内容

手軽に Java（JShell）

はじめに

1999 年 12 月の J2EE/JDK1.2 発表後 Java/Web ブームが一気に広がりましたが、5 年後の 2005 年には既に LAMP[**L**inux **A**pache **M**ySQL (**P**〉 **P**erl or **P**HP or **P**ython ...Ruby)]が J2EE の代替え手段になるという記事が米国のコンピュータ雑誌に出ています。

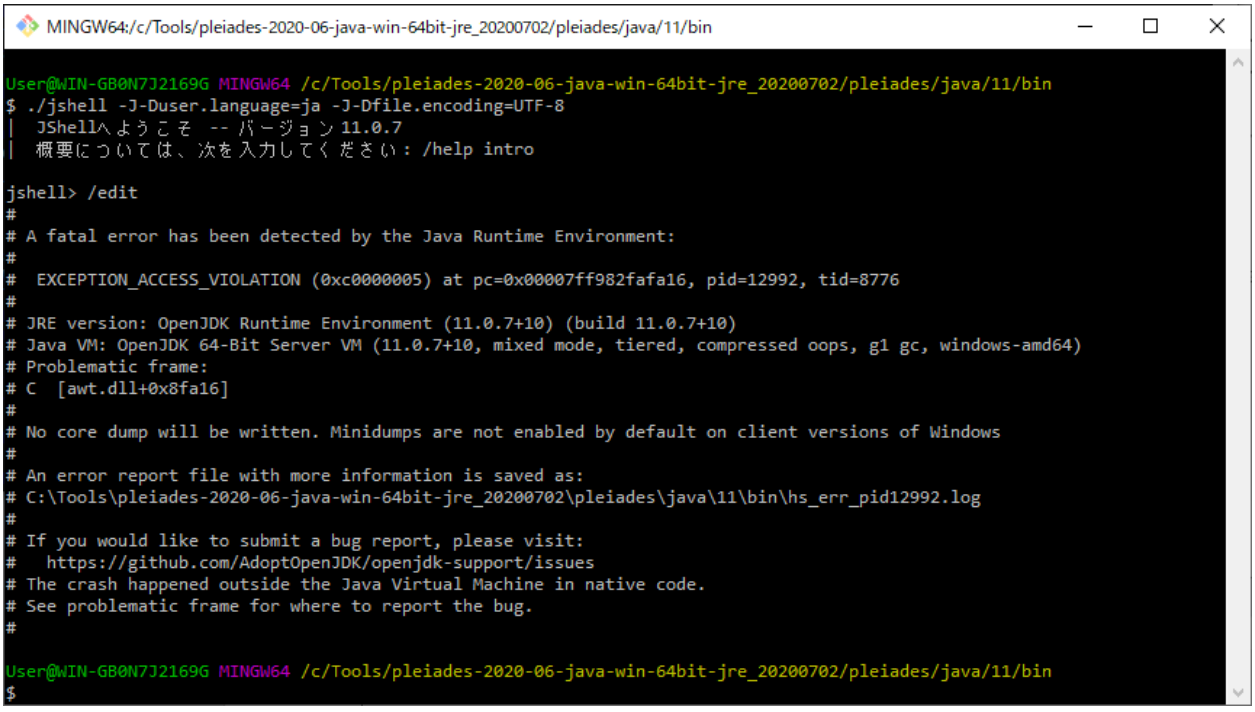
https://www.computerworld.com/article/2556671/the-lamp-alternative-to-j2ee-or--net.html

Perl、PHP、Python 及び Ryby は全てスクリプト言語のためランタイムをダウンロードするだけで始められるのに比べ、Java は Web アプリとして動作可能な環境（Eclipse 等の開発環境、App サーバー、HTML/CSS/JavaScript、コンパイルが通る Java ソース）が必要になるという敷居の高さが敬遠の原因になっていたのではないでしょうか...。Java9 からは JShell というスニペット実行ツールが添付され、手軽にコードのお試しができるようになったので使い方を説明します。

1. Windows で実行する場合

Windows で JShell(Java)を実行する場合、画面表示(awt.dll)やフォントで問題が発生する場合があります。“EXCEPTION_ACCESS_VIOLATION”と表示される以下のようなエラーが発生したり、フリーズするような場合は、最新の JDK をインストールし直してください。

＜エラー発生画面＞

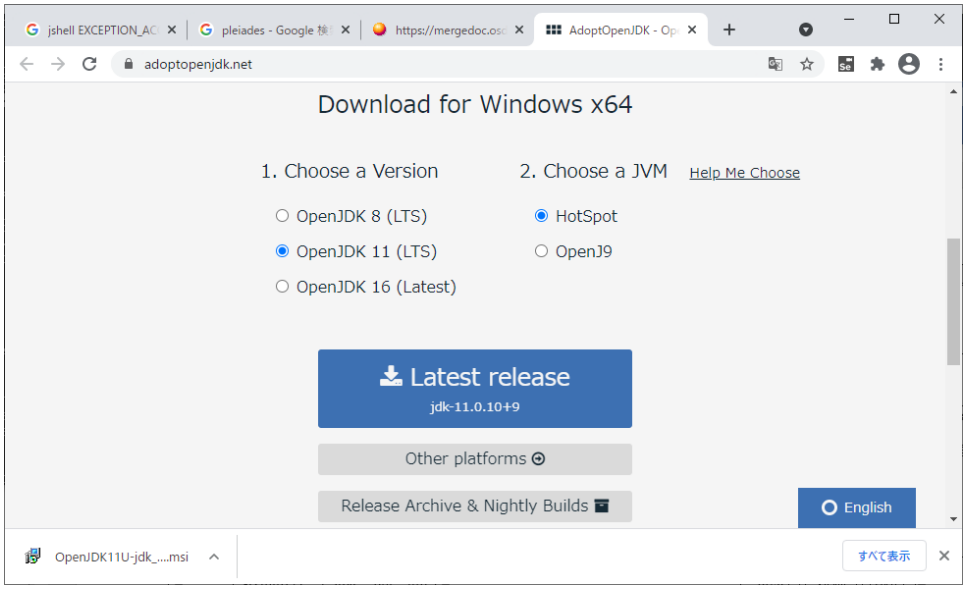


（1）ダウンロード

JDK 9 以降の安定板をダウンロードします。

AdoptOpenJDK サイト：https://adoptopenjdk.net/releases.html

今回は openjdk version "11.0.10" 2021-01-19 で動作確認を行いました。

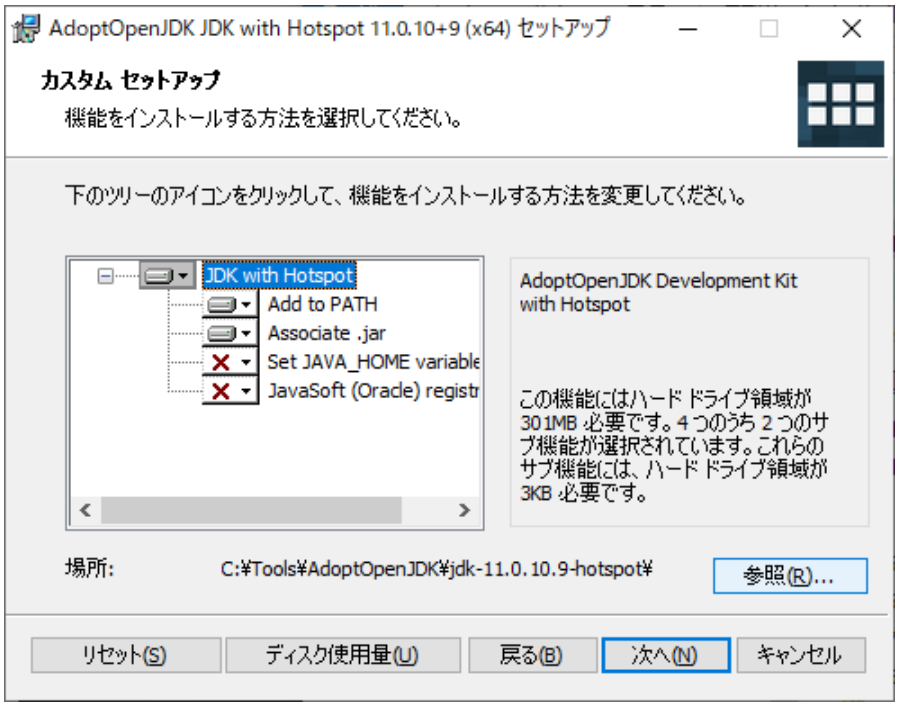


（2）インストール

ダウンロードしたインストーラを起動し、インストール先を指定します。

［参照］ボタンを押下し、インストール先フォルダを指定

※フォルダのパスに空白や日本語が含まれないようにして下さい。他に制約はありません。



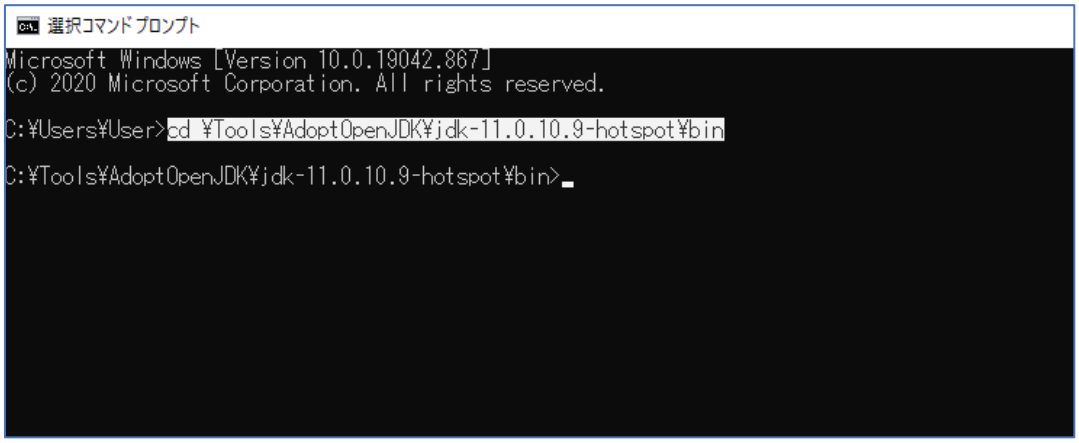
悪い例：C:¥Program Files¥

2. JShell の起動

コマンドプロンプトまたはターミナル[1]を立ち上げ、以下の手順で起動します。

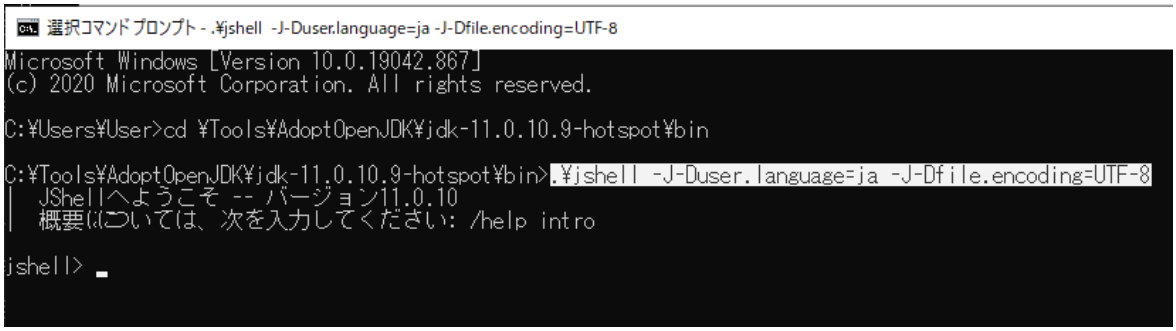
① jshell.exe が入っているディレクトリに移動する

cd *¥Tools¥AdoptOpenJDK¥jdk-11.0.10.9-hotspot¥bin*



② カレントディレクトリから JShell を起動

.¥jshell -J-Duser.language=ja -J-Dfile.encoding=UTF-8　※先頭の[.¥]はターミナルの場合［./］



＜指定オプション＞ openjdk version "11.0.10"の場合、以下オプション省略可

-J-Duser.language=ja　　：日本語

-J-Dfile.encoding=UTF-8　：文字コードは utf-8

※ターミナルの場合も、指定オプションは同一です

…Windows の場合、locale を ja_JP.UTF-8 にしただけでは正常に表示されない場合があります

※※JShell の使い方の詳細は、以下 URL より参照してください。

https://docs.oracle.com/search/?q=jshell

[1] openjdk version "11.0.10"の JShell は Windows10 のコマンドプロンプトから正常に動作しますが、バージョンによっては、Node.Js 等のツールをと同時にインストールされる「Git Bash」や Msys といった Linux をシミュレートしたターミナルでないとキー入力等が正常に動作しない場合があります。

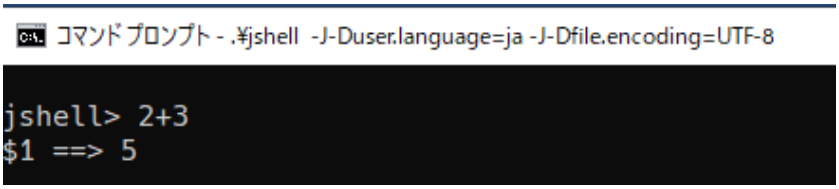
3. JShell プロンプト

JShell の実行中はプロンプト“jshell>”が出ています。ここに JShell に対するコマンドか Java のコードをタイプして Enter を押下すると即実行されます。

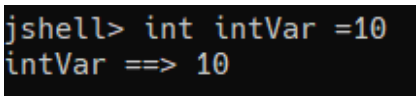
（1）Java コード

① 式

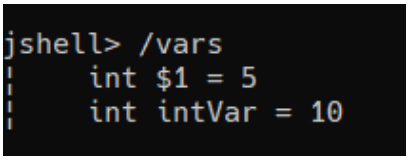
算術式は計算されて、自動的に作成された変数（変数名は$n）に割り当てられます。



② 名前を指定したの変数の割り当て

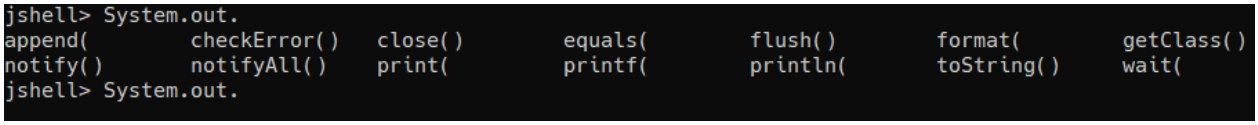


※作成済の変数は、”/vars”コマンドで確認することができます。

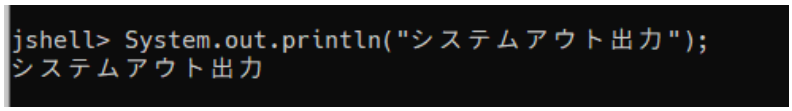


③ ステートメント（補完と実行）

途中まで入力した後に Tab キーを押下すると、その後に続く候補が表示されます。

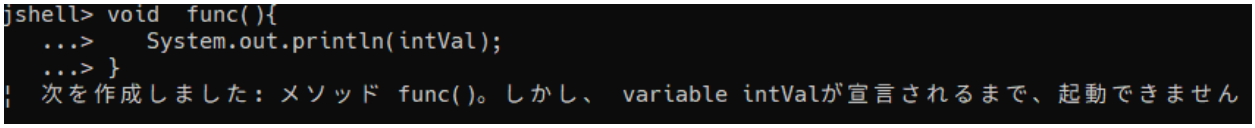


最後まで[2]入力して Enter を押下すると、直ぐに実行され結果が表示されます。



④ メソッド

jshell>プロンプトにはメソッドを書くことができます。



※宣言されていない名前（intVal）が参照されると、“前方参照”（後で宣言されるはずの名前）としてメソッドが作成されます。

---

[2] 最後のセミコロン”;”は単独行の場合は JShell が補完して実行してくれます。但し、メソッド内では”;”が必須です。

（2）コマンド

JShell に対するコマンドは”/”で始まります。

① /list

コマンド時点までに入力した Java コードが表示されます。

左側の数字が id で、*/id* で再実行が行えます。

```
jshell> /list

   1 : 2+3
   2 : int intVar =10;
   4 : System.out.println("システムアウト")
   5 : System.out.println("システムアウト出力");
  10 : void  func(){
          System.out.println(intVal);
       }
```

② /vars

宣言されている変数が表示されます。この変数はメソッドの中から参照できます。

```
jshell> /vars
|    int $1 = 5
|    int intVar = 10
```

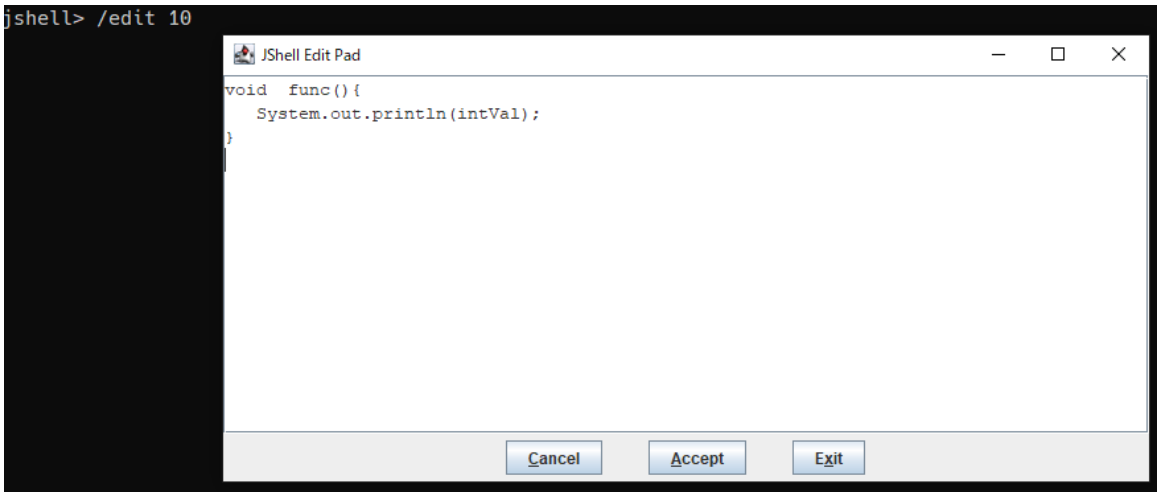
③ /methods

宣言されているメソッドが表示されます。

```
jshell> /methods
|    void func()
|      。しかし、 variable intValが宣言されるまで、起動できません
```

③ /edit *id*

id の行を編集するウィンドが開きます。edit の使い方は後で...



④ /reset

定義済の変数等を消して JShell の状態を初期化します。class-path の再設定もできます。

⑤ /help

JShell の使い方や、コマンドの説明が表示されます。

⑥ /exit

JShell を終了します。

4. JShell の使い方

JShell のコマンドプロンプトは/list コマンドの結果で分かるように、入力したコードの断片を入力順の履歴として持っているだけで処理順序という観点は保存できません。

（1）ロジック／アルゴリズムの確認

「処理」はメソッドまたはクラスとして作成し、JShell プロンプトから呼び出します。

（2）メソッド、クラスの作成と修正

他で作成した複数行のメソッド全体をコピーし、プロンプトにペーストすると書式が正しければメソッドとして認識されます。この際、メソッド全体で一つの id が振られます。

（注意）ソース中にタブが入っていると JShell が補完しようとしてエラーになる場合があります。

```
jshell> void forTestMethod() {
   ...>     String[] st = {"abc","def","ghi"};
   ...>         for ( String elm : st ) {
   ...>             System.out.println(elm);
   ...>         }
   ...> }
|  次を作成しました: メソッド forTestMethod()
```

再度、同一シグネチャのメソッドをペーストすると更新されます。

```
jshell> void forTestMethod() {
   ...>     String[] st = {"abc","def","ghi"};
   ...>     for ( String elm : st ) {
   ...>         System.out.println(elm);
   ...>     }
   ...> }
|  次を変更しました: メソッド forTestMethod()
```

引数を追加してシグネチャを変えると、別メソッドとして作成されます。

```
jshell> void forTestMethod(String a) {
   ...>     String[] st = {"abc","def","ghi"};
   ...>     for ( String elm : st ) {
   ...>         System.out.println(elm);
   ...>     }
   ...> }
|  次を作成しました: メソッド forTestMethod(String)
```

（3）メソッドの呼び出し

プロンプトでメソッド呼び出し文を打つと実行されます。

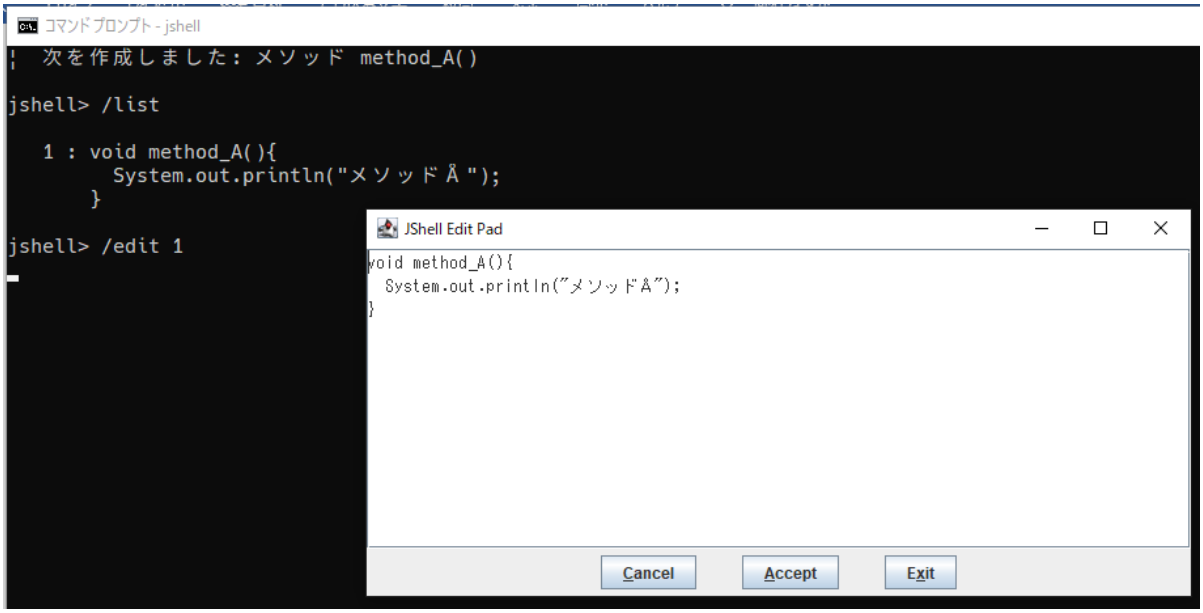
※メソッド名の先頭一文字を打鍵したあと Tab キーを押すと、候補の一覧が表示されます。

```
jshell> forTestMethod()
abc
def
ghi
```

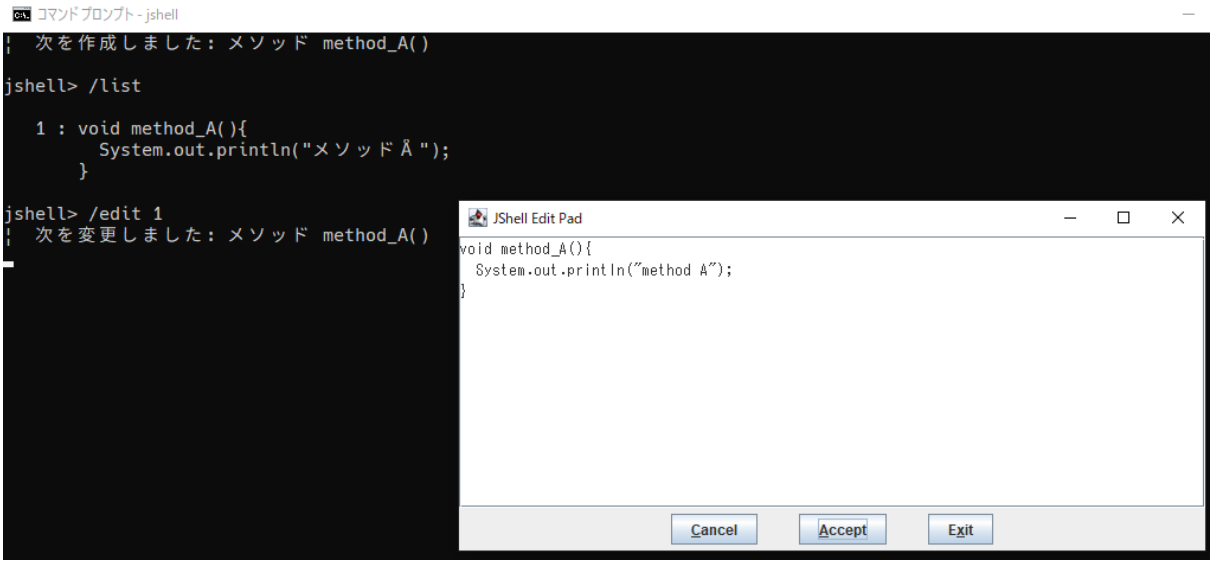

（4）/edit コマンドによる編集

“/edit”コマンドのパラメータに id か変数名やメソッドを指定すると、編集画面を使って登録済の内容を変更することができます。

① /edit *id* で制御が“JShell Edit Pad”という編集画面に移ります



② 内容を編集後、編集画面下部の［Accept］ボタンを押下すると変更が反映されます



③ 最後に編集画面下部の［Exit］ボタンを押下して JShell プロンプトに制御を戻します

※/edit コマンドには以下の制約があります

・jdk11 では一度に編集できるのは、一つの id に関してだけです
（jdk12 以降は複数 id の編集ができるようです https://docs.oracle.com/search/?q=jshell）

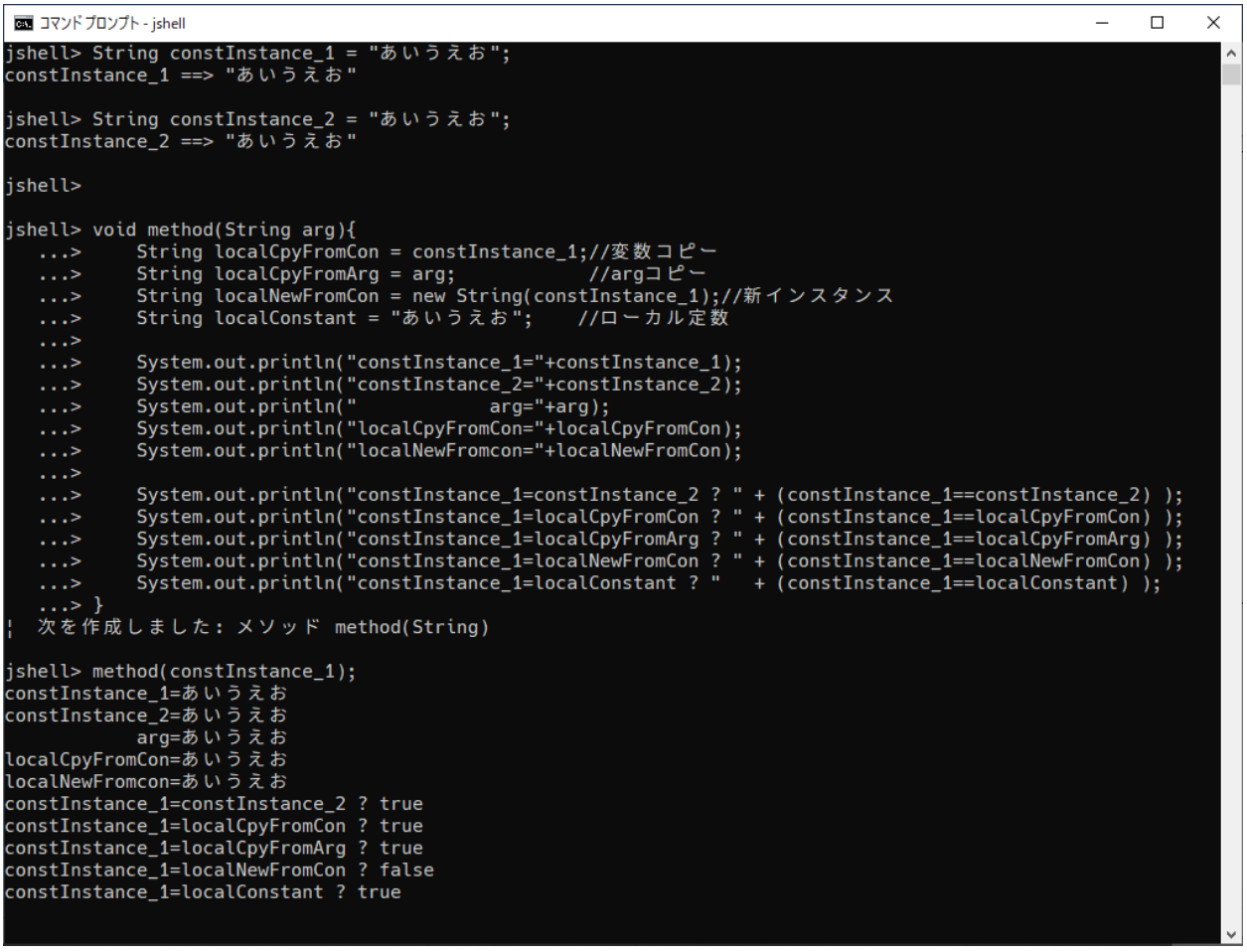
・Tab キーでの補完はできません

5. javac で作るクラスとの違い

・修飾子 public、protected、private、static、および final は使えません
・同期、ネイティブ、抽象、およびデフォルトのトップレベル・メソッドは許可されません

※JShell プロンプトで作成した変数／メソッドは、同列のメソッドから参照でき、プロンプト全体が一つのクラスで、入力更新の都度コンパイルがかかっているようです

＜String クラス／コンスタントプールの実験＞



＜上記で確認できた点＞

・メソッドの中と外で定義した変数が、文字列が同一であれば同一インスタンスを参照している
　⇒ コンパイラーによってコンスタントプールが使われている
・new 演算子で作成した参照は、同一文字列であっても別のインスタンスになっている
・インスタンスを参照している変数の代入は、メソッドの内外／引数経由で正常に行われている

※修飾子等の制限を除き内部クラス、メソッドの動作は javac でコンパイルしたクラスと同一

6. JShell 以外のツール

JShell 以外にも完全なクラス定義になる前の Java ソースを動作させることができるツールがあります。

（1）BeanShell

HP には以下のように説明されています。

『Java で記述された、オブジェクトスクリプト言語機能を備えた、小さくて無料の埋め込み可能な Java ソースインタープリターです。』

以下のプロジェクトで使用されているそうです。

Apache Ant、Apache Camel、Apache Maven、Apache OpenOffice、Apache Taverna、Apache jMeter CA DevTest、Cisco Prime Network、ImageJ、JDE for Emacs、Joget、LibreOffice＜以下省略＞

サイト：https://github.com/beanshell/beanshell

※Java の構文をそのまま解釈できますが、開発の方向は緩い型付けの簡易構文でスクリプト化を進める方針のようです。Java 初級者が学習として使うには注意が必要。

（2）Eclipse のスクラップブック

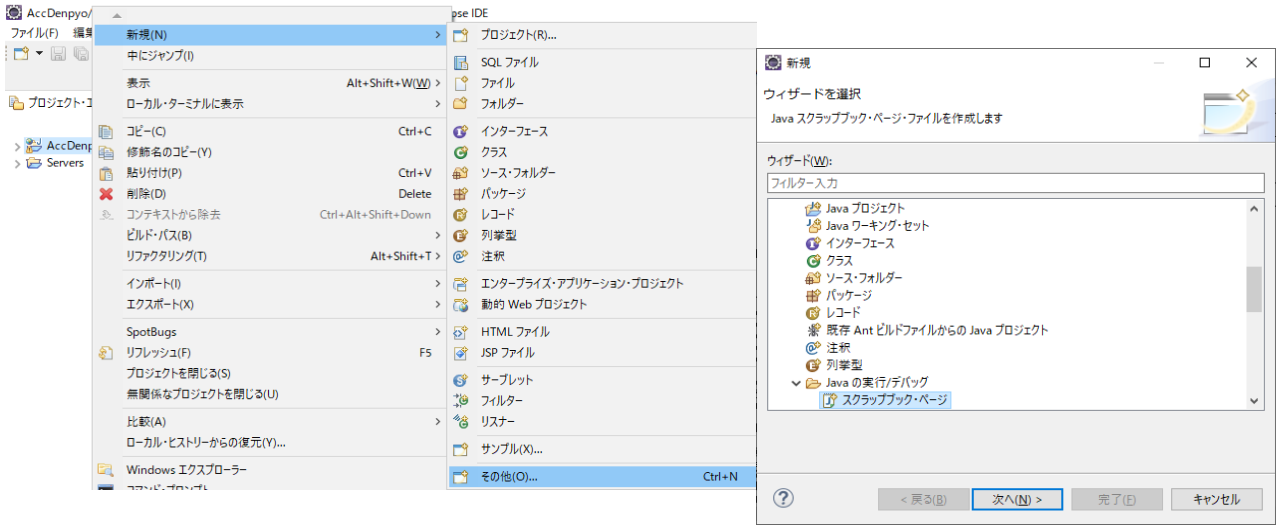
Eclipse はコードの断片だけで実行できる「スクラップブック・ページ」を備えている。

Eclipse を使う段階まできたら、こちらを使う方が開発中資材も使うことができる上に高機能なため JShell を使う価値はあまりありません。

【Eclipse スクラップブック・ページの作成】

プロジェクト･エクスプローラーからプロジェクト(副作用はないため､既存プロジェクトでOK)を選び...

①右クリック→新規(N)→その他...→「ウィザードを選択」画面より、
Java の実行/デバッグ→スクラップブック・ページを選択　→以下、適宜



②スクラップブック・ページを開き、コードを書く→実行したい部分を選んで右クリック→実行(X)

※java.lang パッケージ以外の import が必要なクラスは、「インポートの設定」から型／パッケージを追加する