

.JSとの

# 楽しい距離の保ち方

JavaScript in GWT

KISSAKI.Inc  
toru\_inoue



# Who am I?

KISSAKI.Inc  
toru\_inoue



**横浜で株式会社やっています。  
スマホ向けのサービス作っています。  
WP7とか、iOSとか、HTML5とか。**

# Why GWT?

## 全略

**(どうせ他の人が散々言ってるでしょうし)**

# やりたい事

☆全プラットフォーム貫通させる為に、  
Webビューで全部終らせる

# やりたい事

☆全プラットフォーム貫通させる為に、  
Webビューで全部終らせる



そのためには..

- 既存のJS資産を効率的に使用する
  - 新しいコードを足し易くする
  - 環境差に翻弄されない

# やりたい事

- **既存のJS資産を効率的に使用する**
  - **新しいコードを足し易くする**
  - **環境差に翻弄されない**

# やりたい事

Today!

- 既存のJS資産を効率的に使用する
- 新しいコードを足し易くする
- 環境差に翻弄されない

# GWTにはJSNIってのがあってな (**Java**Script**N**ative~~なんとか~~Interface)

```
private native JavaScriptObject get () /*-{  
    alert(“このまま眠り続けて死ぬ”);  
    return @com.kissaki.client.MessengerGWTCore.(略)::mtd(Ljava/lang/String;);  
}-*/;
```

☆**GWT**内で**JS**のコードを書き、メソッドとして実行出来る



# GWTにはJSNIってのがあってな (**Java**Script**N**ative~~なんとか~~Interface)

## ヤル気の無い記法解説

修飾子(native)      宣言開始(/\*-)

```
private native JavaScriptObject get () /*-{
```

alert(“このまま眠り続けて死ぬ”);  
return @com.kissaki.client.MessengerGWTCore.(略)::mtd(Ljava/lang/String);

}-\*/;

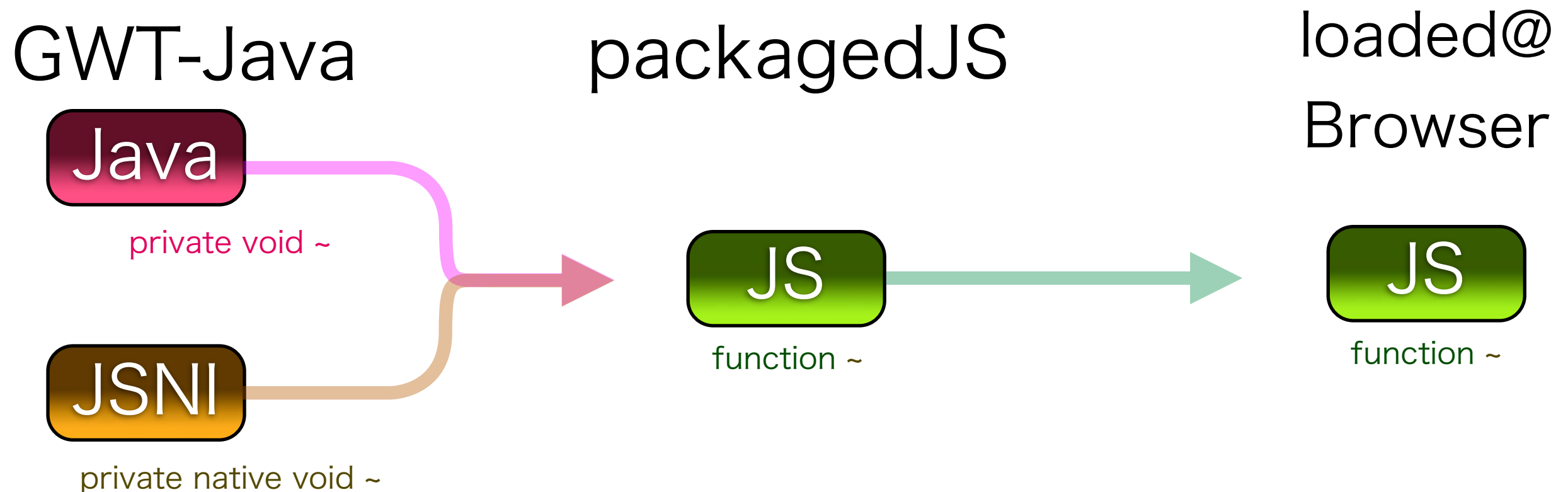
宣言終了(-\*/;)

この辺にJSでいろいろ書ける。  
ここでは、mtdファンクションを返す、  
という実装を書いてる

# GWTにはJSNIってのがあってな (JavaScriptNative~~なんか~~Interface)

☆GWT内でJSのコードを書き、メソッドとして実行出来る  
って言ったけど、具体的にどうなってるの

↓みたいな変換が起こってる。



# GWTにはJSNIってのがあってな (**Java**Script**N**ative~~なんとか~~Interface)

GWT-Java

packagedJS

loaded@  
Browser

`package com.kissaki.client.MessengerGWTCore`

Java

private void ~

JSNI

private native void ~

`com.kissaki.client.MessengerGWTCore.function ~`

JS

JS

最終的なJSのメソッドについて、

- ・ 名前空間はパッケージ単位で独自になってる (Javaからの良いトコ取り) になっている。これで、名前空間の衝突とか回避してる訳だ。

# GWTにはJSNIってのがあってな (**JavaScriptNative**~~なんとか~~Interface)

- ・ 名前空間はパッケージ単位で独自になってる(Javaからの良いトコ取り)  
**なので、**

```
private native JavaScriptObject get () /*-{  
    alert(“このまま眠り続けて死ぬ”);  
    return @com.kissaki.client.MessengerGWTCore.(略)::mtd(Ljava/lang/String;);  
}-*/;
```



**JSNIから、パッケージを指定して  
Javaのメソッドを呼び出すとかも可能**

# GWTにはJSNIってのがあってな (**Java**Script**N**ative~~なんとか~~Interface)

```
private native JavaScriptObject get () /*-{  
    alert(“このまま眠り続けて死ぬ”);  
    return @com.kissaki.client.MessengerGWTCore.(略)::mtd(Ljava/lang/String;);  
}-*/;
```

JSNIから、パッケージを指定して  
Javaのメソッドを呼び出すとかも可能

ここでは、JSNIを使ってJavaのメソッドを  
JavaScriptObject化してる  
(これはこれでド変態だが、後回し。)

# 本題

```
private native JavaScriptObject get (色々) /*-{  
    var p = $wnd.Processing(canvas, aCode);  
    p.size(320,480);  
    p.background(0);  
}-*/;
```



超注目ポイント

JSNIから、jQueryとか  
ProcessingJSとか  
Threeとか  
Box2Dとか呼べるんじゃないの

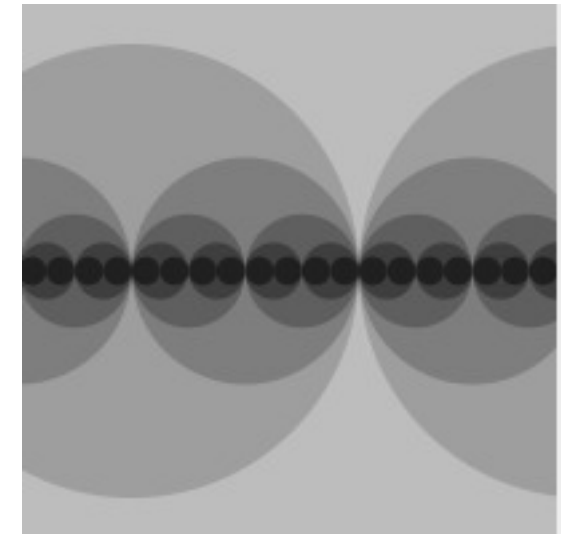
# 本題

(いろいろチョーメンドクサイけど)

呼べる.....！！

呼べるぞ.....！！

```
private native JavaScriptObject get (色々) /*-{  
    var p = $wnd.Processing(canvas, aCode);  
    p.size(200,200);  
    p.background(60);  
  
    p.fill(125);  
    p.ellipse(100,100,100,100);.....  
}-*/;
```



でもさ、

これ、

毎回JSNIで書くの？

# 本題

```
private native JavaScriptObject get (色々) /*-{  
    var p = $wnd.Processing(canvas, aCode);  
    p.size(320,480);  
    p.background(0);  
}-*/;
```

**GWTなのに、JS書いてるのと変わらん。**

**勿論デバッグも同じような感じに成る。**

**飛び込んでからどう死ぬか、という観測に時間喰う。**

**なのに、**

**毎回JSNI書くの??????**

**バカなの？死ぬの？**



# 本題と(途中)結果

```
private native JavaScriptObject get (色々) /*-{  
    var p = $wnd.Processing(canvas, aCode);  
    p.size(320,480);  
    p.background(0);  
}-*/;
```



なら、どうする

JSNI

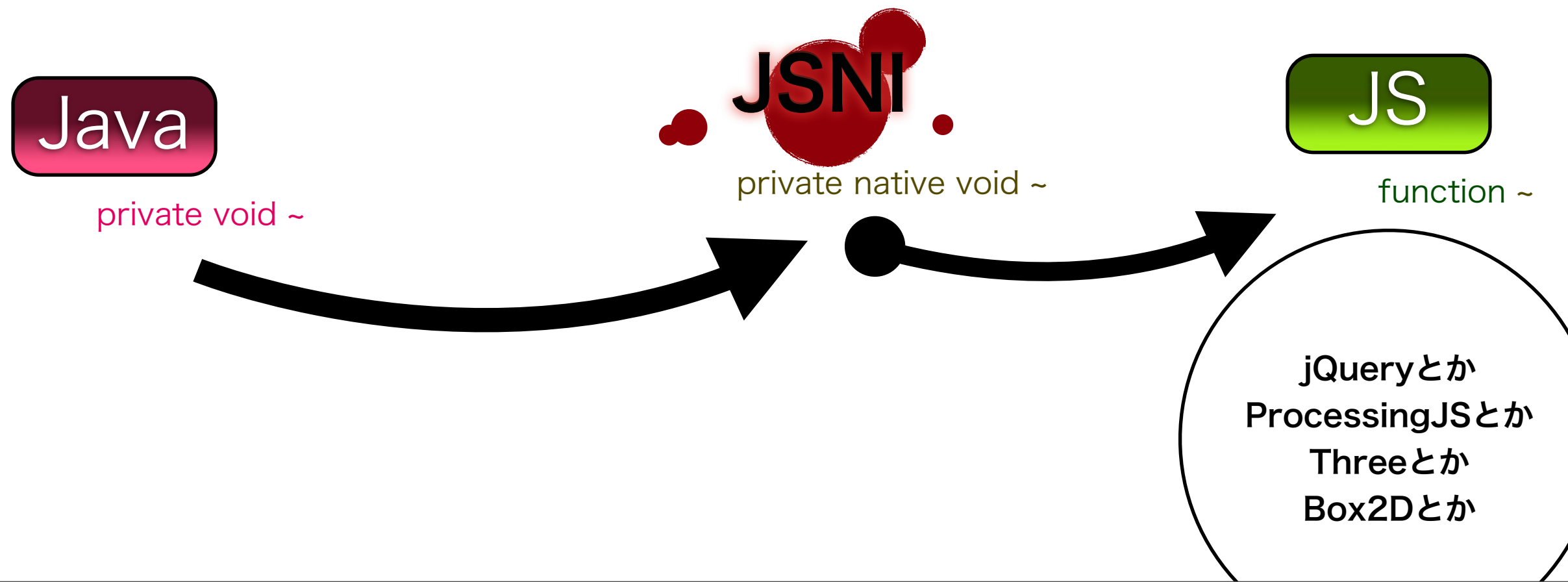
に触らず、Javaで.jsにアクセス出来ると嬉しい。



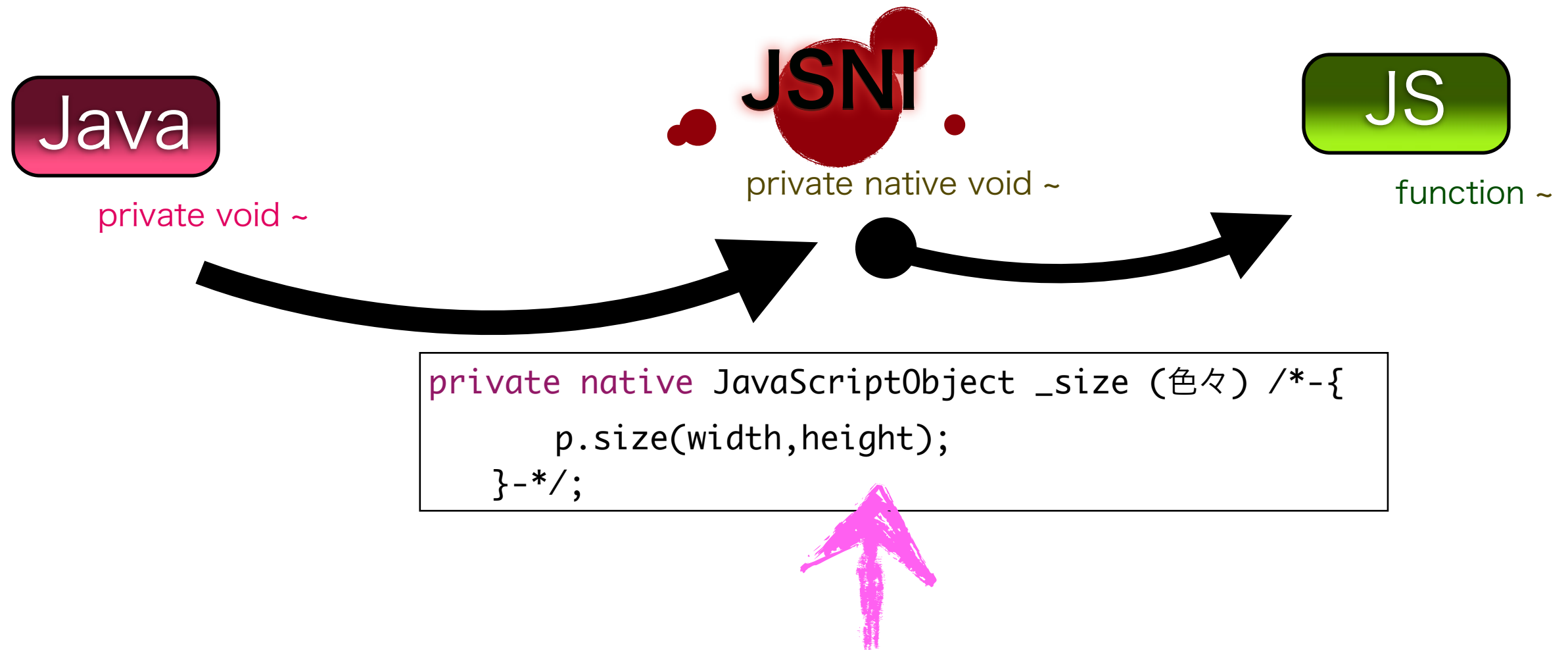
JSと1対1対応するJSNIメソッドを書き、  
それをJavaから叩けば良い

なら、どうする。

JSのFunctionと  
1対1対応するJSNIメソッドを書き、  
それをJavaから叩けば良い



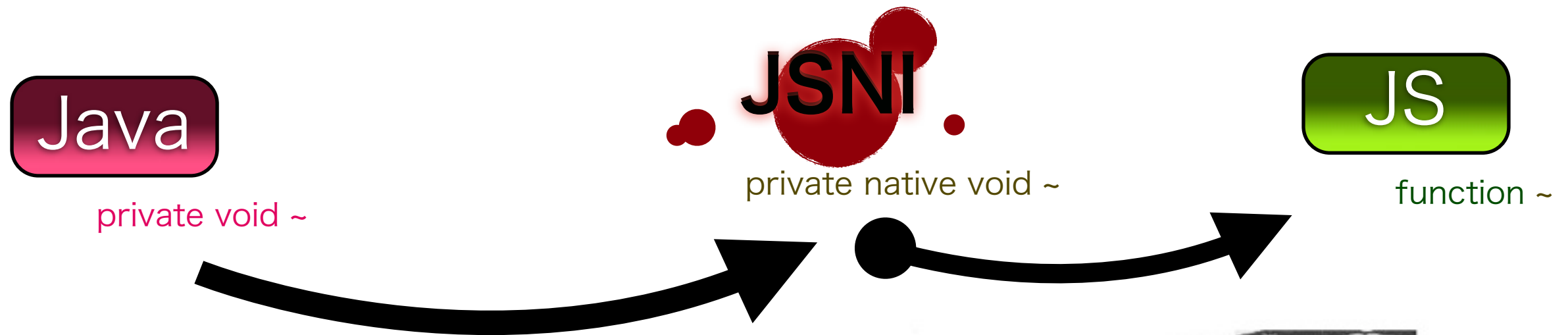
なら、どうする。



“JSと1対1対応するJSNIメソッドを書き、  
それをJavaから叩けば良い” だと？

手で書くの？

なら、どうする。

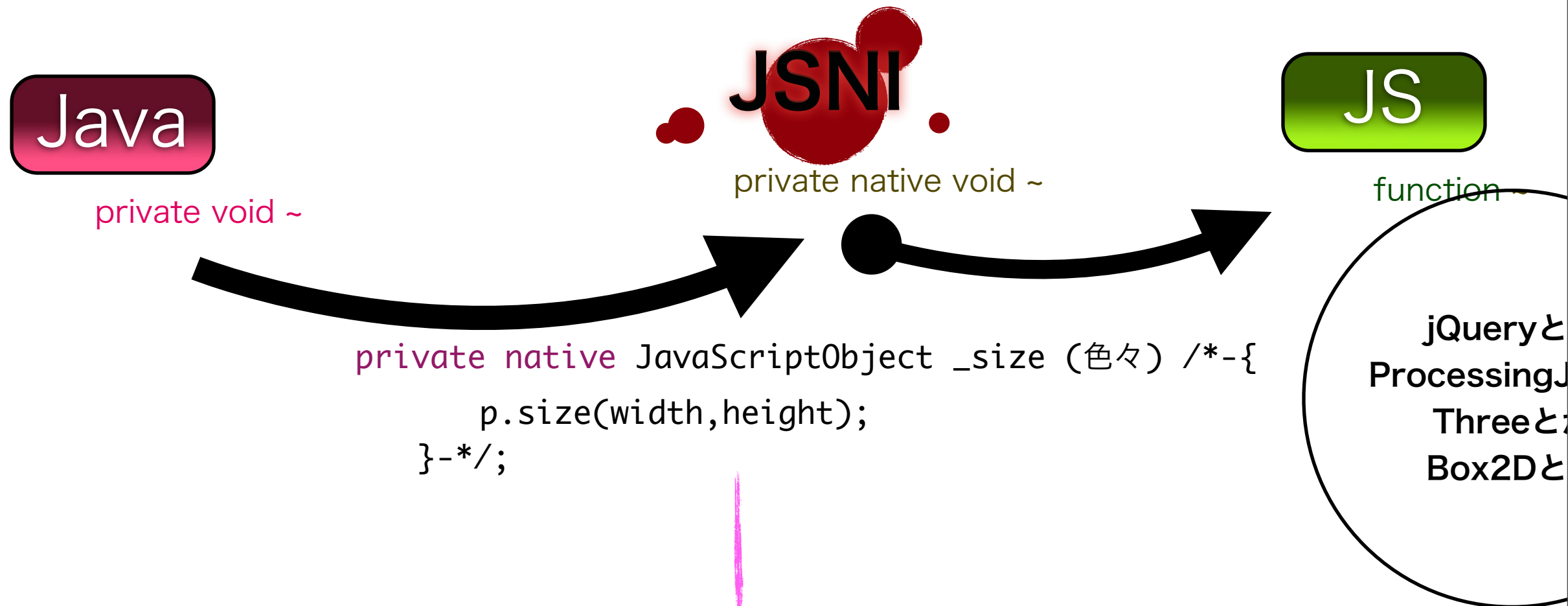


```
private native JavaScriptObject _size (色々) /*-{  
    p.size(width, height);  
}-*/;
```



ですよーw

# 本題と結果



JSを与えると、こいつを**ウイティン**する  
=JSNI入りのGWTJavaソースを吐く  
ジェネレータ作りました。

# RhinoForGWT

JSを与えると、JSNI入りのJavaを**コンパイル**する  
ジェネレータ。

元ネタ：





# RhinoForGWT



## Rhino JavaScript Compiler

### 概要

JavaScript コンパイラは、JavaScript ソースを Java クラス・ファイルへと変換します。

↑ とのことなので、  
利用させてもらいました。



# RhinoForGWT


(超手短に)手法：

Rhino(mozilla)

<https://developer.mozilla.org/ja/Rhino>

こいつが、**JSファイルを実行する為に行っている手段に着目。**

JS→構文解析→Javaのクラスファイルを直に書き出す

 ここね。

構文解析からJavaのクラスに書き出す途中に、構文に対しての型精査や数、名前等、メソッド、パラメータに関する情報がある筈、、だ。

→ありました。

というわけで、**処理を一部ブリッジして、**

JS→構文解析→**GWT用のJSNI**が記述されたファイルを吐き出す

という魔改造をしたRhinoが、**RhinoForGWT**になります。

# RhinoForGWT



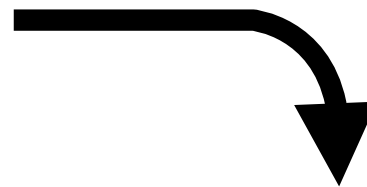
Processing.js

A port of the Processing Visualization language to JavaScript.

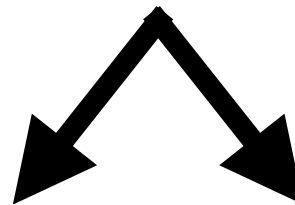


jQuery

Comprehensive DOM, Event, Animation, and Ajax JavaScript Library.



**RhinoForGWT**



ProcessingImplements.java



ProcessingInterface.java

Javaのメソッド集実装と、  
そのInterfaceを吐きます

# RhinoForGWT

## 現行バージョンの性能

対応：

- ・ メソッドのJSNIハンドラ自動作成
- ・ パラメータ名の一致
- ・ 型チェック
- ・ JSのファンクションのargumentによる素敵オーバーロード

(増パラメータのバリエーション種類分Javaメソッド作成とかやってる)  
に対応。

# RhinoForGWT

## 最新バージョンの置き場

リポジトリ：

<https://gitorious.org/rhinoforgwt>

ライセンス：MITLicense

対応：

- ・ メソッドのJSNIハンドラ自動作成
- ・ パラメータ名の一致
- ・ 型チェック
- ・ JSのファンクションのargumentによる素敵オーバーロード  
(増パラメータの種類分Javaメソッド作成)

に対応。

# RhinoForGWT

## USAGE

①リポジトリ：<https://gitorious.org/rhinoforgwt>

<https://gitorious.org/rhinoforgwt/rhinoforgwt/archive-tarball/master>  
からファイルDL

②DLしたプロジェクトをEclipseにimport

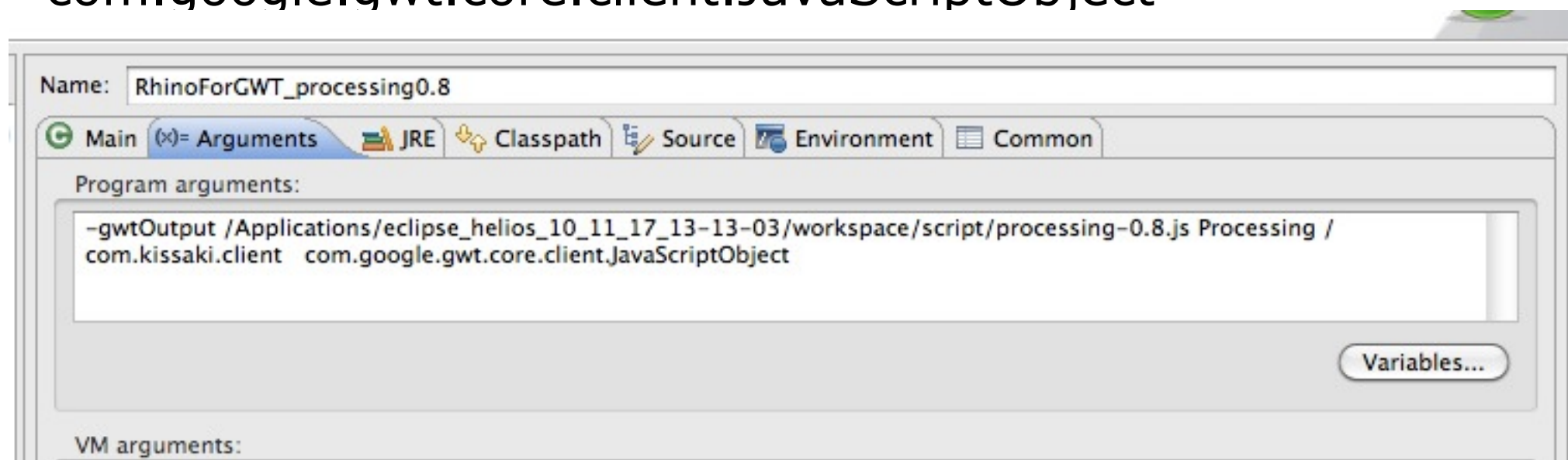
# RhinoForGWT

## USAGE

③実行コンソールに、コマンドを指定

-gwtOutput ソースJS 出力クラス名 出力場所 追加import宣言  
com.google.gwt.core.client.JavaScriptObject

Ex:-gwtOutput /Applications/eclipse\_helios\_10\_11\_17\_13-13-03/  
workspace/script/processing-0.8.js Processing / com.kissaki.client  
com.google.gwt.core.client.JavaScriptObject



# RhinoForGWT

## USAGE

④実行！ (jarでも実行出来ると思うよ！ 俺はいやだけど。)

出来上がったファイルがコチラ



ProcessingImplements.java



ProcessingInterface.java

```
ProcessingImplements.java
ProcessingImplements.java:22  class ProcessingImplements
package com.kissaki.client;

import com.google.gwt.core.client.JavaScriptObject;

public class ProcessingImplements implements ProcessingInterface {

    private JavaScriptObject ProcessingJSObject = null; //JavaScriptObject of Processing

    private JavaScriptObject getInstanceOfJSObject(){
        return ProcessingJSObject;
    }

    private native JavaScriptObject setupProcessingJSObject (String aElement, String aCode) /*- {
        return $wnd.Processing(aElement, aCode);
    }-*/;

    /*コンストラクタ*/
    public ProcessingImplements(String aElement, String aCode){
        ProcessingJSObject = setupProcessingJSObject(aElement, aCode);
    }

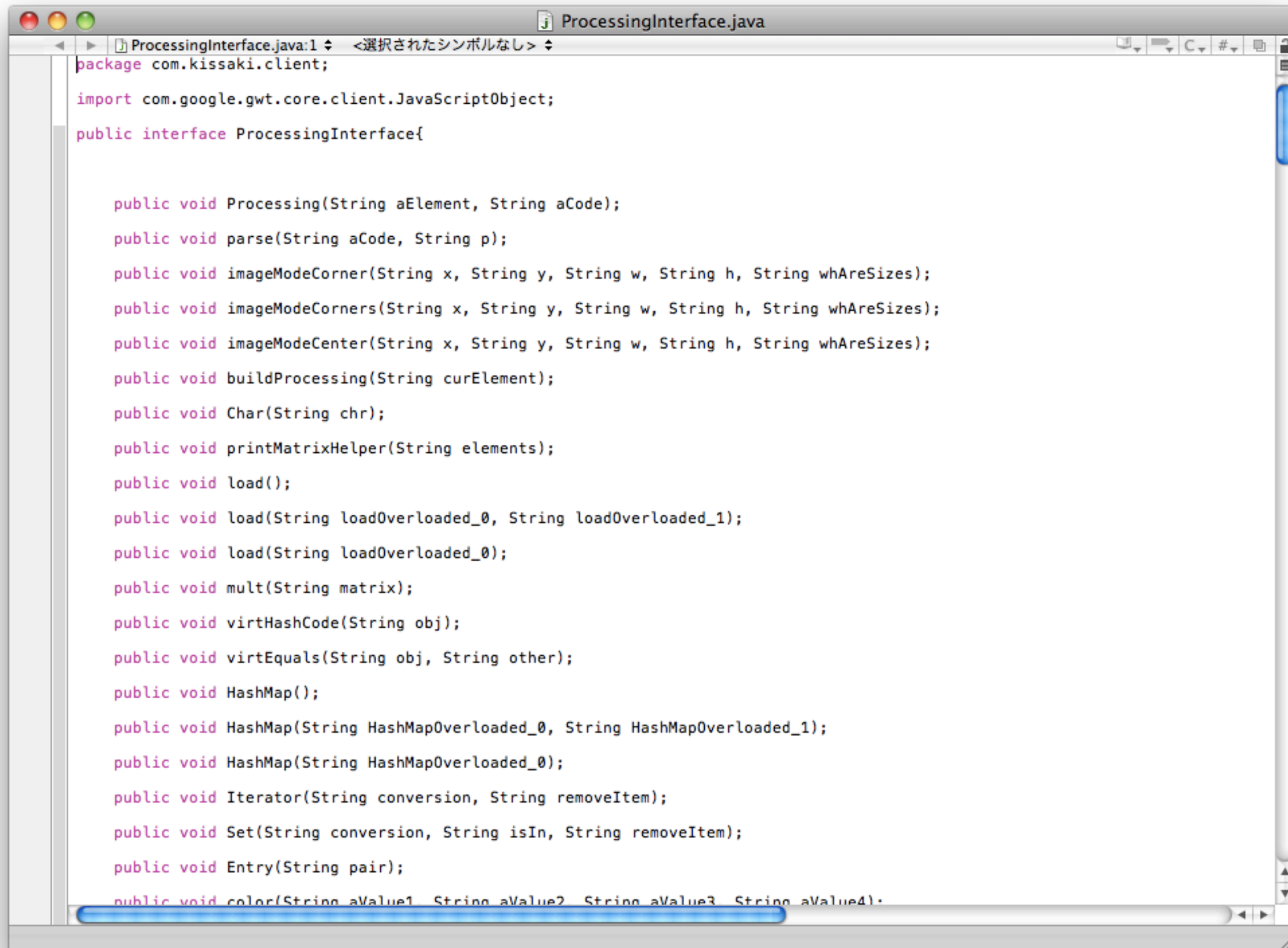
    public void Processing(String aElement, String aCode){
        _Processing(aElement, aCode);
    }
    private native String _Processing(String aElement, String aCode)/*- {
        this.@com.kissaki.client.ProcessingImplements::getInstanceOfJSObject().Processing(aElement, aCode);
    }-*/;

    public void parse(String aCode, String p){
        _parse(aCode, p);
    }
    private native String _parse(String aCode, String p)/*- {
        this.@com.kissaki.client.ProcessingImplements::getInstanceOfJSObject().parse(aCode, p);
    }-*/;

    public void imageModeCorner(String x, String y, String w, String h, String whAreSizes){
        _imageModeCorner(x, y, w, h, whAreSizes);
    }
    private native String _imageModeCorner(String x, String y, String w, String h, String whAreSizes)/*- {
        this.@com.kissaki.client.ProcessingImplements::getInstanceOfJSObject().imageModeCorner(x, y, w, h, whAreSizes);
    }-*/;

    public void imageModeCorners(String x, String y, String w, String h, String whAreSizes){
        _imageModeCorners(x, y, w, h, whAreSizes);
    }
}
```





```
ProcessingInterface.java
ProcessingInterface.java:1 <選択されたシンボルなし>
package com.kissaki.client;

import com.google.gwt.core.client.JavaScriptObject;

public interface ProcessingInterface{

    public void Processing(String aElement, String aCode);
    public void parse(String aCode, String p);
    public void imageModeCorner(String x, String y, String w, String h, String whAreSizes);
    public void imageModeCorners(String x, String y, String w, String h, String whAreSizes);
    public void imageModeCenter(String x, String y, String w, String h, String whAreSizes);
    public void buildProcessing(String curElement);
    public void Char(String chr);
    public void printMatrixHelper(String elements);
    public void load();
    public void load(String loadOverloaded_0, String loadOverloaded_1);
    public void load(String loadOverloaded_0);
    public void mult(String matrix);
    public void virtHashCode(String obj);
    public void virtEquals(String obj, String other);
    public void HashMap();
    public void HashMap(String HashMapOverloaded_0, String HashMapOverloaded_1);
    public void HashMap(String HashMapOverloaded_0);
    public void Iterator(String conversion, String removeItem);
    public void Set(String conversion, String isIn, String removeItem);
    public void Entry(String pair);
    public void color(String aValue1, String aValue2, String aValue3, String aValue4);
```

これ手で書くと



ってなるよねー。

ProcessingImplements.java:

```
public void lightFalloff(String constant, String linear, String quadratic){
    _lightFalloff(constant, linear, quadratic);
}

private native String _lightFalloff(String constant, String linear, String quadratic)/*-
    this.@com.kissaki.client.ProcessingImplements::getInstanceOfJSObject().lightFalloff(constant, linear, quadratic);
*/-;

public void lightSpecular(String r, String g, String b){
    _lightSpecular(r, g, b);
}

private native String _lightSpecular(String r, String g, String b)/*-
    this.@com.kissaki.client.ProcessingImplements::getInstanceOfJSObject().lightSpecular(r, g, b);
*/-;

public void spotLight(String r, String g, String b, String x, String y, String z, String nx, String ny, String nz, String angle, String concentration){
    _spotLight(r, g, b, x, y, z, nx, ny, nz, angle, concentration);
}

private native String _spotLight(String r, String g, String b, String x, String y, String z, String nx, String ny, String nz, String angle, String concentration)/*-
    this.@com.kissaki.client.ProcessingImplements::getInstanceOfJSObject().spotLight(r, g, b, x, y, z, nx, ny, nz, angle, concentration);
*/-;

public void camera(String eyeX, String eyeY, String eyeZ, String centerX, String centerY, String centerZ, String upX, String upY, String upZ){
    _camera(eyeX, eyeY, eyeZ, centerX, centerY, centerZ, upX, upY, upZ);
}

private native String _camera(String eyeX, String eyeY, String eyeZ, String centerX, String centerY, String centerZ, String upX, String upY, String upZ)/*-
    this.@com.kissaki.client.ProcessingImplements::getInstanceOfJSObject().camera(eyeX, eyeY, eyeZ, centerX, centerY, centerZ, upX, upY, upZ);
*/-;

public void camera(String cameraOverloaded_0){
    _camera(cameraOverloaded_0);
}

private native String _camera(String cameraOverloaded_0)/*-
    this.@com.kissaki.client.ProcessingImplements::getInstanceOfJSObject().camera(cameraOverloaded_0);
*/-;

public void perspective(String fov, String aspect, String near, String far){
    _perspective(fov, aspect, near, far);
}

private native String _perspective(String fov, String aspect, String near, String far)/*-
    this.@com.kissaki.client.ProcessingImplements::getInstanceOfJSObject().perspective(fov, aspect, near, far);
*/-;
```

# で、(1)

もちろんあるぞ！！ 現在の問題点。 なんとって、1年以上前に書いたやつだからね！！

- ・パラメータの型チェックが甘い

数字か、JavaScriptObjectか、Stringになる。

Eclipse使って型のリファクタリングを行うことがめっちゃ多い。

数値、doubleとint、とかが判別できない。これは限界ある。

オブジェクトは大体JSObjectかStringになる。JSObjectは扱いが面倒いのでStringに寄るようにしてある。

→どの型を優先するか、設定ファイルで設定出来る。だが面倒い。

- ・戻り値がない

JSNIレベルでは戻り値を返しているんだが、returnを書いてない。これは、すぐに改善出来る部分。

# で、(2)

- ・インターフェースが最悪

コマンドラインが基礎なので、正直俺も面倒い。

なので、せめてEclipseプロジェクトにしてるのだが、なんというか、やはり面倒い。

簡単なAppleScriptとかと一緒に配布しようかなって。

## ここで俺に電流走る



☆サーバで走らせればみんな幸せになるとおもう。

→**GAE**とかで走らせられるかもね！！ よく考えたら！！

JSを放り込むと、GWT用のJavaとかを返してくれるサイト。だれか手伝って。

ぶっちゃけRhino はGAEで動くとの事なので、インターフェース作れば終わりくさい。

# それはさておいて、最終目標

- ・ 真面目に考える最終目標：

最終出力物として、**JS**を自動的に**.jar**にしてくれるやつを作りたい。

外部**js**ファイルや生成された**java**を組み込むというのが面倒いので。

- ・ **jar**ファイルにしておけば、**js**ファイルをハンドルせずにすむ。
- ・ **js + java** → **.jar**ファイル になる。
- ・ **jar**ライブラリとしてビルドに巻き込めれば、名前空間の心配も消える。

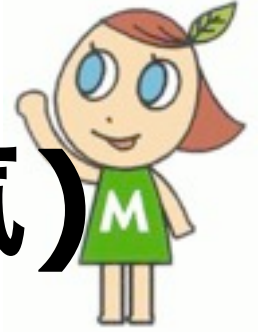
これは素敵。

了。

# ハンズオン部分



# やってみます！(CV ミドリ電気)



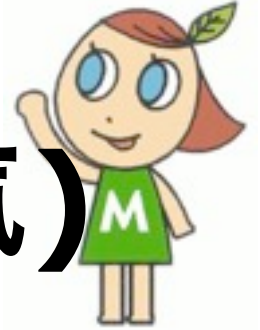
①

```
/**
 * This is the entry point method.
 */
public void onModuleLoad() {
}
```

プロジェクト適当に作って、  
onModuleLoadメソッドを  
空になるまで親のカタキのように痛めつけます



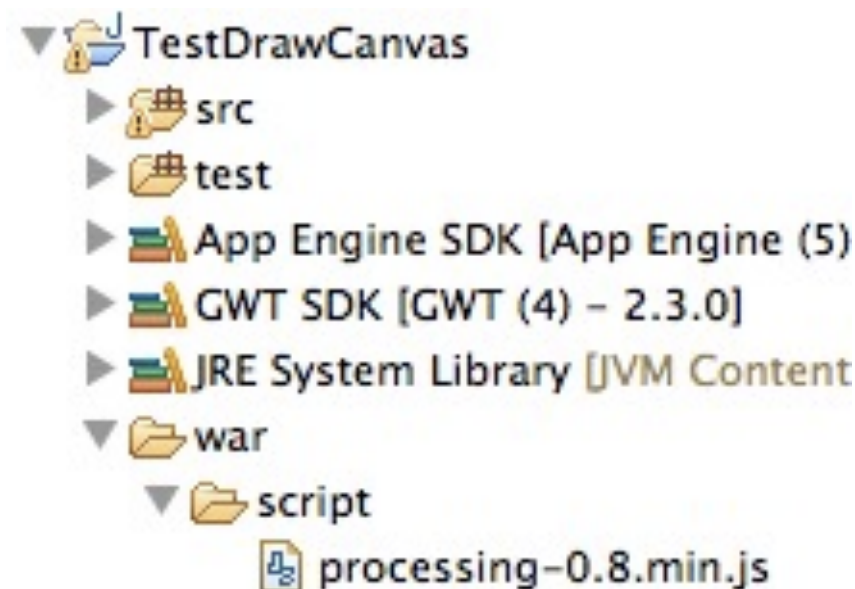
# やってみます！(CV ミドリ電気)



スクリプトを読み込むように、設定ファイルに記述。

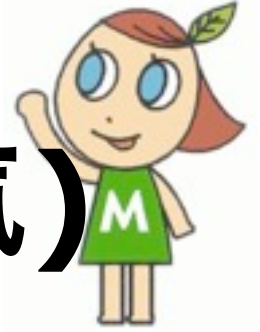
②

読み込む用のスクリプトを、warフォルダ内に置いときます  
ここでは、scriptというフォルダを作ってその中に(ry



<http://processingjs.org/content/download/processing-js-0.8/processing-0.8.min.js>

# やってみます！(CV ミドリ電気)



スクリプトを読み込むように、設定ファイルに記述。

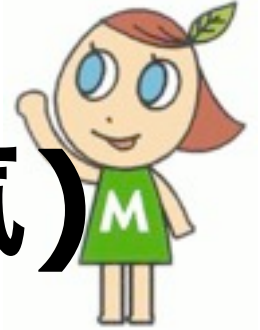
③

プロジェクト名.gwt.xml

moduleの項に、下記を追加

```
<script src="/script/processing-0.8.min.js"></script>
```

# やってみます！(CV ミドリ電気)



④

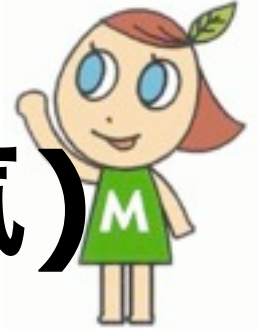
```
Canvas canv = Canvas.createIfSupported();  
canv.setWidth(200 + "px");  
canv.setHeight(200 + "px");  
canv.setCoordinateSpaceWidth(200);  
canv.setCoordinateSpaceHeight(200);
```

```
RootPanel.get().add(canv); //画面へのキャンバス要素のセット
```

```
CanvasElement canvE = canv.getCanvasElement();  
canvE.setWidth(200);  
canvE.setHeight(200);
```

onModuleLoadメソッドにキャンバスのソースを書き込み

# やってみます！(CV ミドリ電気)

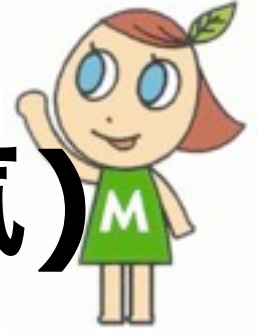


⑤

```
ProcessingImplements p =  
    new ProcessingImplements(canvasE, "");  
  
p.size("200", "200", "");  
p.ellipse("100", "100", "40", "40");
```

ProcessingJSのコード書き込み

# やってみます！（CV ミドリ電気）



⑥

```
ProcessingImplements p =  
    new ProcessingImplements(canvE, "");  
  
p.size("200", "200", "");  
p.ellipse("100", "100", "40", "40");
```

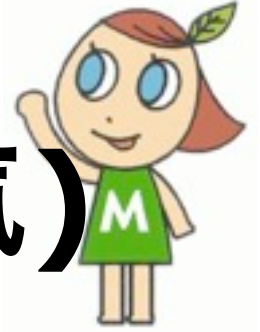
自動変換でアレな所を、、手で直す、、、、！！

ProcessingImplementsメソッドの第一引数の型を、  
String → JavaScriptObject に変える。

Eclipseの自動補完にやらせると、JSNI内の引数の名前が変わらずにエラーが出る  
（まあ、、 気づけるレベルだけど、、）

このへんは、自動変換の精度を上げて行けば、何とかなるんだが、、。

# やってみます！（CV ミドリ電気）



↓ 起動でこんなカンジ。

