導を目指して頑張ってください。

2.2.7 繰り返し処理を使ってスムーズな移動

先ほど迷路のゴールへたどり着くための操作は、所詮ロボットを動かすコマンドを並べただけのように見えて、イマイチプログラムを書いた気がしませんよね。

そこで、ちょっとだけJavaプログラムっぽくなるテクニックをお教えしましょう。それが、繰り返し処理です。詳しくは、第5章制御でお話しますが、今回は同じ処理を何度も繰り返すために使うものだと理解してください。

繰り返し処理を使ってロボットを移動させる場合、User.javaを以下のように書き換えます。

```
import jp.co.shuwasystem.java.maze.map.MazeManager;
import jp.co.shuwasystem.java.maze.map.Robot;

public class User extends MazeManager {

    protected void action(Robot robot) {
        for(int i = 0; i < 6; i++){
            robot.down();
        }
    }
}
```

お、なんかプログラムっぽくなった気がしませんか？

これをコンパイルして実行してみてください。

```
C:\¥Java>javac -cp Maze.jar User.java
```

```
C:\¥Java>java -cp Maze.jar;. StartMaze
```

ここでは簡単のため迷路ではなくてただっぴろいフィールドを使ってロボットを移動させていますが、`robot.down()`;というロボットを下に移動させるコマンドを一つしか書いていないのに、6回下に移動したと思います。

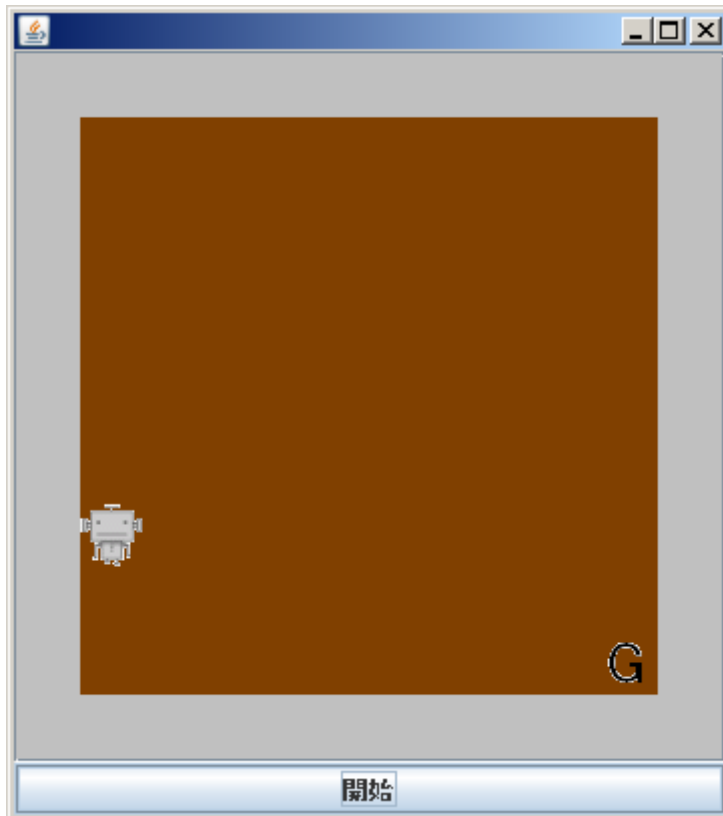


図 15 6 歩すすむ

このような繰り返し処理を **for ループ** と呼びます。ま、詳しいことは第 5 章まで待ってもらおうとして、とりあえずこのテクニックを使うと簡単に迷路を脱出できるかも知れません。例えば、右に 5 歩歩いて、下に 5 歩歩かせるならこんな風にしましょう。

ちなみに、for ループを使うとこんな事も出来ますよ。User.java をこんな風に変えてコンパイルしましょう。

```
import jp.co.shuwasystem.java.maze.map.MazeManager;  
import jp.co.shuwasystem.java.maze.map.Robot;
```

```
public class User extends MazeManager {  
  
    protected void action(Robot robot) {  
        for(int i = 0; i < 100; i++){  
            robot.right();  
            robot.down();  
            robot.left();  
            robot.up();  
        }  
    }  
}
```

これを実行するとちょっと面白いと思いますので、試してみてください。

2.3 コンパイルと実行

2.3.1 コンパイル

さて、ロボット君と迷路で遊ぶのもそろそろ飽きてきたことと思いますので、ここで Java の構造についてお話ししたいと思います。

Java のプログラムを行うときに、必ず**コンパイル**という作業があります。なんでこんな面倒くさいことをしなければいけないのかと疑問に思う方も多いかと思いますが、Java の構造を知れば納得いくと思います。

まず、Java でプログラムを書く場合、テキストファイルなどを使ってプログラムを書きます。先ほどのロボット君のプログラムなら、`User.java` などですね。

このプログラムは基本的にテキストで書かれていて**人間に見やすい構造**になっています。まあ、慣れないうちは「どこが見やすいんだ？」と思うかも知れませんが、筆者なんて下手な小説を読むより Java のプログラムの方が読みやすい気がします。ま、こうなっては人間としてはおしまいです。

で、人間に読みやすい構造というのは、コンピュータには理解しがたいものだったりするのです。そのため、コンパイルという作業を経てコンピュータに読みやすい形にプログラムを直す作業が必要なのです。

ちなみに、コンパイルした結果出来たコンピュータに読みやすいプログラムは、人間の目にはなんのこっちゃさっぱり分からないものになります。

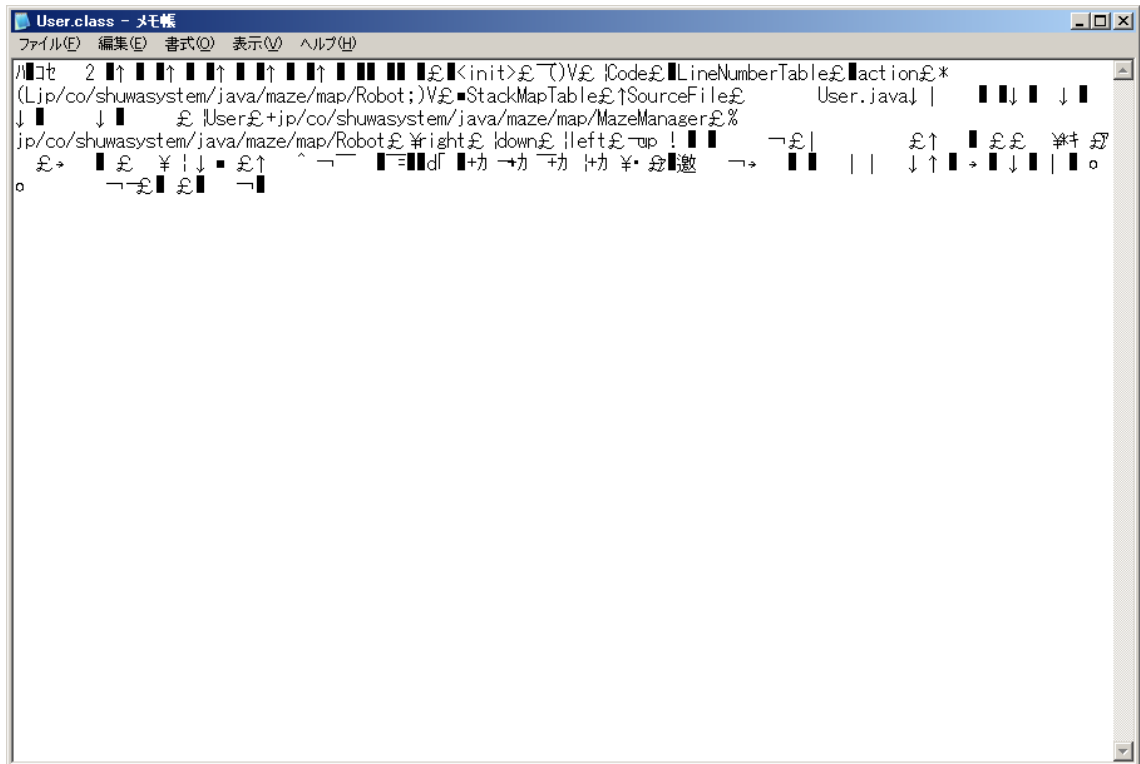


図 16 人間には理解不能なコンパイル後のファイル

このコンピュータ用に変換されたプログラムを **Java** は実行します。

ちなみに、**Java** の場合は人間がプログラムを書くファイルは

〇〇.java

というファイル名で、コンピュータ用にコンパイルしたファイルは同じ名前でも拡張子だけ変わり、

〇〇.class

という名前になります。

この **〇〇.class** ファイルを **Java** プログラムとして実行すると、初めてプログラムが起動するというわけです。

このように、人間がプログラムを書いたら、それをコンピュータがわかる形に一回翻訳してあげる、その作業がコンパイルだと思ってください。

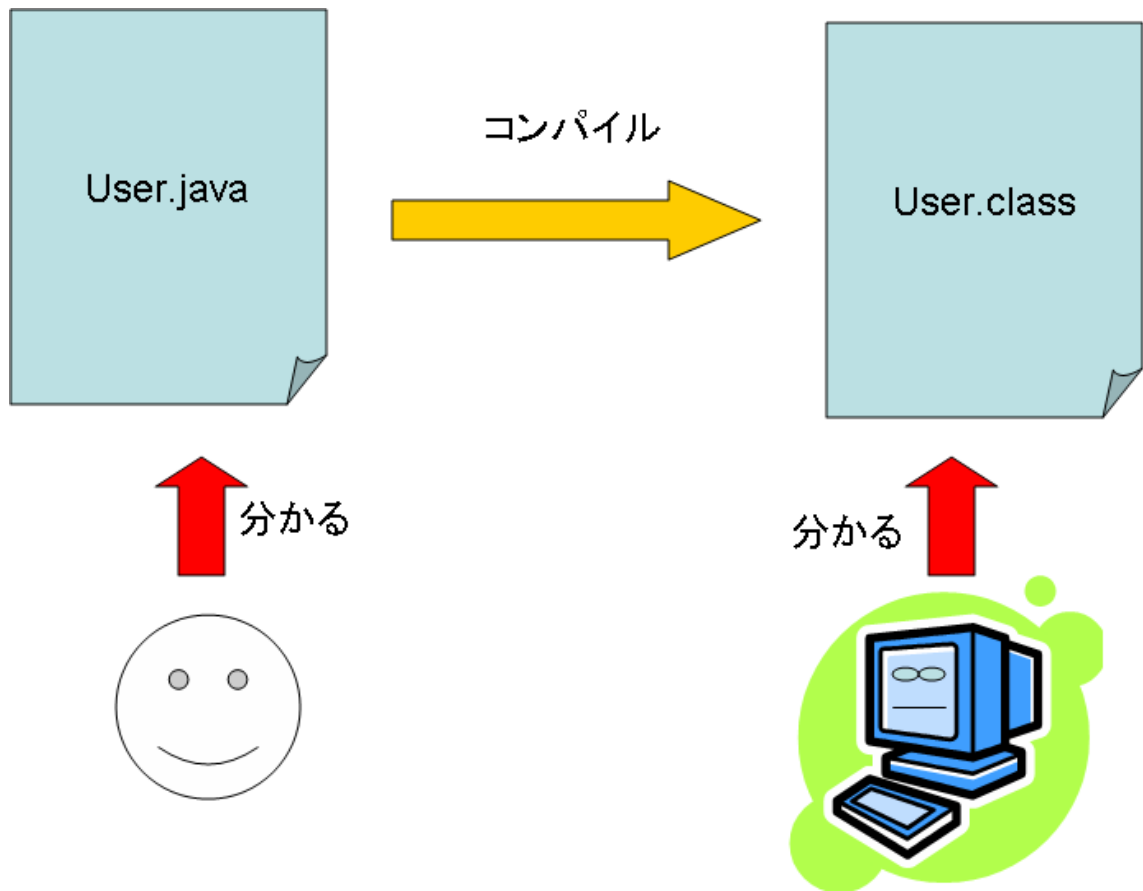


図 17 java ファイルは人間用，class ファイルはコンピュータ用

java ファイルをコンパイルして class ファイルにするためには、JDK についてくる `javac` というプログラムを利用します。

すでにロボット君プログラムでコンパイルは経験済みなので、大体わかると思いますが、基本的には、

```
C:¥Java> javac OO.java
```

とすることでコンパイルが可能です。

ただし、ロボット君プログラムの場合は自分で作ったプログラム以外のプログラムも利用していたため、コンパイル方法が少し複雑でした。この複雑なコンパイルについては、本書で扱う範囲を超えていますので、今回は割愛しておきます。今回は、こういうやり方もあるんだな～程度に思っておいてください。

2.3.2 Java プログラムの実行

Java のプログラムをコンパイルし終わったら、次は実行です。実行コマンドも割りと簡単で、基本的には、

```
C:¥Java> java OO
```

とすれば実行されます。

では、○○には何を入れればよいのでしょうか？これが実はちょっと難しくて、ここには「プログラム起動用クラス名」を入れることになります。

ロボット君プログラムでは、**StartSample** と **StartMaze** という二つのプログラムを実行していたと思いますが、これは筆者が用意した「プログラム起動用クラス」の名前なのです。クラスってのがなんだか第 7 章でお話しますが、そういうものがあると思ってください。

では、今度は筆者が用意したロボット君から離れて、完全自作プログラムを作ってみましょう。

2.3.3 完全自作プログラム

完全自作プログラムの第一弾は、単に文字列を表示するだけの単純なものです。

このプログラムを通して、**Java** プログラミングのルールとコンパイル、実行の手順を覚えるためのものですので、つまらないけど我慢してください。

まず、**HelloWorld.java** というファイルを作成します。このとき、名前を間違えないように気をつけてください。

Java プログラミングにおいてファイル名は非常に重要な意味を持ちます。ファイル名が違くと動かないこともありますので、注意しましょう。

HelloWorld.java というファイルを新規作成したら、ノートパッドなどで開き、以下のように入力してください。

```
public class HelloWorld{  
    static public void main(String[] args){  
  
    }  
}
```

これが、すべての **Java** プログラムの基本形となります。

これでコンパイルして実行しても良いのですが、これは何もしないプログラムですので、いくらなんでもつまらなすぎです。それに、ちゃんと実行されたのかもわかりません。

というわけで、プログラムを実行したら、ちゃんと実行したことが分かるように文字列を出力するようにします。

先ほどのプログラムを以下のように変更します。

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



```
}  
}
```

これでとりあえずのプログラムが完成です。

次に此をコンパイルします。2.3.2 で説明したとおり、コンパイルは `javac` というコマンドを使って行います。

```
C:¥Java> javac HelloWorld.java
```

これを実行してください。

エラーメッセージが表示されたら、どこか間違えていると意味ですので、間違えている箇所を修正しましょう。

エラーメッセージが表示されなければ、コンパイル成功です。念のため同じフォルダの中を見ると、

```
HelloWorld.class
```

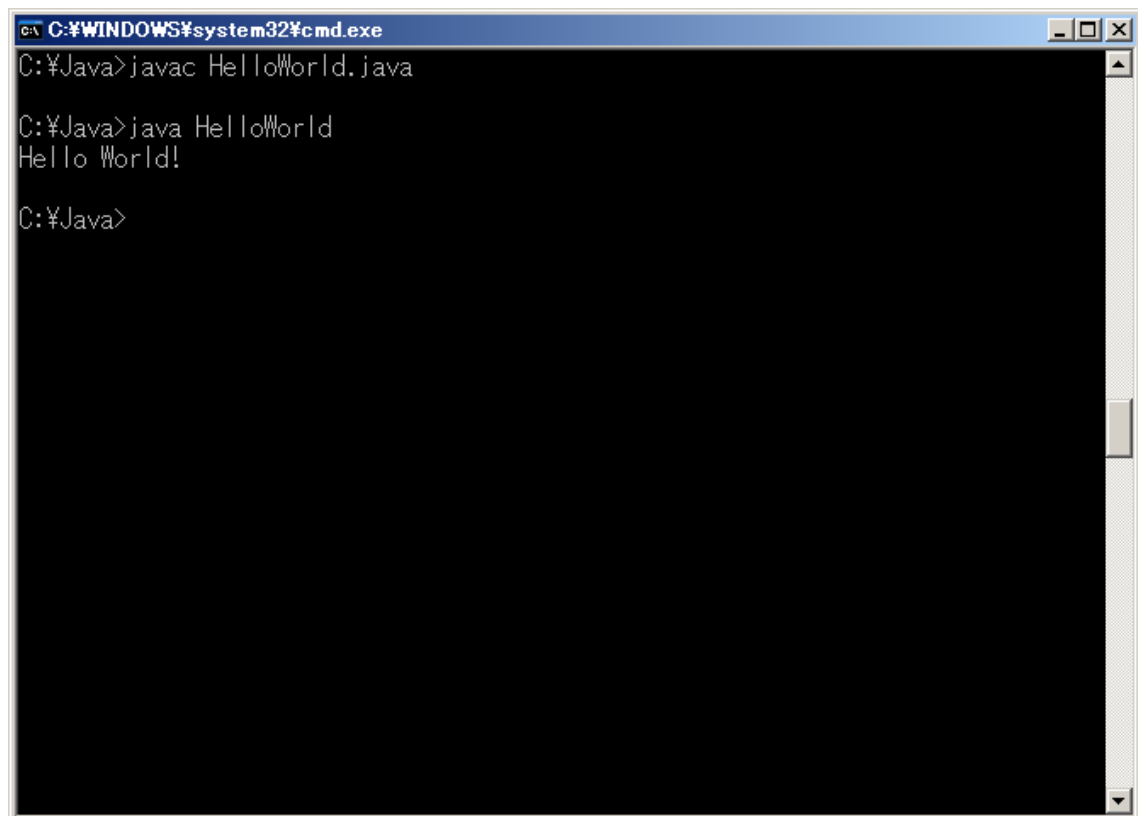
というファイルが増えているはずです。これが、コンピュータにわかりやすく変換されたプログラムですね。

次に、実行します。

実行コマンドは以下の通りです。

```
C:¥Java> java HelloWorld
```

さあ、実行するとどうなるでしょうか。

A screenshot of a Windows command prompt window. The title bar reads "C:\WINDOWS\system32\cmd.exe". The command prompt shows the following sequence of commands and output:

```
C:¥Java> javac HelloWorld.java  
  
C:¥Java> java HelloWorld  
Hello World!  
  
C:¥Java>
```

こんな感じで、画面に「Hello World!」と表示されたはずです。

地味の国から地味を広めに来たようなプログラムですが、とりあえず完全自作 Java プログラミングの完成です！

ちなみに、プログラム中の、

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

この部分を書き換えれば、どんな文字列でも表示可能です。

```
public class HelloWorld{
    static public void main(String[] args){
        System.out.println("本書名絶賛発売中");
    }
}
```

要書き換え

と書き換えて、コンパイルして実行すれば、

```
C:¥Java>java HelloWorld
本書名絶賛発売中
```

となり、本書の宣伝だって出来ちゃいます。是非このプログラムを世界中に配布してください。

2.4 Java の構文基本中の基本

2.4.1 プログラム雛形作成の基本ルール

さて、先ほど HelloWorld.java の最も基本形として、

```
public class HelloWorld{
    static public void main(String[] args){

    }
}
```

と書きました。

このように書いたプログラムが、「プログラム起動用クラス」と呼ばれるものです。このように記述したプログラムのクラス名、ここでいう HelloWorld という名前を指定することでプログラムが実行されるのです。

このようなプログラムを作成するときのポイントは、
一行目の

```
public class HelloWorld{
```

この HelloWorld と、二行目からの

```
    static public void main(String[] args){  
  
    }
```

この部分になります。

まず、一行目ですが〇〇.java ファイルを作成した場合まずは、

```
public class 〇〇{  
}
```

と書きます。これを**クラスの宣言**と呼びます。Java のプログラムはすべてクラスが基本となっていますので、まずこのように新しいクラスを宣言するところからプログラミングがスタートするのです。

基本的に、プログラムはこの {} に囲まれたところに書かれていきます。若干の例外はありますが、それについては、第 9 章でお話しします。それまでは忘れておいてください。

とにかく、最初にこのように書く癖を付けてください。

このとき、〇〇は、必ずファイル名の .java の前になります。ファイル名が HelloWorld.java ならば、HelloWorld と書きます。それ以外を書いてはいけません。それ以外のものを書いた場合、コンパイルに失敗しますから、注意しましょう。特に、途中でファイル名を変えた場合など忘れがちです。

ちゃんとクラス宣言が出来たら、次に {} の中に、

```
public class HelloWorld{  
  
    static public void main(String[] args){  
  
    }  
}
```

と書きます。中に書きますよ。外に書いては駄目です。これを**メインメソッド**と呼びます。

プログラムは、すべてメインメソッドからスタートして、上から順に処理を行っていきます。

たまに戻ったり、ゼンゼン別の場所に移動したりしますが、基本的にはメインメソッドの { からプログラムはスタートして、} まできたら終了します。

そのため、メインメソッドの外に命令を書いても基本的に実行されません。注意してく

ださい.

これが、Java プログラミングの雛形作成の基本です.

2.4.2 Java の文法のルール

Java でプログラムを書くときは、いくつか基本的なルールがあります. まずはそのルールについて説明したいと思います.

2.4.2.1 半角文字で書く

Java のプログラミングは基本的にすべて半角文字で行います. また、大文字小文字を厳格に区別しますので、注意してください.

サンプルプログラムを自分で書いてみる場合には特に注意してください.

なお、例外的に全角文字を利用可能な場所もあります. それが、文字列を書く場合です. 文字列は、基本的にすべて""(ダブルクォーテーション)に囲まれた中に書き込まれますので、それ以外の箇所では全角文字は使われないと思ってください.

2.4.2.2 命令の最後に;(セミコロン)

Java では、一つの命令が終わったことを示すために;(セミコロン)を使います. このセミコロンを付け忘れるとコンパイルできません.

ロボット君のプログラムを書いたときも、

```
robot.right();
```

のように、最後に必ず;を付けていましたよね.

忘れてしまうと、コンパイルに失敗しますのでご注意ください.

2.4.2.3 ブロック

Java でプログラミングをしていると、{}がやたらとでてきます.

このような、{から}までの間を**ブロック**と呼び、一つの塊であることを意味します.

例えば、

```
public class HelloWorld{  
    //中略  
}
```

とあれば、HelloWorld の直後の{から一番最後の}までが、HelloWorld というクラスの中身ですよ、という意味になります.

また、

```
static public void main(String[] args){
```

```
}
```

とあれば、この{から}までの間がメインメソッドですよ、という意味になります。
Java においてブロックは色々な場所でできますので、注意してみてください。

2.4.2.4 コンパイルを忘れない

Java プログラミングをしていて最初のうちにありがちなのが、プログラムを変えるだけで、コンパイルするのを忘れて、「直っていない!」と焦ることですね。

筆者も良くやりましたが、プログラムを直すだけで満足してしまって、コンパイルをしないで古いファイルで実行してしまうことが良くあります。

一度修正したら、必ずコンパイルを行うクセを付けておきましょう。

2.4.3 画面に文字を出力する

2.4.3.1 文字出力の基本

さて、先ほど画面上に Hello World!と書いたり、本書名絶賛発売中と書いたりするプログラムを作成しましたが、このときキーワードになったのが、`System.out.println` です。

これは、Java の最も基本的な命令の一つで、画面に文字を出力するために使う命令です。使い方は簡単で、

```
System.out.println(表示したい文字);
```

と書けばよいだけです。

このとき、表示したい文字は必ず"" (ダブルクォーテーション) で囲みます。例えばこんな感じ。

```
System.out.println("文字を表示します");
```

""を付け忘れると、コンパイルに失敗しますよ。

例えば、以下のようなプログラムを書いてみましょう。

```
public class StringTest {  
    public static void main(String[] args) {  
        System.out.println(ダブルクォーテーションなんて知らないよ);  
    }  
}
```

あえてダブルクォーテーションをつけずに書くという挑戦的なことをしています。しかしながら、これをコンパイルしようとする時、

```
C:\¥Java> javac StringTest.java  
StringTest.java:6: シンボルを見つけられません。  
シンボル: 変数 ダブルクォーテーションなんて知らないよ  
場所 : StringTest の クラス  
System.out.println(ダブルクォーテーションなんて知らないよ);
```

エラー 1 個

となり、怒られてしまいました。やはりダブルクォーテーションは必要です。ただし、ダブルクォーテーションを書くのは、文字列を書く場合だけです。

2.4.3.2 色々な出力

例えば数字を書く場合は、ダブルクォーテーションで囲まなくても Ok です。

```
public class StringTest {  
  
    public static void main(String[] args) {  
        System.out.println(1192);  
        System.out.println("1192");  
    }  
}
```

こんな風に、数字を""で囲んだり囲まなかったりしていますが、どちらもコンパイル可能で、実行すると、

```
1192
```

```
1192
```

と出力されます。

また、数字の場合は計算させた結果を出力することも出来ます。

```
public class StringTest {  
  
    public static void main(String[] args) {  
        System.out.println(100*120);  
    }  
}
```

この結果はちゃんと、

```
12000
```

となります。

また、数字と文字列を組み合わせることも出来ます。組み合わせるためには+記号を使います。

```
public class StringTest {
```

```
public static void main(String[] args) {  
    System.out.println(1192+"作ろう鎌倉幕府");  
}  
}
```

この出力結果は、

```
1192 作ろう鎌倉幕府
```

となります。

2.4.3.3 エスケープシーケンス

`System.out.println` を使って複数行にわたる文章を書く場合は、以下のようにすれば Ok です。

```
public class StringTest {  
    public static void main(String[] args) {  
        System.out.println("春眠暁不覚");  
        System.out.println("处处聞啼鳥");  
        System.out.println("夜来風雨声");  
        System.out.println("花落知多少");  
    }  
}
```

一つの `System.out.println` で出力すると、改行が入りますので、

```
春眠暁不覚  
处处聞啼鳥  
夜来風雨声  
花落知多少
```

となり、漢詩の表示だつてできてしまいました。

ただし、いちいち `System.out.println` を書くのが面倒な場合は、途中で改行を示す文字列を入れることができます。改行を表す文字は、`¥n` です。

```
public class StringTest {  
    public static void main(String[] args) {  
        System.out.println("途中で改行を¥n 入れることができるよ！");  
    }  
}
```

これを実行すると、

途中で改行を 入れることが出来るよ！

となります。

このような改行を表す特殊な記号をエスケープシーケンスと呼びます。

ちなみに、本当に改行を入れて、

```
public class StringTest {  
    public static void main(String[] args) {  
        System.out.println("途中で改行を  
                            入れることが出来るよ！");  
    }  
}
```

などとしなくてくださいね。

```
StringTest.java:12: 文字列リテラルが閉じられていません。  
                    System.out.println("途中で改行を  
                    ^  
StringTest.java:13: ¥65281 は不正な文字です。  
                    入れることが出来るよ！");  
                    ^  
StringTest.java:13: 文ではありません。  
                    入れることが出来るよ！");  
                    ^  
StringTest.java:13: 文字列リテラルが閉じられていません。  
                    入れることが出来るよ！");  
                    ^  
エラー 4 個
```

こんなコンパイルエラーが続出することになります。

さて、エスケープシーケンスには改行以外にも何種類かありますので、ここでは比較的良く使うものだけに限りご紹介しておきます。

エスケープシーケンス	意味
¥n	改行
¥t	タブ
¥¥	¥
¥"	"
¥'	'

例えば、通常""で囲んだ中には、"はかけませんが、¥"と書くことによって表示は"とさせることができます。こんな感じ。

```
public class StringTest {  
  
    public static void main(String[] args) {  
        System.out.println("Java では¥"¥"に囲まれた箇所を文字列として扱います");  
    }  
}
```

とすれば、

Java では""に囲まれた箇所を文字列として扱います

となります。

覚えておいて損はありません。活用してみてください。

2.4.4 コメント

プログラムを書いていると、ここで何をするのか、とかここを後で修正しなければ、といった覚書を残しておきたくることが良くあります。

そんなときに利用するのが**コメント**です。

コメントは、プログラム上に書いてはありますが、コンパイルするときにすべて無視されます。したがって、コメント部分では**Java**の文法を一切無視したことが書けることとなります。

そんなコメントは、以下のように作成します。

```
public class CommentTest {  
  
    public static void main(String[] args) {  
        //これはコメントです。  
        System.out.println("コメントは無視されます");  
    }  
}
```

このように、//と書くと、同じ行でそれ以降に書いてあるものは全てコメントとして扱われ、コンパイル時には無視されます。

ですので、こんな事をする・・・

```
public class CommentTest {  
  
    public static void main(String[] args) {  
        System.out.println("コメントは無視されます");  
        //System.out.println("これはコメント");  
    }  
}
```

```
}
```

こんな風に書くと、その実行結果は、

コメントは無視されます

となります。

ちなみに、文の途中に//を書くと、それ以降全てが無視されますので、

```
System.out.println("コメントは無視されます");//コメントに関して出力します
```

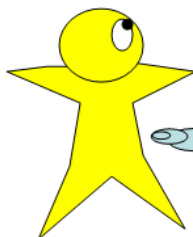
と書けば、後半部分はすべて無視されることになります。

また、//は一行だけコメントアウトしましたが、複数行にわたってコメントアウトする場合は、/* */を使います。使い方はこんな感じ。

```
public class CommentTest {  
  
    public static void main(String[] args) {  
        System.out.println("コメントは無視されます");  
        /*  
        System.out.println("コメント 1 行目");  
        System.out.println("コメント 2 行目");  
        */  
    }  
  
}
```

この、/*と*/に書いてある間のコードはすべて無視されることになります。

```
public class CommentTest {  
    public static void main(String[] args) {  
        System.out.println("コメントは無視されます");  
        /*  
        System.out.println("コメント 1 行目");  
        System.out.println("コメント 2 行目");  
        */  
    }  
}
```



コメントアウト
しているな



public class CommentTest {
 public static void main(String[] args) {
 System.out.println("コメントは無視されます");
 }
}
と書いてあるな...

図 18 コメントアウトは無いものとして扱われる

複数行にわたって一気にコメントアウトするとき、調子に乗ってコメントの場所を長くしすぎないようにしましょう。また、`/*`があったら、必ず`*/`が必要です。コメントの閉じ忘れがあると、それ以降のプログラムはすべて消えているのと同じになりますので、ご注意ください。

コラム・コメントの上手な使い方

プログラムを書いていると、コメントを使う場面はかなり多いですが、主な用途をいくつかご紹介しましょう。

1. プログラムの説明を書く場合

プログラムに関してコードを読んだだけではわかりづらいことをコメントとして残しておきます。これで、後でプログラムを読んだときに何をしているのかをすぐに思い出すことができます。

2. ひとまずいらなくなったコードを実行しないようにしておく

プログラムを書いていると、以前に書いたけど、もしかしたらいらなくなってしまう、と思うような箇所が出てくることがあります。このような箇所を消してしまうと、あとでやっぱり必要だったときに困ってしまいます。

そこで、こんなときは必要ない箇所をコメントで囲んでしまいます。こうすることによって、プログラムを動かすときには使われないけれど、やっぱり必要になったときに簡単に復活させることができます。

3. JavaDoc のため

Java には、作ったプログラムの説明書を自動的に作ってくれる **JavaDoc** という機能があります。この機能を使いこなすためには特殊なコメントをプログラム中に書いておく必要があります。

この機能はかなり便利なので、いずれ大規模なプログラムを作るときには是非活用していただきたいと思います。が、まあ、最初のうちは気にしなくてもいいと思います。JavaDoc については第 12 章（なんと、最終章！）で説明しますのでそれまでじっくりお待ちください。

2.5 本書でプログラムを作る上で

2.5.1 新しいプログラムを作る場合

本書に載っているサンプルプログラムを作成する場合は、以下のような手順を踏みます。

1. 適当な名前+.java というファイルを作成. (Ex. Sample.java)
2. テキストエディタで上記ファイルを開く
3. クラス宣言とメインメソッドを書く

```
public class Sample{  
    static public void main(String[] args){  
  
    }  
}
```

4. メインメソッド内にプログラムを記述する.
5. コンパイルする
6. 実行する

ただし、第7章クラスからは、メインメソッドがあるクラス以外のクラスも作成していくこととなります。

これについては第7章で詳しく説明します。とりあえず、第6章まではプログラムを書くときはサンプルのようにファイルを作成して、メインメソッドの中にプログラムを書くようにしてください。

ちなみに、結構多くのプログラムを作成していくこととなりますので、章ごとにフォルダを分けていくことをお勧めします。

例えば、第2章(本章)のプログラムを書く場合は、C:\Javaの下にchapter2という名前のフォルダを作成して、

```
C:\Java\chapter2\
```

にサンプルプログラムを作成した方が、あとあと管理に困らずに済むでしょう。

2.5.2 実行フォルダに注意

Javaのプログラミングをする場合、フォルダとファイル名が非常に重要になります。

まず、フォルダについてですが、基本的に実行は、classファイルが存在するフォルダでなければいけません。

例えば、C:\Java\chapter2というフォルダを作って、本章のプログラムを作成したとしてコンパイルしたとしたら、実行するときはフォルダをC:\Java\chapter2に移動しなければ行けません。

もし、C:\Javaフォルダで実行しようとする、

```
C:\Java>javac chapter2\HelloWorld.java
```

```
C:\Java>java chapter2\HelloWorld
```

```
Exception in thread "main" java.lang.NoClassDefFoundError: chapter2\HelloWorld (
wrong name: HelloWorld)
    at java.lang.ClassLoader.defineClass1(Native Method)
    at java.lang.ClassLoader.defineClass(ClassLoader.java:620)
    at
java.security.SecureClassLoader.defineClass(SecureClassLoader.java:12
4)
    at java.net.URLClassLoader.defineClass(URLClassLoader.java:260)
    at java.net.URLClassLoader.access$000(URLClassLoader.java:56)
    at java.net.URLClassLoader$1.run(URLClassLoader.java:195)
    at java.security.AccessController.doPrivileged(Native Method)
    at java.net.URLClassLoader.findClass(URLClassLoader.java:188)
    at java.lang.ClassLoader.loadClass(ClassLoader.java:306)
    at sun.misc.Launcher$AppClassLoader.loadClass(Launcher.java:276)
    at java.lang.ClassLoader.loadClass(ClassLoader.java:251)
    at java.lang.ClassLoader.loadClassInternal(ClassLoader.java:319)

C:\Java>
```

こんな感じでエラーが発生してしまいます。
実行する場合は、フォルダを移動して、

```
C:\Java>cd chapter2 ←フォルダの移動
```

```
C:\Java\chapter2>java HelloWorld
Hello World!
```

```
C:\Java\chapter2>
```

実行することが出来ます。

それでは、Java でプログラムを書く準備はできたでしょうか？いよいよ次章から、本格的な Java プログラミングの勉強に入ります。頑張りましょう。