

Java 基礎

Ver.1.4

Java の基本

- Javaの特徴
- Javaプログラムの作成と実行
- Eclipse

コンピュータは素早く正確に様々な処理 (入力する、計算する、保存する、出力する) が行える機械。

- ・ プログラムとはコンピュータに対して様々な仕事を指示するためのもの
- ・ 仕事を指示するための言語がプログラミング言語

これから Java 言語を使って、コンピュータに仕事(処理)を行わせるプログラムを学習していく。

- Javaは 1995 年にSun Microsystems社によりリリース (*)
されたプログラミング言語および動作環境
- オブジェクト指向言語のなかで最も普及した言語のひとつ
- プラットフォームに依存しない

同じコード・バイナリがOS (Windows / Mac / Linux / Unix 等)
を問わずそのまま動作する

(*) Sun Microsystems 社は2009年 Oracle 社に買収され、現在 Java は Oracle 社によりリリースされている

- JDK (Java Development Kit)
Java の開発環境(コンパイラ・デバッガなどが含まれる)
JDK には JRE が含まれている
- JRE (Java Runtime Environment)
Java で作成したソフトウェアの実行環境



Java はコンパイルとインタプリタという2つのソフトウェアを使って、コンピュータが理解できる言語(機械語)に翻訳する。

① ソースファイル(テキストデータ)を作成

ソースファイルの拡張子は .java (例: Sample.java)

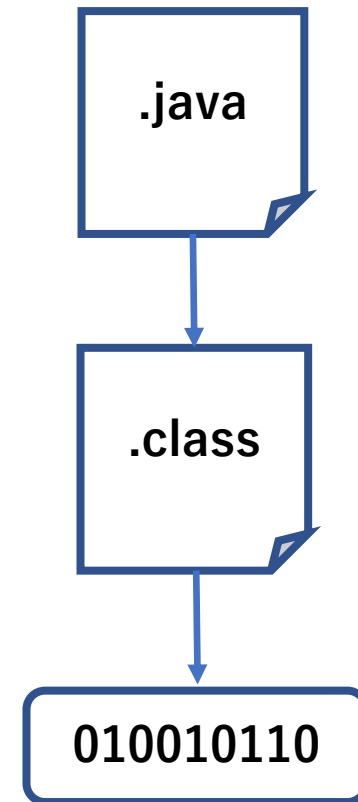
↓ (コンパイラによるコンパイル) ※ コンパイラは JDK に入っている

② クラスファイル

クラスファイルの拡張子は .class (例: Sample.class)

↓ (インタプリタによる解釈実行) ※ インタプリタは JRE に入っている

③ 機械語に変換されて実行される



ソースファイルに記述するプログラムをソースコードと呼ぶ

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

- System.out.println() は、() 内に記述したものを画面に出力する処理 (命令)
- コードを読みやすくするためにインデント(字下げ)や改行を入れる
- インデントは [Tab] で入れる

テキストエディタ(メモ帳等)を使用して前頁のソースコードを入力しましょう。ファイル名は Sample.java とします。

※注意点

- ・ 全角文字ではなく半角文字で作成すること
- ・ 大文字と小文字 (Aとa) を間違えずに作成すること
- ・ 本研修では D:¥work¥myname フォルダに Sample.java を作成すること

javac というコマンド(コンパイラ)を使ってコンパイルする。コンパイルするとクラスファイル(拡張子は .class) が生成される。コンパイルはコマンドプロンプトを起動し、以下のコマンドで実行する。

```
> javac Sample.java
```

コマンドプロンプトを起動し、前頁のコンパイルを実行しましょう(コンパイルに失敗するとエラーメッセージが表示されます)。

コンパイルでクラスファイルが作成されているか確認しましょう。

※注意点

- ・ コマンドプロンプトを起動したら、ソースファイルを作成したフォルダに移動してからコンパイルを実行します。

```
>cd /d D:¥work¥myname
```

- ・ javac にパスが通っていないと実行できません。
以下のエラーが発生する場合は、講師の指示に従ってください。

'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。

java というコマンド (インタプリタ) を使って実行する。
コマンドプロンプトを起動し、以下の通り実行する。

```
>java Sample
```

(注) .class をつける必要はない。

- ・ 実行結果

コマンドプロンプト(画面)上に以下の通り出力される。

```
Hello, World!
```

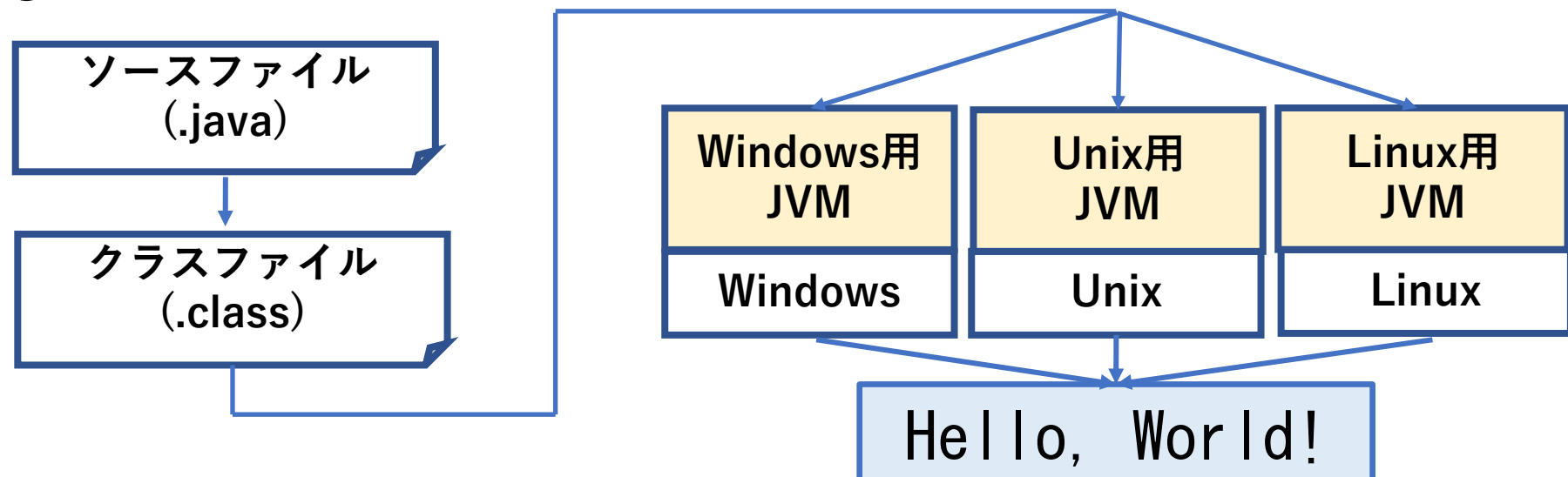
コマンドプロンプトを起動し、前頁の通り実行しましょう。

※注意点

- ・ コマンドプロンプトを起動したら、.class ファイルが生成されたフォルダに移動してから実行します。

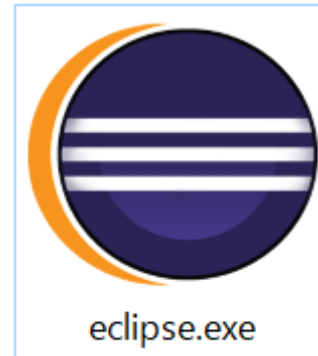
```
>cd /d D:¥work¥myname
```

- ・ インタプリタのことを Java Virtual Machine (JVM/JavaVM) と呼ぶこともある
- ・ JVM は Java のバイナリ形式のファイル (.class ファイル) を、解釈実行できる仮想マシン
- ・ 各 OS 専用の JVM を使うことで、OS を問わず同じ実行結果が得られる



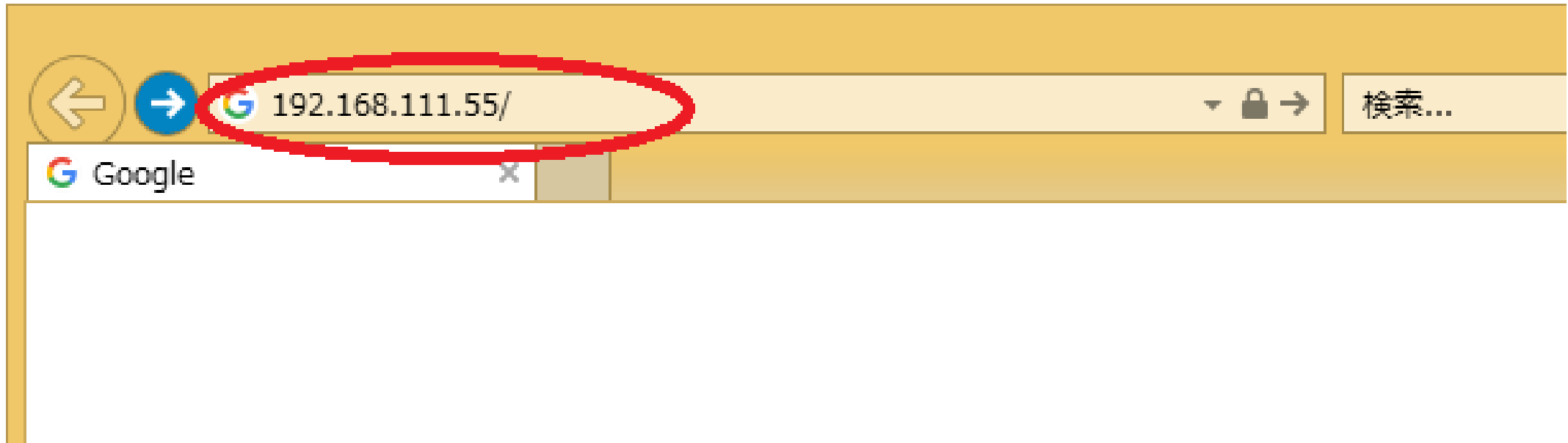
- Eclipse は代表的な Java 用統合開発環境
<http://www.eclipse.org>

- Eclipseを起動
eclipse.exeをダブルクリック



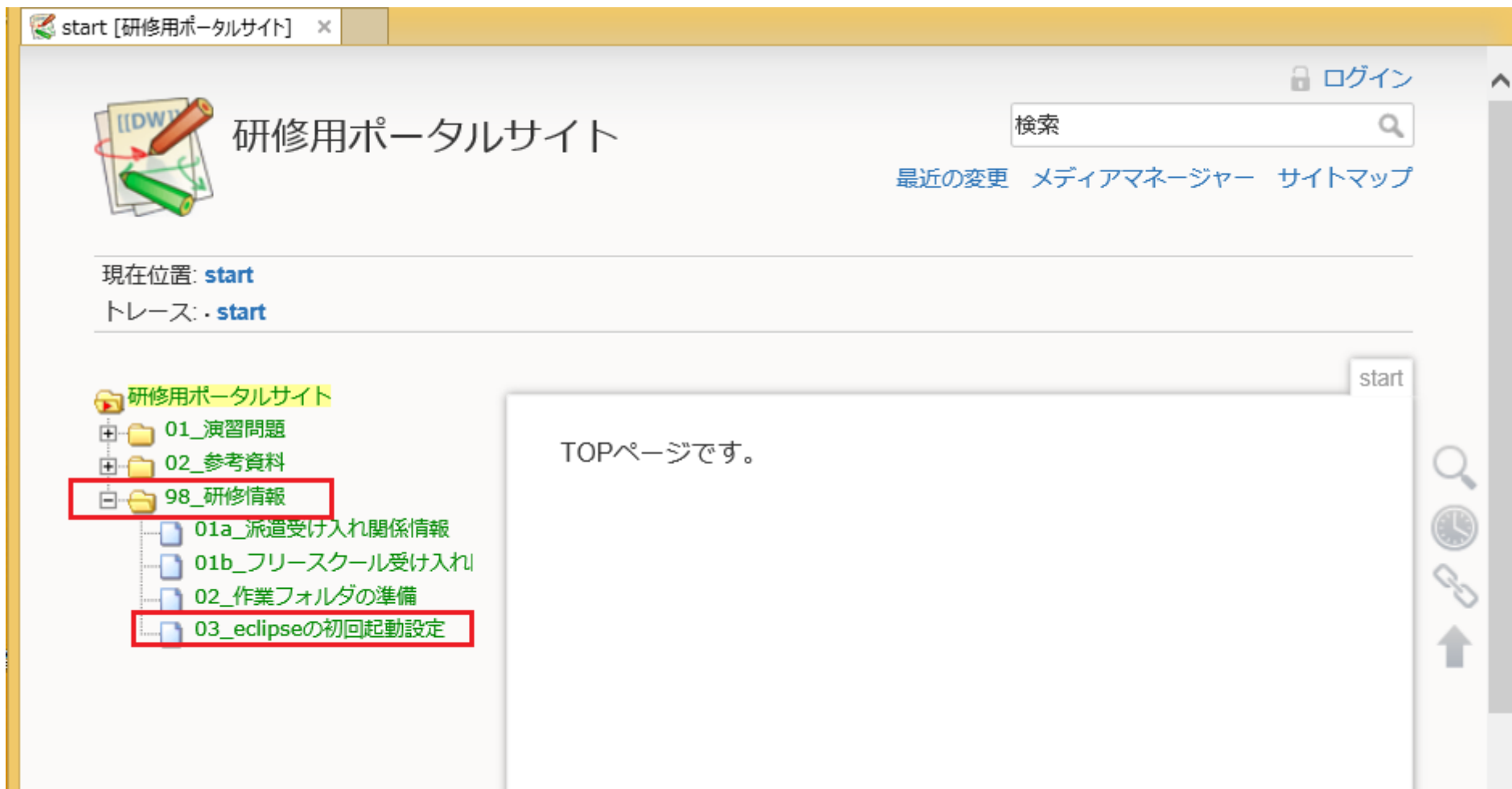
Eclipseの初期起動設定

- ・ インターネットエクスプローラを開き、URLに「192.168.111.55」を入力してEnter



- ・ 研修用ポータルサイトのツリーを展開

「98_研修情報」 → 「03_eclipse初回起動設定」を開く





The screenshot shows a web browser window with the address bar displaying `http://192.168.111.55/doku.php/98_研修情報/eclipse_setup`. The page title is "03_eclipseの初回起動設定...". The main content area is titled "03_eclipseの初回起動設定" and includes a section "このページの内容" with a list of topics: "eclipseの起動方法" and "eclipseの初回起動時の設定". A sidebar on the left shows a tree view of the portal site, with "03_eclipseの初回起動設定" highlighted. A sidebar on the right shows a table of contents for the current page, listing "03_eclipseの初回起動設定", "このページの内容", "前提", "eclipseの起動方法", and "eclipse初回起動時の設定". The page also features a search bar, a login link, and a breadcrumb trail: "現在位置: start » 98_研修情報 » 03_eclipseの初回起動設定".

研修用ポータルサイト

ログイン

検索

最近の変更 メディアマネージャー サイトマップ

現在位置: [start](#) » [98_研修情報](#) » [03_eclipseの初回起動設定](#)

トレース: [start](#) · [03_eclipseの初回起動設定](#)

98_研修情報:eclipse_setup

03_eclipseの初回起動設定

このページの内容

- eclipseの起動方法
- eclipseの初回起動時の設定

前提

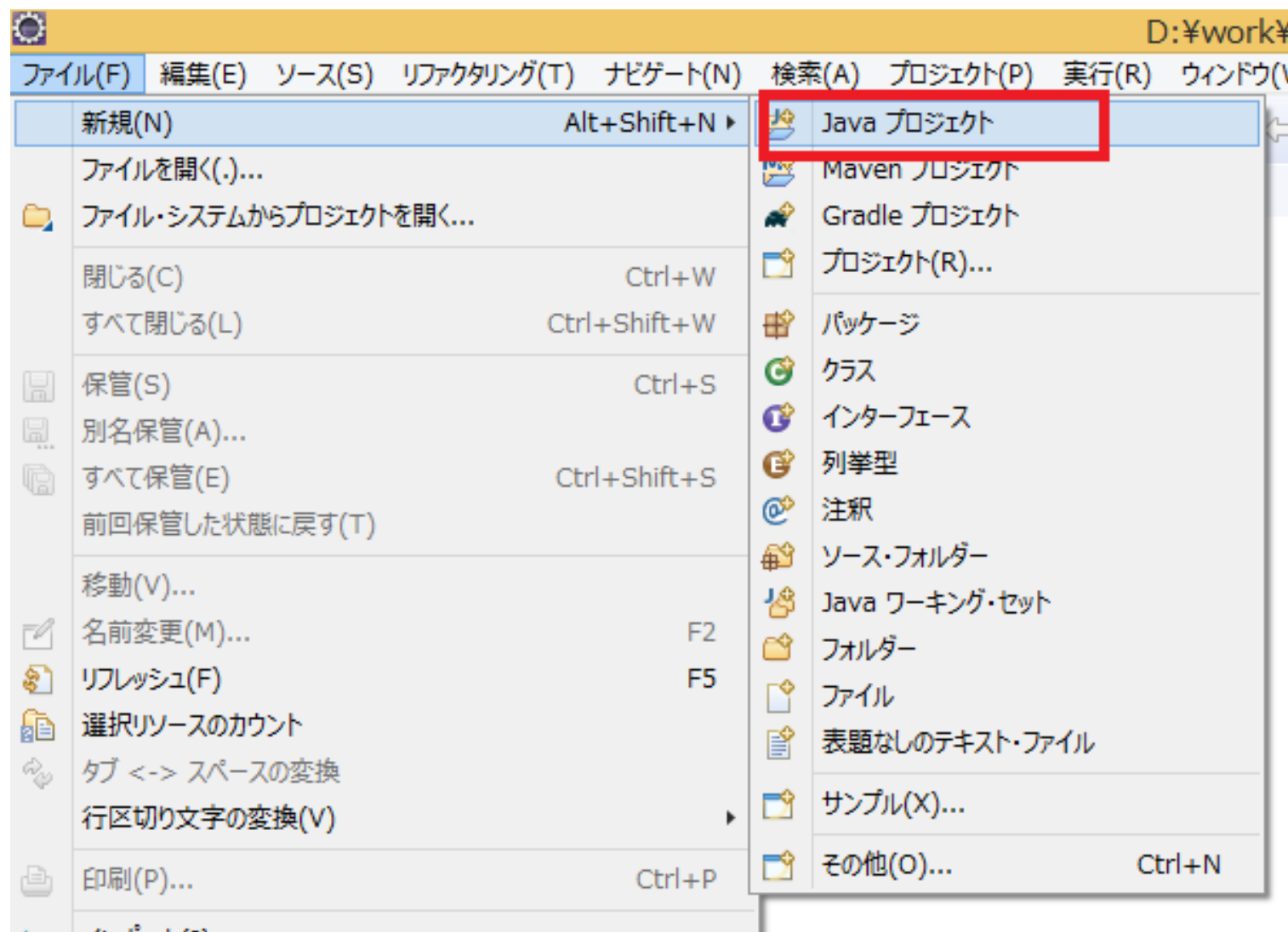
- eclipseで使用するフォルダの準備ができていること
 - フォルダの準備方法は『[02_作業フォルダの準備](#)』を参考のこと

eclipseの起動方法



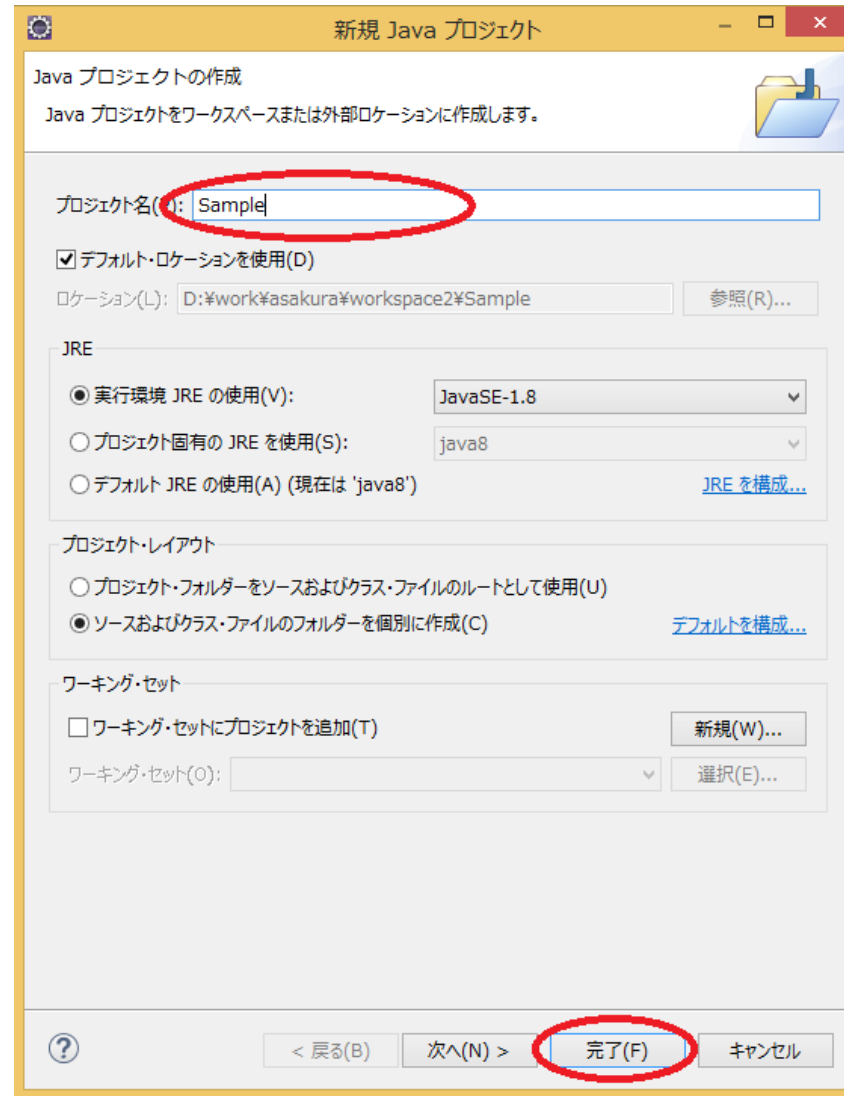
EclipseでJavaプロジェクトを作成

- ・ ファイル → 新規 → javaプロジェクト を選択



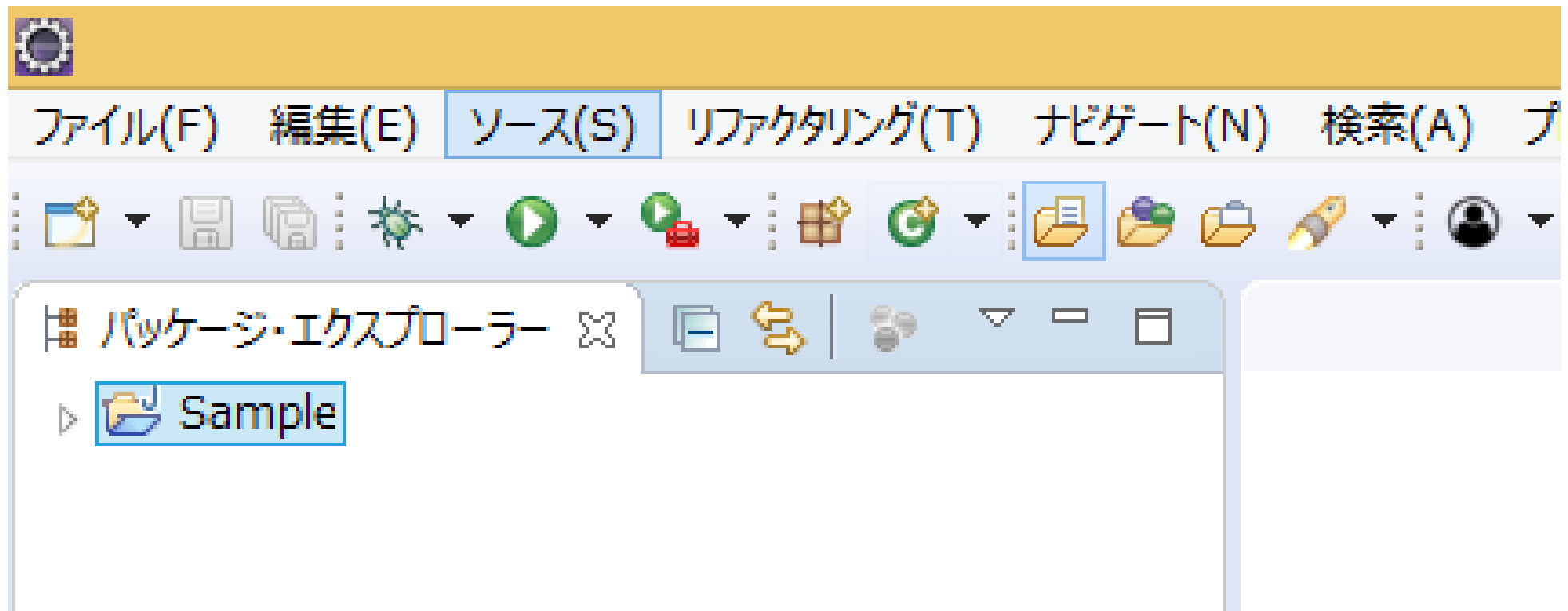
EclipseでJavaプロジェクトを作成

プロジェクト名 (Sample) を入力し「完了」を選択



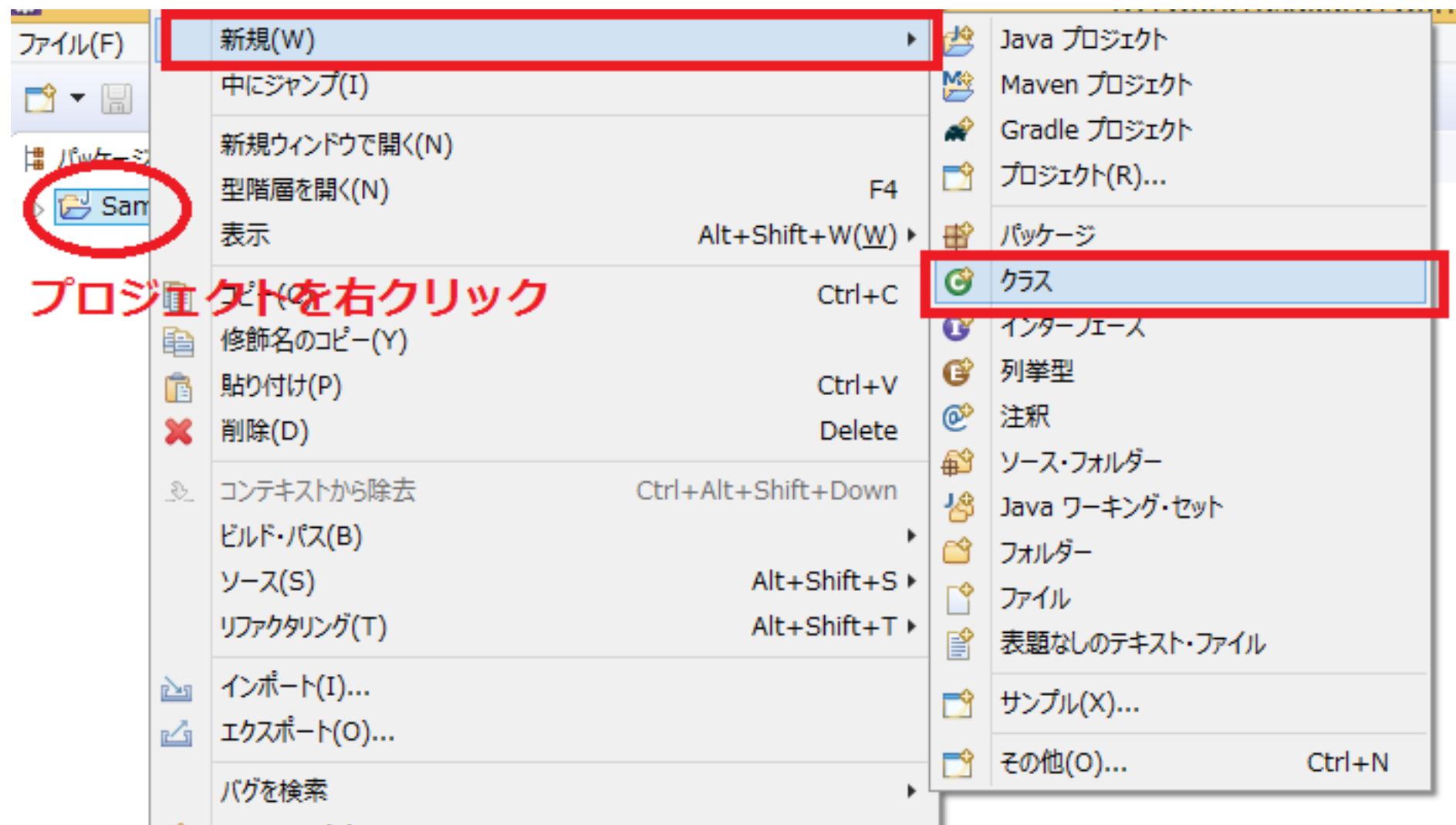
EclipseでJavaプロジェクトを作成

- ・ パッケージエクスプローラーに作成したプロジェクトが表示される



EclipseでJavaプログラムを作成

プロジェクトを右クリック → 新規 → クラス を選択



EclipseでJavaプログラムを作成

- 図のように名前とチェックを付け、完了を選択。

新規 Java クラス

Java クラス

⚠ デフォルト・パッケージの使用は推奨されません。

ソース・フォルダー(D): Sample/src 参照(Q)...

パッケージ(K): (デフォルト) 参照(W)...

☐ エンクロージング型(Y): 参照(W)...

名前(M): Sample

修飾子: ☒ public(P) ☐ パッケージ(C) ☐ private(V) ☐ protected(I)
☐ abstract(I) ☐ final(L) ☐ static(C)

スーパークラス(S): java.lang.Object 参照(E)...

インターフェイス(I): 追加(A)...

除去(R)

どのメソッド・スタブを作成しますか?

☒ public static void main(String[] args)(V)

☐ スーパークラスからのコンストラクター(U)

☐ 継承された抽象メソッド(H)

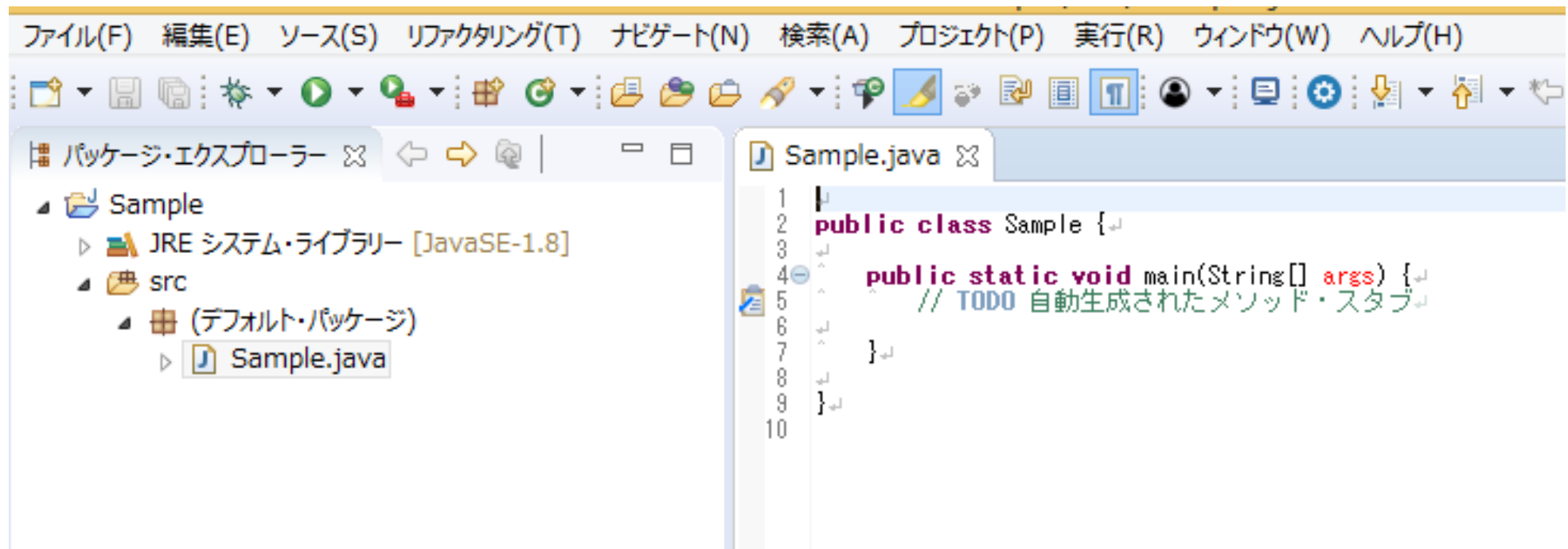
コメントを追加しますか? (テンプレートの構成およびデフォルト値については[ここ](#)を参照)

☐ コメントの生成(G)

完了(E) キャンセル

EclipseでJavaプログラムを作成

- ・ ソースファイルが作成されるので、そこにソースコードを記述して行く



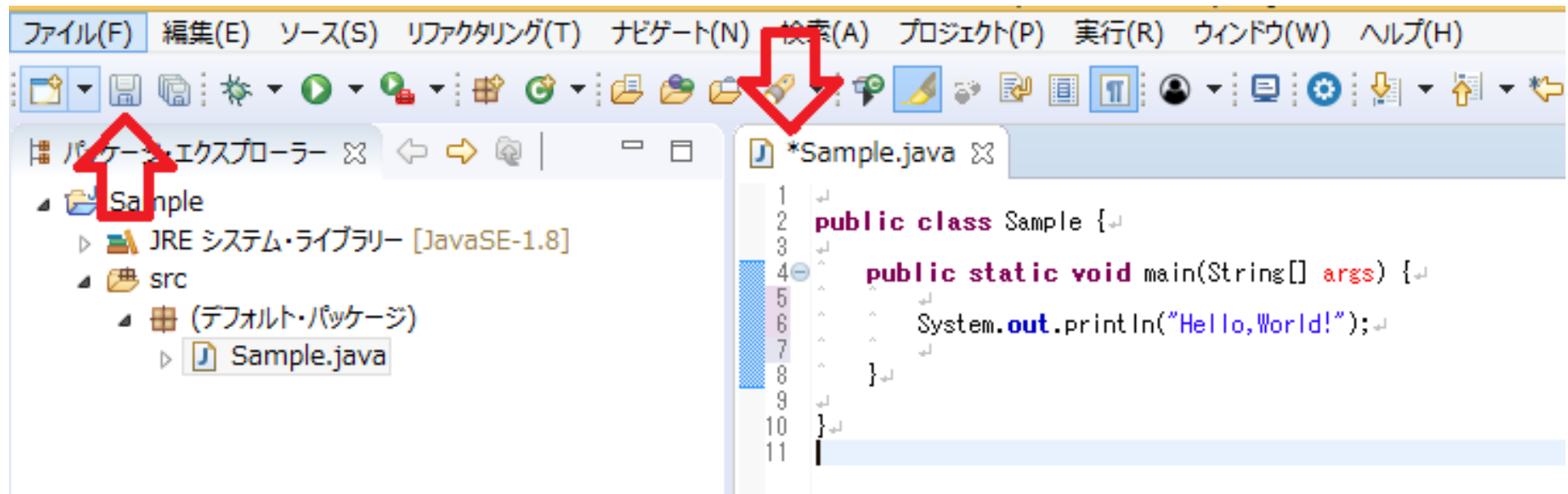
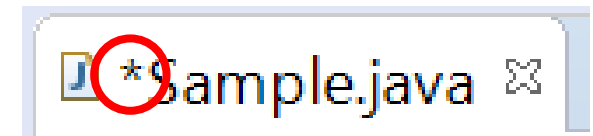
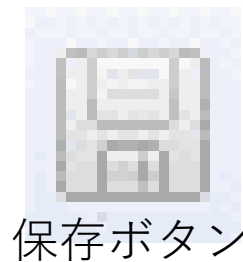
- ・ 先に作成したコード (Sample.java) の内容を、Eclipse で作成しましょう。

- Eclipse でコンパイル(保存)

Eclipse ではソースを保存すると同時にコンパイルが行われる
保存されていないソースには「*」が付く。 保存すると「*」が消える。

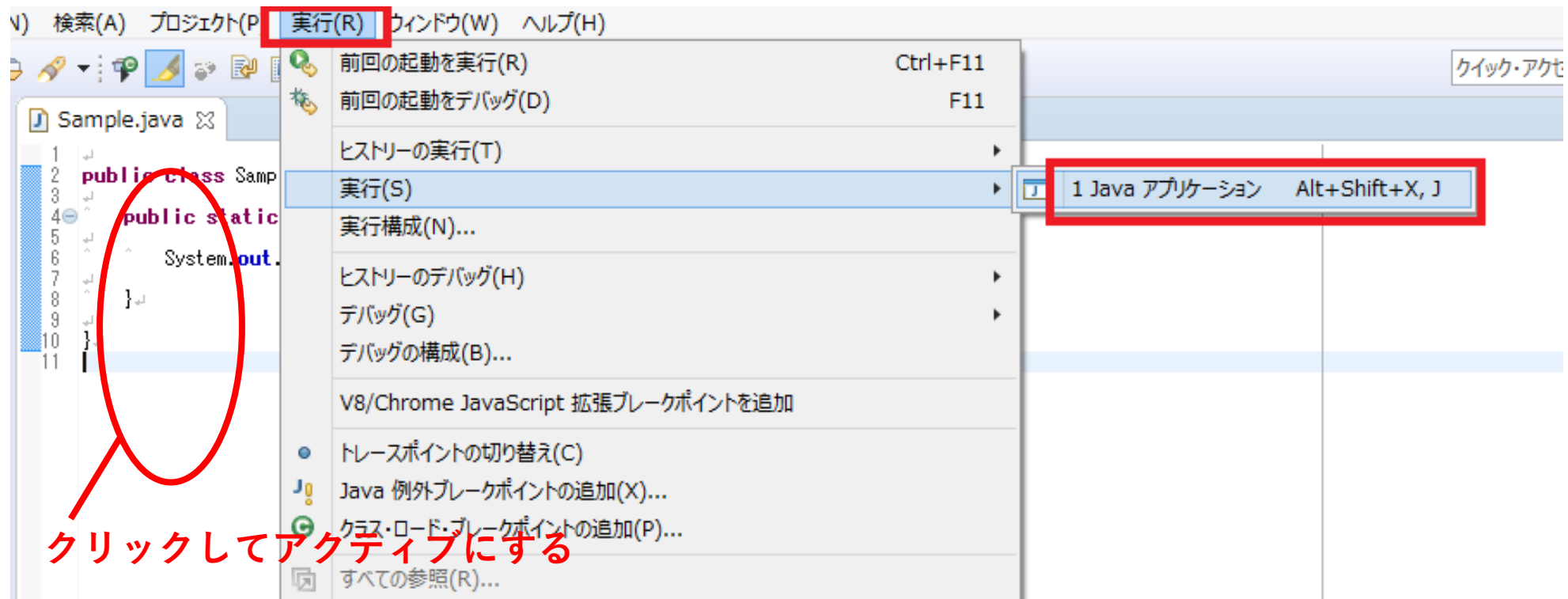
※保存方法:

- ・ [保存]ボタンをクリックする
- ・ 「Ctrl-S」を押下



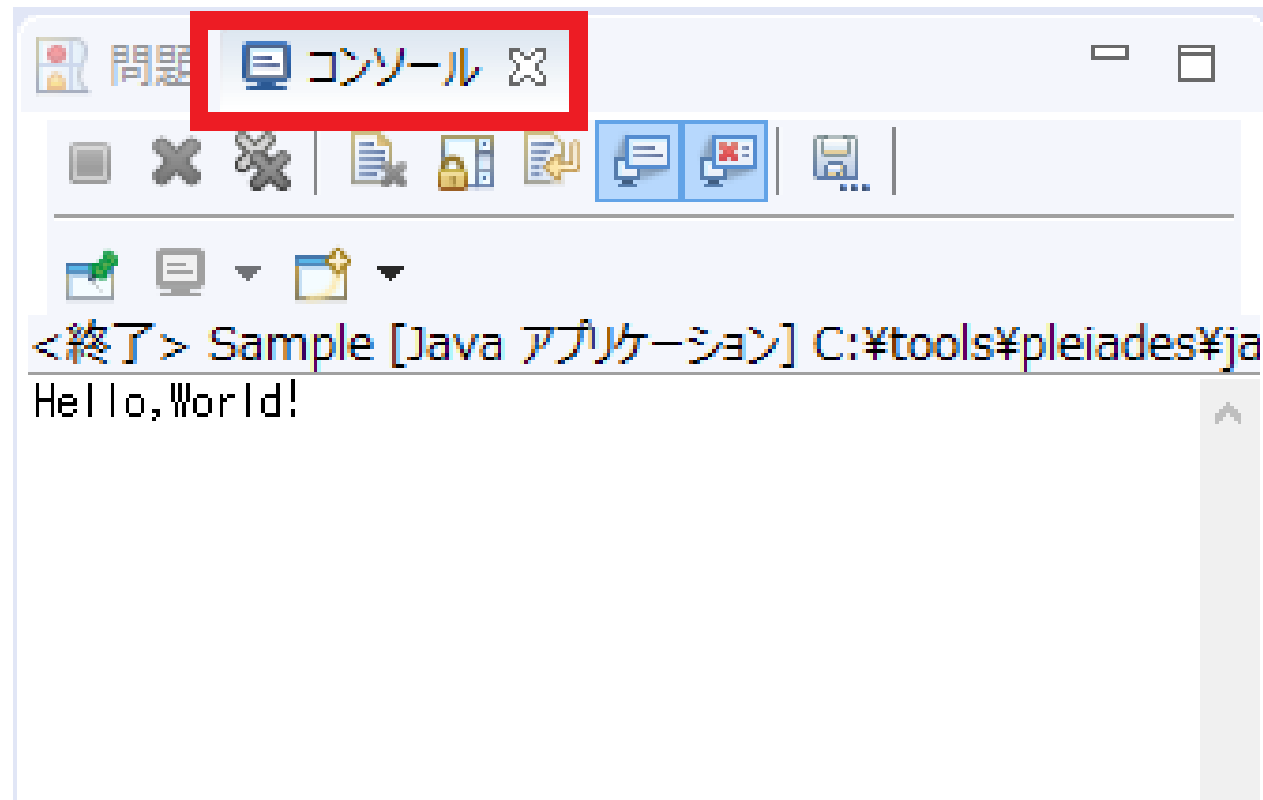
・ 実行

実行したいソースファイルをアクティブにして、ボタンバーの
[実行] ボタンを押下し、[実行]→[Javaアプリケーション] を選択



- ・ 実行結果

実行結果は[コンソール]ビューに表示される



- ・ 先に作成したコード (Sample.java) を書き換えて、コンソールに出力する文字列を (“Hello, World!”) から変更しましょう。
- ・ コンソールに文字列を2行出力しましょう。

Java プログラムの基本構造

- main メソッド
- ステートメント
- クラス

Java のプログラムは、main と記述されている部分から処理が行われる。

```
public class クラス名 {  
    public static void main(String[] args) {  
        プログラム  
    }  
}
```

} main
メソッド

※メソッドの詳細はオブジェクト指向のセクションで説明します。

文(ステートメント)

1つの処理(命令文)の単位を文(ステートメント)と呼ぶ

基本的に上の文から順番に実行される

文の最後は ; で区切る

最初に実行する処理
“Hello, World!”

```
public static void main(String[] args) {  
    System.out.println("Hello, World!");  
    System.out.println("こんにちは、こんばんは!");  
}
```

次に実行する処理
“こんにちは、こんばんは!”

プログラムにはコードを読みやすくするために、インデント(字下げ)やコメントを挿入する。

- ・ // から始まる1行の文章（一行コメント）
- ・ /* ~ */ に囲まれた文章（複数行コメント）

```
public class Sample {  
    public static void main(String[] args) {  
        /*  
        main メソッドから  
        処理が開始されます  
        */  
        // 最初に実行する処理  
        System.out.println("Hello, World!");  
        // 次に実行する処理  
        System.out.println("こんにちは、こんばんは！");  
    }  
}
```

- ・ クラスという単位でプログラムを作成する
- ・ クラスはプログラムをまとめる入れ物
- ・ ソースファイル(.java)には、1つ以上のクラスを定義する
- ・ クラス名とソースファイルの名前は一致させる

```
public class クラス名 {  
    public static void main(String[] args) {  
        プログラム  
    }  
}
```

} クラス

※クラスの詳細はオブジェクト指向のセクションで説明します。

クラス名のつけ方

- ・ 先頭は通常大文字
- ・ キャメルケース（単語の区切りを大文字にする）
（例）Item**M**anager、Sample**P**rogram
- ・ 使用できる記号は _ と \$ のみ
- ・ 数字で始めることはできない
- ・ 予約語（次頁参照）は使用できない

識別子(クラス名やメソッド名や変数名)に予約語および定数(true/false/null)を使用することはできない。

abstract	assert	boolean	break	byte
case	catch	char	class	const
continue	default	do	double	else
enum	extends	final	finally	float
for	goto	if	implements	import
instanceof	int	interface	long	native
new	package	private	protected	public
return	short	static	strictfp	super
switch	synchronized	this	throw	throws
transient	try	void	volatile	while

以前作成した Sample クラスの名前を SampleProgram6 クラスに変更して、以下の処理を実行しましょう。

```
public class SampleProgram6 {  
    public static void main(String[] args) {  
        /*  
        main メソッドから  
        処理が開始されます  
        */  
        // 最初に実行する処理  
        System.out.println("Hello, World!");  
        // 次に実行する処理  
        System.out.println("こんにちは、こんばんは！");  
    }  
}
```

文字と文字列と数値

- 文字と文字列
- エスケープシーケンス
- 数値表現

様々な値を出力する

文字の出力、文字列の出力、数値の出力

```
public class SampleProgram7 {  
    public static void main(String[] args) {  
        // 文字の出力  
        System.out.println('A');  
        // 文字列の出力  
        System.out.println("Hello, ¥nWorld!");  
        // 数値の出力  
        System.out.println(123);  
    }  
}
```

前頁の実行結果を確認しましょう。

Java では文字と文字列を区別して扱う。

- ・ 文字 (1文字) を表すには ' ... ' で括る。
- ・ 文字列を表すには " ... " で括る。

キーボードから入力できないような特殊な文字について、
¥ をつけた2文字の組み合わせで表現する。

エスケープシーケンス	意味
¥b	バックスペース
¥t	水平タブ
¥n	改行
¥r	復帰
¥f	改ページ
¥'	シングルクォーテーション
¥"	ダブルクォーテーション
¥¥	¥文字
¥ooo	8進数の文字コードが表す文字
¥uhhhh	16進数の文字コードが表す文字

数値は '...' や "..." で括らないで記述する。
16進数を数値を表現することもできる。

- 10 ... 10進数の10
- 0x10 ... 16進数の10 = (10)₁₆
- 0xF ... 16進数のF = (F)₁₆

```
public class SampleProgram8 {  
    public static void main(String[] args) {  
        // 10進数の10  
        System.out.print("10進数の10=");  
        System.out.println(10);  
        // 16進数の10  
        System.out.print("16進数の10=");  
        System.out.println(0x10);  
        // 16進数のF  
        System.out.print("16進数のF=");  
        System.out.println(0xF);  
    }  
}
```

前頁の実行結果を確認しましょう。

変数

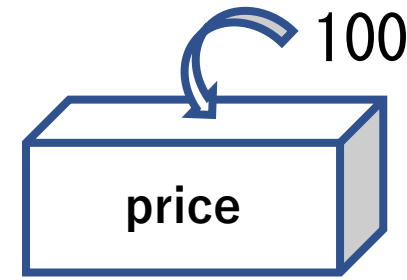
- 宣言と初期化
- 値の代入
- 値の参照

変数とはデータを読み書きする入れ物。
変数を使う前には、必ず宣言を行う必要がある。

- ・ 変数の宣言

[型名] **[変数名]** = [初期値];

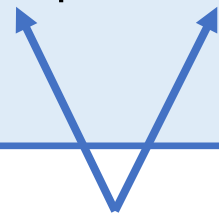
(例) **int** **price** = 100;



- ・ = は、値を変数に格納するという意味(「等しい」という意味ではない)
- ・ 変数名の先頭は通常小文字
- ・ キャメルケース (単語の区切りを大文字にする)
(例) item**P**rice
- ・ 変数名は数字で始めることはできない
- ・ 変数名に予約語は使用できない

変数(price)の値(100)を出力する

```
public class Sample2 {  
    public static void main(String[] args) {  
        int price = 100; // 変数の宣言と初期化  
        System.out.println("価格は" + price + "円です");  
    }  
}
```



price の値を文字列として連結する

変数には、宣言した型の値を格納することができる。
型に合わない値は格納できない。

・ Javaのデータ型

種類	データ型	値の範囲
整数型	byte	1バイト整数 (-128 ~ 127)
	short	2バイト整数 (-32768 ~ 32767)
	int	4バイト整数 (-2147483648 ~ 2147483647)
	long	8バイト整数 (-9223372036854775808 ~ 9223372036854775807)
浮動小数点型	float	4バイト単精度浮動小数点数
	double	8バイト倍精度浮動小数点数
文字型	char	1文字
論理型	boolean	「true」か「false」
文字列型	String	文字列 (複数の文字)

基本
データ型

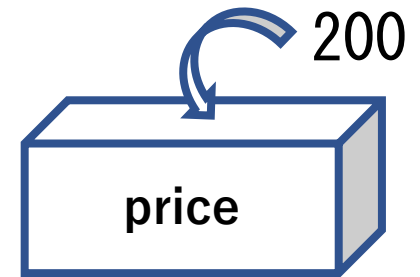
定義済み変数に値の代入

変数にデータを格納することを代入という。

定義済みの変数は、変数の宣言をしなくても値を代入できる。

[定義済みの変数名] = [格納したい値];

(例) price = 200;



※もともと格納されていた値100は消えて200が格納される

```
public class Sample2 {  
    public static void main(String[] args) {  
        int price = 100; // 変数の宣言と初期化  
        System.out.println("価格は" + price + "円です");  
        price = 200; // 定義済み変数に値の代入  
        System.out.println("価格は" + price + "円です");  
    }  
}
```

変数に初期値を格納することを変数の初期化と呼ぶ。
変数の初期化を行わないと、変数の参照はできない。



```
int intNum; // 変数の宣言  
System.out.println(intNum);
```

← コンパイルできない

以下の様に変数に初期値を代入する。



```
int intNum; // 変数の宣言  
intNum = 10; // 初期化  
System.out.println(intNum);
```

もしくは以下のように変数の宣言と初期値の代入を同時に行う(推奨)



```
int intNum = 10; // 変数の宣言と初期化  
System.out.println(intNum);
```

ある変数の値を別の変数に代入することも可能

```
public class Sample3 {  
    public static void main(String[] args) {  
        int intNum1 = 100;  
        int intNum2 = intNum1;  
        System.out.println(intNum1);  
        System.out.println(intNum2);  
    }  
}
```

変数xに100を入れます。変数yに200を入れます。
xの値(100)とyの値(200)を入れ替える処理を作成しましょう。

```
public class SampleProgram9 {  
    public static void main(String[] args) {  
        int x = 100;  
        int y = 200;  
        /*  
        この部分にxの値(100)とyの値(200)を  
        入れ替える処理を挿入する  
        */  
        System.out.println("x = " + x);  
        System.out.println("y = " + y);  
    }  
}
```

代入は、= の左右の型が異なると代入できないが、
代入する値(右側)を変数(左側)の型に変換すれば代入可能

- ・ 精度の低い型から高い型には自動的に変換が行われる

[精度の高い型] = [精度の低い型]

(例)

```
int a = 100;  
long b = a; // int → long に自動的に型変換
```

- ・ 精度の順位
(低い) short / char → int → long → float → double (高い)

精度の高い型から低い型には自動変換できない。

(例)

```
double doubleNum = 10.5  
int intNum = doubleNum;
```

← コンパイルできない

キャスト(変換する型を明示)すれば型変換できる場合がある。
ただし精度が落ちるため、以下の例では小数点以下は切り捨てられる。

(例)

```
double doubleNum = 10.5  
int intNum = (int) doubleNum;  
System.out.println(intNum);
```

変換する型を明示

式と演算子

- 式
- 演算子
- 型

演算子を用いた演算を式と呼ぶ。

(例) $1 + 2$

この例では $+$ が演算子。

- ・ 式の値を出力する

```
public class Sample4 {  
    public static void main(String[] args) {  
        System.out.println(1 + 2);  
    }  
}
```

変数に格納した値で演算を行うことができる。

```
public class Sample5 {  
    public static void main(String[] args) {  
        int x = 100;  
        int y = 200;  
        int sum = x + y;  
        System.out.println("x =" + x);  
        System.out.println("y =" + y);  
        System.out.println("sum =" + sum);  
    }  
}
```

【演習10】

キーボードから数値を2回入力して、合計値を出力する足し算プログラムを作成しましょう。次ページに補足があります。

```
import java.io.*;
public class SampleProgram10 {
    public static void main(String[] args) throws Exception {
        BufferedReader in =
            new BufferedReader(new InputStreamReader(System.in));
        String input1 = in.readLine();
        String input2 = in.readLine();
        // 入力された値を出力する
        System.out.println("input1=" + input1);
        System.out.println("input2=" + input2);
        // 文字列を数値に変換する
        int num1 = Integer.parseInt(input1);
        int num2 = Integer.parseInt(input2);
        /* この部分に入力された値の
         * 合計値を出力する処理を挿入する */
    }
}
```

- BufferedReaderについて

キーボードから値を読み取るのに必要な記載です。

下記記載でキーボードから読み取る準備を行います。

```
BufferedReader in = new BufferedReader(new InputStreamReader(System.in));
```

- in.readLine()について

BufferedReaderを使い実際にキーボードから値を読み取る記載です。下記例では読み取った値を文字列型として変数に代入しています。

```
String input1 = in.readLine();
```

- throws Exceptionについて

BufferedReaderを使用する際に例外と呼ばれるエラーが発生する可能性があるため必要な記載です。例外処理についてはオブジェクト指向の単元で解説します。

- Integer.parseInt(引数)について

Integer.parseInt(引数)は引数の値を文字列型から整数型へ変換する記載です。下記例ではinput1変数の値を整数型に変換してnum1に代入しています。

```
int num1 = Integer.parseInt(input1);
```

・ 主な Java の演算子 ①

演算子	構文	説明
+	$x + y$	数値の場合は加算 (xにyを加える) 文字列の場合は連結 (xにyを連結する)
-	$x - y$	減算 (xからyを引く)
*	$x * y$	乗算 (xにyを掛ける)
/	x / y	除算 (xをyで割る)
%	$x \% y$	剰余 (xをyで割った余り)
++	++x	インクリメント (xに+1してからxを評価する)
	x++	インクリメント (xを評価したあとxに+1する)
--	--x	デクリメント (xを-1してxを評価する)
	x--	デクリメント (xを評価したあとxを-1する)

・ 主な Java の演算子 ②

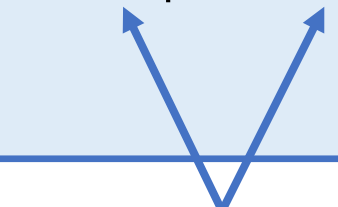
演算子	構文	説明
==	x == y	等価 (xとyは等価)
!=	x != y	非等価 (xとyは等価ではない)
<	x < y	未満 (xはyより小さい)
<=	x <= y	以下 (xはy以下)
>	x > y	より大きい (xはyより大きい)
>=	x >= y	以上 (xはy以上)
&&	x && y	論理積 (AND)
	x y	論理和 (OR)
^	x ^ y	排他的論理和 (XOR)
!	!x	論理否定 (NOT)

・ 主な Java の演算子 ③

演算子	構文	説明
=	<code>x = y</code>	yをxに代入する
+=	<code>x += y</code>	xに (x+y) を代入する → xの値は、xにyを加えた値となる
-=	<code>x -= y</code>	xに (x-y) を代入する → xの値は、xからyを引いた値となる
*=	<code>x *= y</code>	xに (x*y) を代入する → xの値は、xにyを掛けた値となる
/=	<code>x /= y</code>	xに (x/y) を代入する → xの値は、xをyで割った値となる
%=	<code>x %= y</code>	xに (x%y) を代入する → xの値は、xをyで割った余りとなる

+演算子は加算だけでなく、文字列連結演算子の役割もある。

```
public class Sample2 {  
    public static void main(String[] args) {  
        int price = 100; // 変数の宣言と初期化  
        System.out.println("価格は" + price + "円です");  
    }  
}
```



price の値を文字列として連結する

インクリメントとは、変数の値を1増やす演算のこと。

```
x++; // x = x + 1; と同じ
```

デクリメントとは、変数の値を1減らす演算のこと。

```
y--; // y = y - 1; と同じ
```

- ・ 前置インクリメント演算子

`++x; //` (x に +1 してから x を評価する)

- ・ 後置インクリメント演算子

`x++; //` (x を評価した後に +1 する)

実行すると x1=, y1=, x2=, y2= それぞれの値はなんと出力されるでしょう。

```
public class SampleProgram11 {  
  
    public static void main(String args[]) throws {  
        int x = 100;  
        int y = 100;  
  
        System.out.println("x1=" + x++);  
        System.out.println("y1=" + --y);  
  
        System.out.println("x2=" + x);  
        System.out.println("y2=" + y);  
    }  
}
```

= は「等しい」という意味ではなく「代入」するための演算子。
代入演算子は = だけでなく、+= とほかの演算子を組み合わせて利用できる。

```
a += b; // a = a + b; と同じ
```

```
a -= b; // a = a - b; と同じ
```

キーボードから数値を3回入力して、合計値を出力する足し算プログラムを += 演算子を使用して作成しましょう。

異なる型どうしで演算を行った場合、精度の低い型が精度の高い型のサイズに変換されてから実行される。

```
int x = 10;  
double p = 3.14  
double result = x * p;
```

int 型の x が double 型に変換されて演算される。
result の値は double 型の 31.4 となる。

同じ型どうしで演算が行われ結果も同じ型となる。

```
int x = 3  
int y = 2  
double div = x / y;
```

x / y の演算の結果は、一旦 int 型で計算されるため 1 となる。
それを double に格納するため、div の値は 1.0 となる。

小数点以下も正しく計算するためには、キャスト演算子を使う。

```
double div = (double) x / (double) y;
```


キーボードから数値を3回入力して、合計値と平均値を出力する(小数点以下まで出力する)プログラムを作成しましょう。

補足

キーボードから受け取った値を実数へ変換するには以下の記載を使用してみましょう。

```
Double.parseDouble(input1);
```

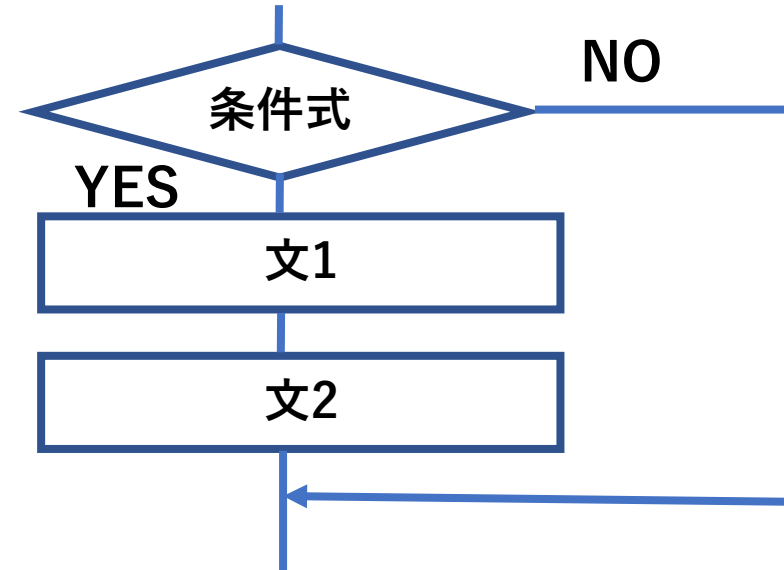
関係演算子と分岐処理

- if 文
- switch 文

場合に応じた処理を if 文で構築することができる。

- 基本構文

```
if (条件式) {  
    文1;  
    文2;  
}
```



() 内の条件式が成立 (演算結果がtrue) するとき文1、文2を実行する。

(例) score の値が 80 以上の場合にのみ "合格" と表示する

```
if (score >= 80) {  
    System.out.println("合格");  
    System.out.println("おめでとうございます");  
}
```

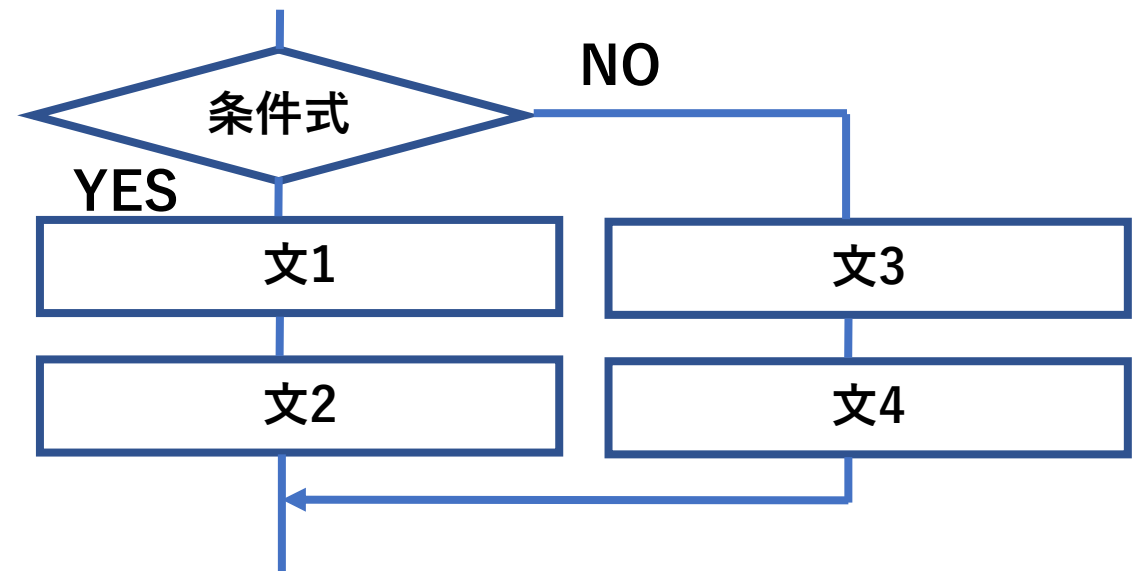
【演習14】

キーボードから年齢(x)を入力し、 x が20未満だったら「子供です」と出力するコードを作成しましょう。

if ~ else 文

・ 基本構文

```
if (条件式) {  
    文1;  
    文2;  
} else {  
    文3;  
    文4;  
}
```



()内の条件式が成立(演算結果がtrue)するとき文1、文2を実行する。

条件式が成立しない(演算結果がfalse)とき文3、文4を実行する。

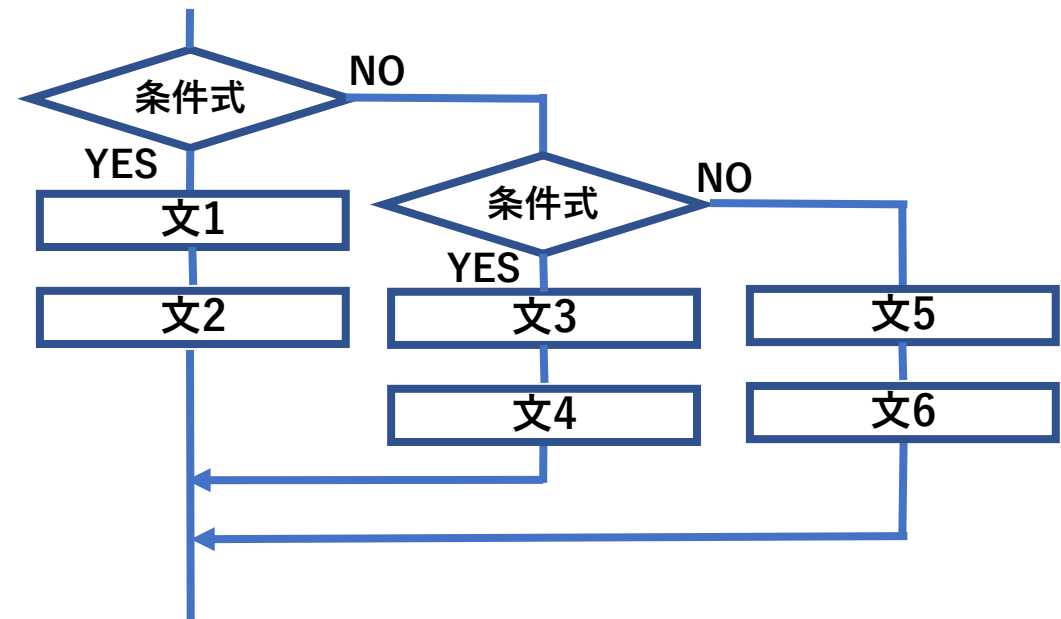
キーボードから年齢(x)を入力し、 x が20未満だったら「子供です」、それ以外は「大人です」と出力するコードを作成しましょう。

キーボードから数値(x)を入力し、 x が偶数だったら「偶数」、それ以外は「奇数」と出力するコードを作成しましょう。

複数の条件を判断する

基本構文

```
if (条件式1) {  
    文1;  
    文2;  
} else if (条件式2) {  
    文3;  
    文4;  
} else {  
    文5;  
    文6;  
}
```



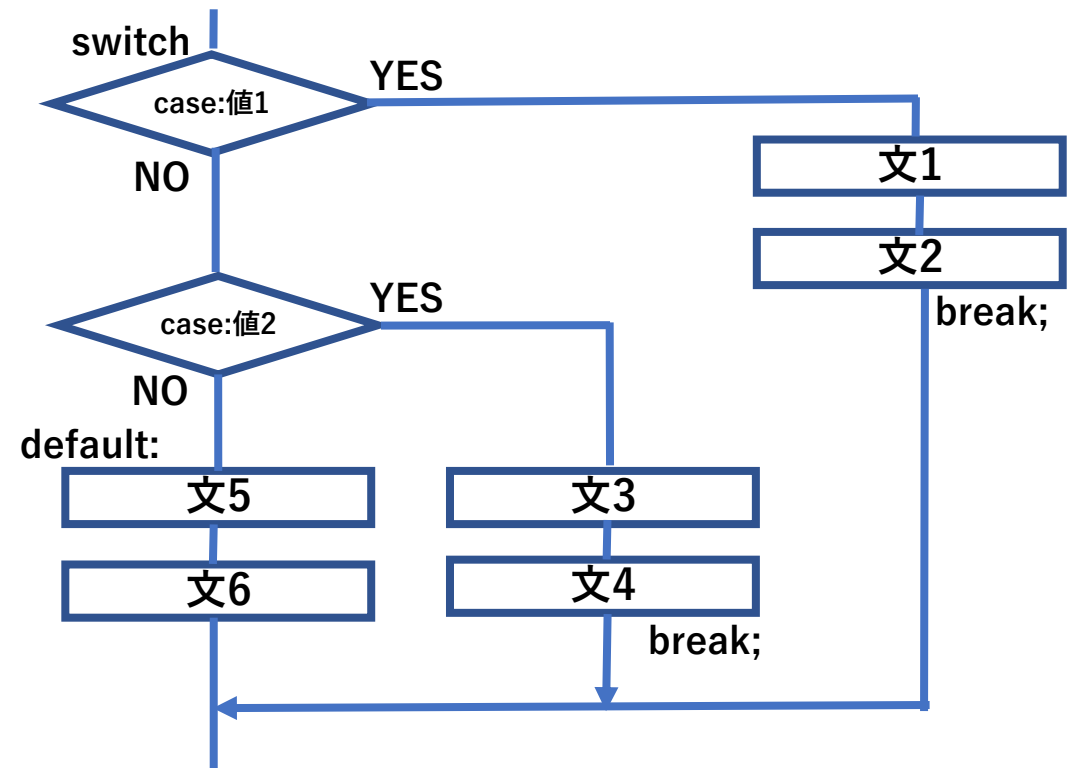
- 条件式1が成立すれば、文1、文2を実行してif文を終了
- 条件式1が成立しないとき:
 - 条件式2が成立すれば、文3、文4を実行してif文を終了
 - 条件式2が成立しなければ、文5、文6を実行してif文を終了

switch～case 文

switch文では、ある値の内容によって、処理を分岐することができる。

・ 基本構文

```
switch (条件値) {  
    case 値1:  
        文1;  
        文2;  
        break;  
    case 値2:  
        文3;  
        文4;  
        break;  
    default:  
        文5;  
        文6;  
}
```



()内の条件値によって、以下の通り処理を分岐する。

- ・ 値1と等しければ、文1、文2を実行してswitch文を終了。
- ・ 値2と等しければ、文3、文4を実行してswitch文を終了。
- ・ どの値とも一致しなければ、文5、文6を実行してswitch文を終了。

switch()内の値(条件値)には、int型 char型 enum型(後述)などを指定できる。

【演習17】

キーボードから数値(x)を入力し、場合に応じて以下のメッセージを出力するコードを作成しましょう。

x の値	コンソールに表示される内容
1	松
2	竹
3	梅
上記以外の値	XX

論理演算子を使うと、条件を組み合わせて複雑な条件を作ることができる。

&&	x && y	論理積 (AND) ... かつ
	x y	論理和 (OR) ... または
!	!x	論理否定 (NOT)

```
int a = 10;

if (a > 3 && a == 4) {
    // 実行されない
}

if (a > 3 || a == 4) {
    // 実行される
}
```

```
int a = 10;

if (!(a == 4)) {
    // 実行される
}

if (a != 4) {
    // 実行される
}
```

キーボードから文字列を入力し、入力された文字が「y」もしくは「Y」の場合に「Yes」と出力するコードを作成しましょう。次ページに補足があります。

前ページで学んだ「==」は文字列では使えないので、この演習では[.equals()メソッド]を使ってみましょう。

構文

```
変数.equals(“比較したい文字列”)
```

例文

```
String str = “abc”;  
str.equals(“abc”)           //この例では true となる
```

反復処理

- for 文
- while 文

for文を使うと反復処理を行うことができます。

- ・ 基本構文

```
      ①      ②      ④  
for (初期化処理; 条件式; 更新処理) {  
    文1;  
    ③ 文2;  
}
```

- ・ 処理の流れ

- ① 「初期化処理」を実行する
- ② 「条件式」を判定（trueのならば③／falseならばfor 文を終了）
- ③ 文1、文2 を実行する
- ④ 「更新処理」を実行し②へ

for文を使ったコード例

```
public class SampleProgram19 {  
    public static void main(String[] args) {  
        for(int i = 1; i <= 5; i++) {  
            System.out.println(i);  
        }  
  
        for(int i = 0; i < 5; i++) {  
            System.out.println(i);  
        }  
    }  
}
```


- ・ 前頁のコードを作成し実行してみましょう。
- ・ 表示される数値から、初期化处理、条件式、更新処理の関係を考えましょう。

キーボードから数値(x)を入力し、入力された数だけ「★」を出力するコードを作成しましょう。

while 文では条件が true である限り、指定された文を繰り返し処理する。

- ・ 基本構文

```
while(条件式) {  
    文1;  
    文2;  
}
```

- ・ 例

```
int x = 0;  
while(x < 10) {  
    System.out.println(x);  
    x++;  
}
```

while文との違いは、文が必ず1回は実行されるところ。

- 基本構文

```
do {  
    文1;  
    文2;  
} while(条件式);
```

- 例

```
int x = 100;  
do {  
    System.out.println(x);  
    x++;  
} while(x < 10);
```

処理の流れを変える

- break 文
- continue 文

break文は以下の通り処理の流れを変える。

- 反復処理(for文、while文)を途中で中断して終了
- switch～case の分岐処理を終了
- 制御文が入れ子になっている場合は、今いる制御文だけを中断

```
for (int i = 0; i < 10; i++) {  
    if (i == 5) {  
        break;  
    }  
    System.out.println("i=" + i);  
}
```

入れ子の制御文で break 文を用いた場合の例

```
for (int i = 0; i < 10; i++) {  
    for (int j = 0; j < 10; j++) {  
        if (j == 5) {  
            break;  
        }  
        System.out.println("i=" + i + "/" + j);  
    }  
}
```

前頁のコードを作成し実行してbreak文の動作を確認しましょう。

continue文は反復処理を中断するが、次のループから継続する。反復処理そのものが終了するわけではない。

```
for (int i = 0; i < 10; i++) {  
    for (int j = 0; j < 10; j++) {  
        if (j == 5) {  
            continue;  
        }  
        System.out.println("i=" + i + "/" + j);  
    }  
}
```

前頁のコードを作成し実行してcontinue文の動作を確認しましょう。

配列

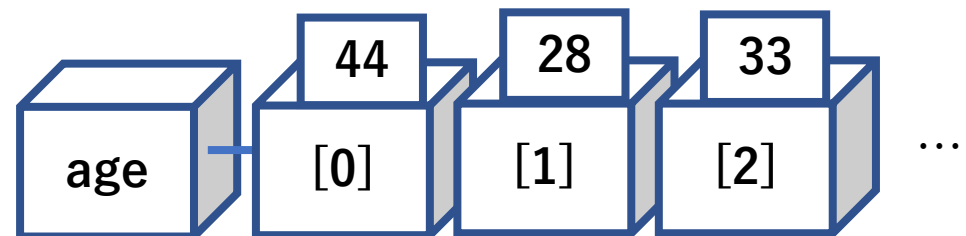
- 配列への値の代入と参照
- 拡張 for 構文
- 配列変数
- 多次元配列

配列を使わずに、30人分の年齢データを管理しようとした場合、30人数分の変数が必要となる。

```
int age1 = 44;  
int age2 = 28;  
int age3 = 33;  
int age4 :
```

配列を使えば、複数の同じ型の値をまとめて(同じ変数名で)管理することができる。

```
int[] age = new int[30]  
age[0] = 44  
age[1] = 28  
age[2] = 33  
age[3] :
```



配列を使う前には、必ず宣言と初期化を行う必要がある。

・配列の宣言

型名[] 変数名 = new 型名[配列の大きさ]

(例) `int[] age = new int[30];`

配列の生成と同時に値を設定することもできる。
この場合配列の大きさは、自動的に要素の数に設定される

型名[] 変数名 = {要素1,要素2,要素3,...}

(例) `int[] age = {44,28,33,18};`

配列の各要素は番号で管理されている。
配列の各要素に値を代入するには、添え字に番号を指定する。

変数名[添え字] = [格納したい値];
(例)

```
int[] age = new int[3];  
age[0] = 44;  
age[1] = 28;  
age[2] = 33;
```

- ・ 上記の例では、大きさ3の配列を宣言
- ・ 添え字が0～2の3つの要素が利用可能
- ・ 一度宣言した配列の大きさは後から変更することができない

配列の大きさを超える添え字を指定して値を代入しようとした場合の動作を確認しましょう。

配列に格納した情報を参照する場合、for 文を利用すると便利。

```
int[] age = new int[3];  
age[0] = 44;  
age[1] = 28;  
age[2] = 33;  
for (int i = 0; i < 3; i++) {  
    System.out.println(age[i]);  
}
```


配列やリスト(後述)を対象に for 文を利用する場合、拡張 for 構文を利用できる。

- ・ 基本構文

```
for(変数定義 : 配列名またはリスト名) {  
    文1;  
    文2;  
    ...  
}
```

前頁と同じ処理が下記のコードで実現できる。

```
int[] age = new int[3];  
age[0] = 44;  
age[1] = 28;  
age[2] = 33;  
for (int val : age) {  
    System.out.println(val);  
}
```

```
for (int val : age) {  
    System.out.println(val);  
}
```

- ① int 型の変数 val を定義
- ② 配列 age から、1番目の要素を取り出し ① の変数 (val) に代入
- ③ ブロック内の文 (System.out.println()) を実行
- ④ 配列 age から、次の要素を取り出し ① の変数 (val) に代入して ③ へ
- ⑤ age から取り出す要素がなくなったらループを終了

配列の長さ(大きさ)は以下のコードで調べることができます。

配列変数名.length

(例)

```
int[] age = {44, 28, 33, 18};  
System.out.println("配列の長さ=" + age.length);  
for (int i = 0; i < age.length; i++) {  
    System.out.println("age[" + i + "]= " + age[i]);  
}
```

配列を格納する変数を配列変数と呼ぶ。
前頁の例では `int[]` 型の変数 `age` が配列変数。

配列変数には配列を代入することができる。

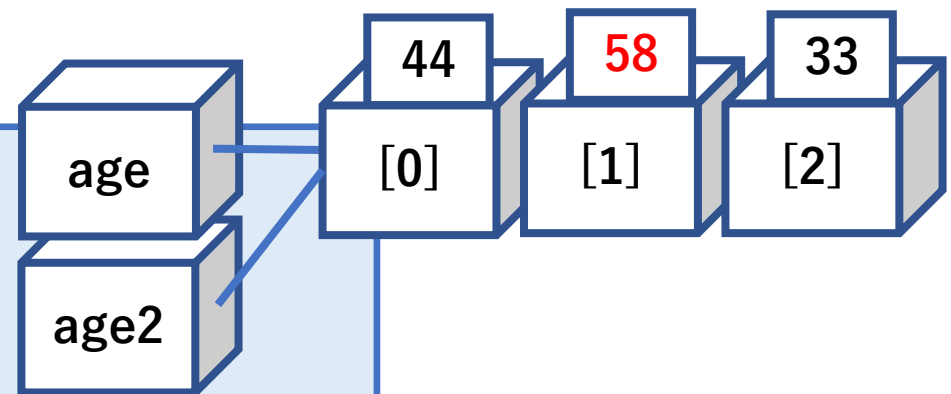
(例)

```
int[] age = new int[3];  
age[0] = 44;  
age[1] = 28;  
age[2] = 33;  
int[] age2 = age;  
for (int val : age2) {  
    System.out.println(val);  
}
```

前頁の例では age と age2 は異なる2つの配列ではなく、同じ1つの配列を指している。

(例)

```
int[] age = new int[3];  
age[0] = 44;  
age[1] = 28;  
age[2] = 33;  
int[] age2 = age;  
// age の添え字1の要素の値を58に変更  
age[1] = 58;  
for (int val : age2) {  
    // age2 の要素の値も更新されている  
    System.out.println(val);  
}
```



前頁のコードを実行し、配列変数の動作を確認しましょう。

配列の要素をさらに配列とすることで、多次元配列を作成できる。2次元配列は x 軸、y 軸で表現した「表」のイメージ。

- ・ 多次元配列の宣言（2次元配列の例）

型名[][] 変数名 = new 型名[配列の大きさ1][配列の大きさ2]

（例）int[][] area = new int[2][3]; と記述した場合

	0	1
0	78	23
1	9	21
2	49	918

```
area[0][0] = 78;  
area[0][1] = 9;  
...
```

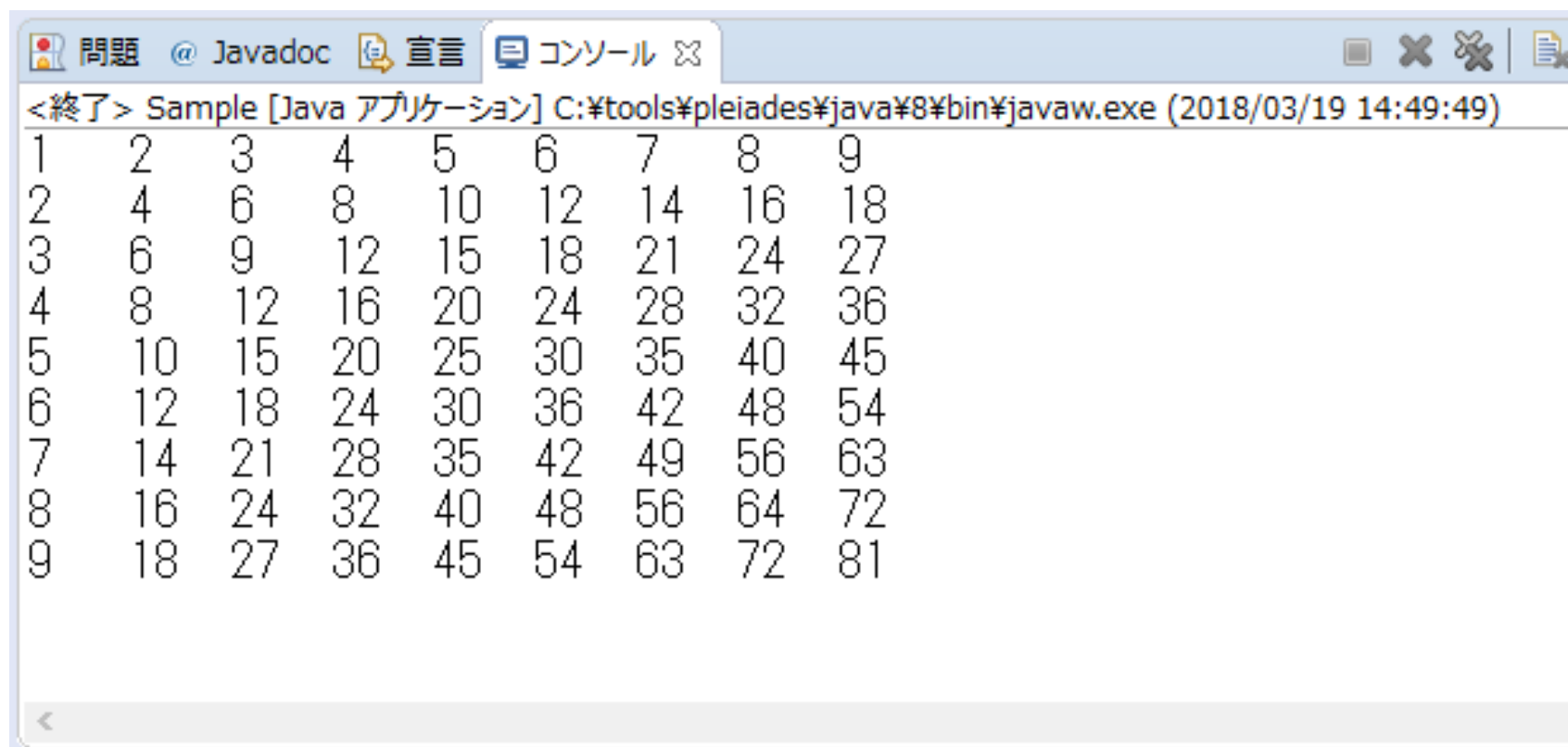
- ・ 前頁の2次元配列に値を代入するコードを作成しましょう。
- ・ 値を代入した2次元配列について、拡張 for 構文で全ての値を出力しましょう。

復習演習

- ・ キーボードから身長と体重を入力してBMI値を計算するコードを作成しましょう。
- ・ 計算した BMI 値によって、警告(「痩せすぎ」「ふつう」「太りすぎ」)を出力するコードを作成しましょう。

- ・ 九九の一覧表を出力するコードを作成しましょう。

出力例:



```
<終了> Sample [Java アプリケーション] C:\tools\pleiades\java\8\bin\javaw.exe (2018/03/19 14:49:49)
1    2    3    4    5    6    7    8    9
2    4    6    8    10   12   14   16   18
3    6    9    12   15   18   21   24   27
4    8    12   16   20   24   28   32   36
5    10   15   20   25   30   35   40   45
6    12   18   24   30   36   42   48   54
7    14   21   28   35   42   49   56   63
8    16   24   32   40   48   56   64   72
9    18   27   36   45   54   63   72   81
```

- ・ キーボードから10個の数値を入力して、それを小さい順に並べて出力するコードを作成しましょう。