Introduction to Java

Special Care Set for Java Begineers/ Ex 01

・Notice!
The motivation of this page is to lead regular exercises.
We prepare many useful information in this page as hints of exercises, but this page has no answer.
If you have question, please don't hesitate to ask and discuss your teacher and teaching assistants.
このページは、通常の演習問題への導きとして作られています。
多くの有用な情報をヒントとして準備しておりますが、ここに回答が隠されているとかはありません。
もし、演習に対して質問等ございましたら、遠慮なく教員・TAに質問・議論をしてください。

Example 1: Let's try to use Java Applications!

　JavaはCと同じプログラム言語です。当然ですが、上手く組めばいろんなアプリケーションが出来ます。
　Hello Worldはあなたが一番最初に書くJava Applicationですが、Javaを使ったアプリケーションはもう使っているかもしれませんね。
　ここでは、これから勉強するだろうJavaの書き方を使ったアプリケーションを、あなたが書く必要無く、ただビルド（ソースコードをコンピュータが使えるようにする）して、アプリケーションを実行するだけで、遊べるものを幾つか紹介します。
　何度かビルドしてみて、Javaで書かれたアプリケーションを楽しんでみてください。

The Java is a programming language which same as C/C++, then Java is able to construct so many applications on any platform.
Maybe, Hello World application is your first application for Java, but we prepare several example application to just use them for your study, and these application are able to use only build by Java Compiler!
Please try to use Java compiler and Java application.

Buildの方法 / How to build:

1. Buildするファイル名を確認する。Javaのソースコードは、拡張子が*.javaとなっております。
   Please check the file name which you want to use. File extention of Java source code is ***.java**

2. Buildするクラス名を確認する。（ファイルを開いて、『public static void main(...)』が書かれている一番外側の class ____ の名前を確認する。 大抵、上記のクラス名とファイル名は同じ名前にしているはずです。）
   Please check the class name. You can open the file, and search a line which written as "pubic static void main...". Then, you can find the class name "class [Class name]" which has upper line of "public static void main". Generally, class name and file name is same.

3. 次の様にコマンドを入力する
   Input command as below:
   ```
   % javac [ファイル名]
   ```
   コマンドを打つと、[クラス名].classというファイルが出来ているので、確認する。
   If you done this command, compiler will make [class name].class file.

4. 実行のために、次の様にコマンドを入力する
   To execute application, please input command such as:
   ```
   % java [クラス名]
   ```
   このとき、クラス名がHogeだった場合、java Hoge.class ではなく、 java Hoge で実行できる。
   Notice: If class name was "Hoge", this command is "java Hoge", not "java Hoge.class".

サンプルアプリケーション集 - ソースコードをダウンロードし、コンパイルしてみてください。
Sample Source Codes

- Hello World (HelloWorld.java)
- Fibonacci (Fibonacci.java)
- BMI 計算サンプル (SampleBMI.java)
- GUIでの画像表示サンプル (AWTRotationSample.java, akabee.png)

Example 2: What is this error?

Javaでは、色々なエラーメッセージを出力します。これは、ルールに則らない、不正なソースコードがどこにあるかを見つけるために非常に重要なものです（デバッグと言います）。
Javaの演習では、このデバッグを自分でこなすことが出来れば、大体出来たも同然です。
ここでは、どんなエラーメッセージが出るかを、簡単なソースコードをビルドして実行しながら、順を追ってどこにエラーが潜んでいたかを見つけるようにしましょう。

Java provides a kind of error message to teach illegal expression in your source code. This function is very important for you to find error so easily.
One of the important knowledge of Java exercise is to find illegal expression of your code and chenge correct code.
In this part, we show some basic error messages and illustrate where error code included in the source code.

Case 1: No main method

次のようなコードを書くと、コンパイルは出来ますが、実行時にエラーを起こします。
This code occurs runtime error (but source code is able to compile).

```
public class NoMain {

    public static void noMainCode(String[] args) {

       System.out.println("No Main Code");

   }
}
```

```
% javac NoMain.java
% java NoMain
エラー: メイン・メソッドがクラスNoMainで見つかりません。次のようにメイン・メソッ ドを定義してください。
public static void main(String[] args)
```

Case 2: No semi-colon

行末のセミコロン（;）を忘れると、次のようなコンパイルエラーを起こします。
If you forget to put semi-colon on the tail of a line, it occurs compile error such as:

```
public class NoSemiColon {
        public static void main(String[] args){
                System.out.println("No Semi Colon")
        }
}
```

```
% javac NoSemiColon.java
NoSemiColon.java:4: エラー: ';'がありません
```

```
System.out.println("No Semi Colon")
                                   ^
```

なお、行末のセミコロン等、問題がはっきりしている場合は、Javaコンパイル時に行番号と、どこでエラーを起こしているか、またその理由を正しく出力してくれます。
Then, the problems are clear, Java compiler indicates where that error occurs in the source code by line number and ^ mark.

Case 3: No definition of the variable

変数（int, String, float等）をちゃんと定義していないと、その変数が定義されていないと、Javaコンパイラはエラーを出力します。
If source code don't defined variable, Java compiler indicates what variable is not defined.

```
public class NoDefVar {
        public static void main(String[] args) {
                // int a;  <- 本当は必要
                a = 1 + 2;
        }
}
```

```
% javac NoDefVar.java
NoDefVar.java:4: エラー: シンボルを見つけられません
a = 1 + 2;
^
シンボル: 変数 a
場所: クラス NoDefVar
エラー1個
```

Case 4: Mistyping method name

使用したいメソッドの名前が違う場合なども、変数等と同じく定義されていないとして、Javaコンパイラはエラーを出力します。
Java compiler indicates error when the method name mistyped or unknown method used.

```
public class MistypeMethod {
        public static void main(String[] args) {
                system.out.println("Mistype 1"); // Systemの大文字が違う
                System.out.print1n("Mystype 2"); // print 1 n (数字の1になってる)
        }
}
```

```
% javac MistypeMethod.java
MistypeMethod.java:3: エラー: パッケージsystemは存在しません
system.out.println("Mistype 1"); // Systemの大文字が違う
^
MistypeMethod.java:4: エラー: シンボルを見つけられません
System.out.print1n("Mystype 2"); // print 1 n (数字の1になってる
)
^
シンボル: メソッド print1n(String)
場所: タイプPrintStreamの変数 out
エラー2個
```

Javaコンパイラは複数のエラーがあった場合、それを全て表示しようとします。エラーの数が多い場合、UNIX等のコンソール上では表示しきれない場合があります。
この場合、エラー出力を別なファイルに格納して見る等の工夫が必要になりますので注意してください。
エラー出力をログとして別なファイルに格納する場合は次のような方法があります。
Java compiler indicates all error at once. If so many errors occured, then unix console can't explain all error.
Please notice you need redirect error log of Java compiler to other file in this case.
An example of redirecting error message is written such as:

```
% javac MistypeMethod.java >& errorlog.log
% more errorlog.log
```

Case 5: Source file missing at compiling

初心者のたまにあるミスとしては、ソースファイルが無い（若しくは間違った名前でソースファイルをコンパイルしようとしている）ケースがあります。
その場合のエラーメッセージは次のようになります。

```
% javac Hoge.java
javac: ファイルが見つかりません: Hoge.java
使用方法: javac <options> <source files>
使用可能なオプションのリストについては、-helpを使用します
```

Case 6: Class file missing at executing

Case 5と似ていますが、こちらはコンパイルした後のclassファイルが無い（若しくは間違った名前で実行しようとしている）ケースです。
ソースコードの名前とpublic static void main (String[] args) の含まれるクラス名が一致しない場合について、よく起こるエラーです。

```
% java Hoge
エラー: メイン・クラスHogeが見つからなかったかロードできませんでした
```

Example 3: How to move Java Applications?

Javaはオブジェクト指向プログラミング言語ですが、各メソッドは手続き型と呼ばれるC/C++の表記法をベースにしています。Cで学習した『ポインタ』の概念は直接触れませんが、Javaはポインタを使わなくても実に多種多様な動きをします。
直接、変数やメソッドがどのような動きをするかは、Jeliotと言うJavaのアプリケーションを通して見ることが出来ます。ここでは、Jeliotの使い方を説明します。

Java is a objective oriented programming language, but the representation of methods in Java is based on C/C++.
Actually, Java has no concept of "Pointer" directly but great variety of activity.
If we want to see the activity of variables and methods of Java application directly, we can see by Jeliot tool. This part explains how to use Jeliot tool.

JeliotはJavaの動きをアニメーション化して見せるプログラムです. Javaのコードがどのような動き方をするかを視認できるので、動きのイメージの一助になります。
Jeliot is a system for animating programs in Java. It will help students understand how the Java code is executed.

Jeliotを動かすには、次のコマンドを入力してください。
To run Jeliot type:

```
% java -jar /home/course/javaone/jeliot3/jeliot.jar
```

以下のフォルダに多くのJeliotで動くサンプルコードがあります。
There are a lot of examples in the directory:
/home/course/javaone/jeliot3/examples