

プログラミング実習

日本工学院八王子専門学校 ITカレッジ
情報処理科 モバイルアプリ開発コース

木崎 悟

最終更新日 2015/11/4

授業内容

- Java 言語を使ったプログラミング
 - Javaの文法とその意味
 - オブジェクト指向言語の基本事項
- 講義で修得した知識を活用できる力をつける
 - 適宜演習を行う
- プログラミング環境の構築
 - 統合開発環境やライブラリ等を自分で見つけて活用できるようになる
- 資格取得
 - Oracle Certified Java Programmer, Bronze SE 7

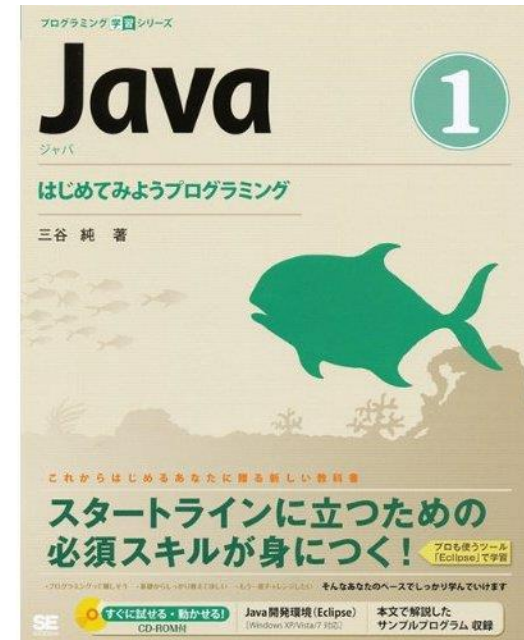
教科書

Java 1 はじめてみようプログラミング
(三谷純 著)

Java1 はじめてみようプログラミング
(三谷純著)

本書は、現在もっとも幅広く使われているプログラミング言語の1つ「Java」の入門書です。文法やプログラムの基本知識を、筆者が大勢の学生にJavaの授業をしてきた経験を活かして、やさしく解説します。本巻では、変数やif文・switch文による条件分岐、for文・while文を使った繰り返しから、クラスやメソッドの使い方・作り方まで、つまずきやすいところをケアしながら説明します。

**スタートラインに立つための
必須スキルが身につく！**



大型本： 280ページ
出版社： 翔泳社 (2010/1/29)
言語 日本語
ISBN-10： 4798120987
ISBN-13： 978-4798120980
発売日： 2010/1/29

開発に必要なもの

- JDK (Java Development Kit)
- Android SDK (Software Development Kit)
- Eclipse (IDE: 統合開発環境)
 - Android Plug-in
 - Platform API
 - Android Virtual Device
 - Pleiades (日本語化Plug-in)

Javaの概要

Java はこんなところで使われています

- クライアントサイド
 - スタンドアロン・アプリ (CUI, GUI)
 - アップレット (Web ブラウザ上)
- サーバサイド
 - サーブレット, JSP (Java Server Pages)
 - Web サービス (Web API を提供)
 - クラウド (Google App Engine, Aptana 等)
- 組み込み
 - 携帯電話各社 (iアプリ, EZアプリ, S! アプリ)
 - Android フレームワーク (Java と類似のプラットフォーム)

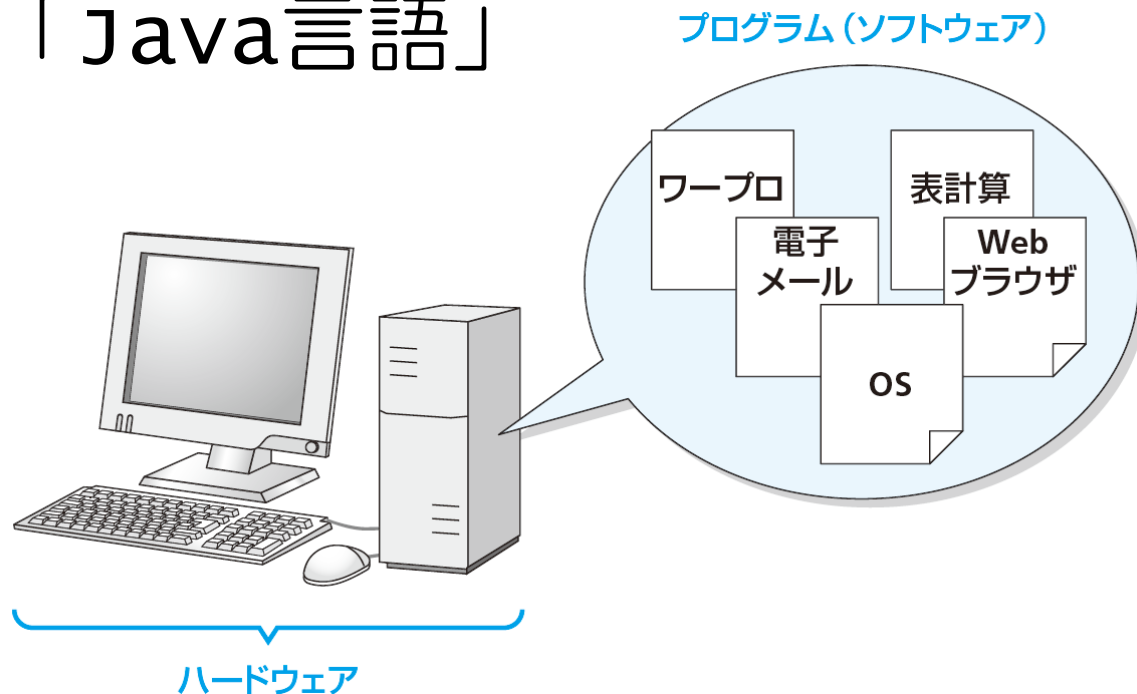
統合開発環境 Eclipse

- Eclipse は90年代後半に IBM が開発した IDE
 - <http://www.eclipse.org/>
 - 現在はオープンソース。Eclipse Foundation がメンテナンスしている
- 最も広く利用されている IDE の一つ
- 当初は Java 言語での開発の為に設計された
- プラグインにより機能拡張が容易
 - C, C++, JavaScript, Perl, Ruby, Python, PHP, Scala 等のプラグインがある
- クロスプラットフォーム
 - windows, Mac OS X, Linux 等に対応
- 主にJava言語で開発されている
 - GUI に関しては SWT (Standard widget Toolkit) と呼ばれるライブラリを使い、OS ネイティブに近い性能を実現している

第1章 Java言語に触れる

プログラムとは

- コンピュータに命令を与えるものが「プログラム」
- プログラムを作成するための専用言語が「プログラミング言語」
- その中の1つに「Java言語」がある



Java言語のプログラムコード

Java言語のプログラムコードを見てみよう

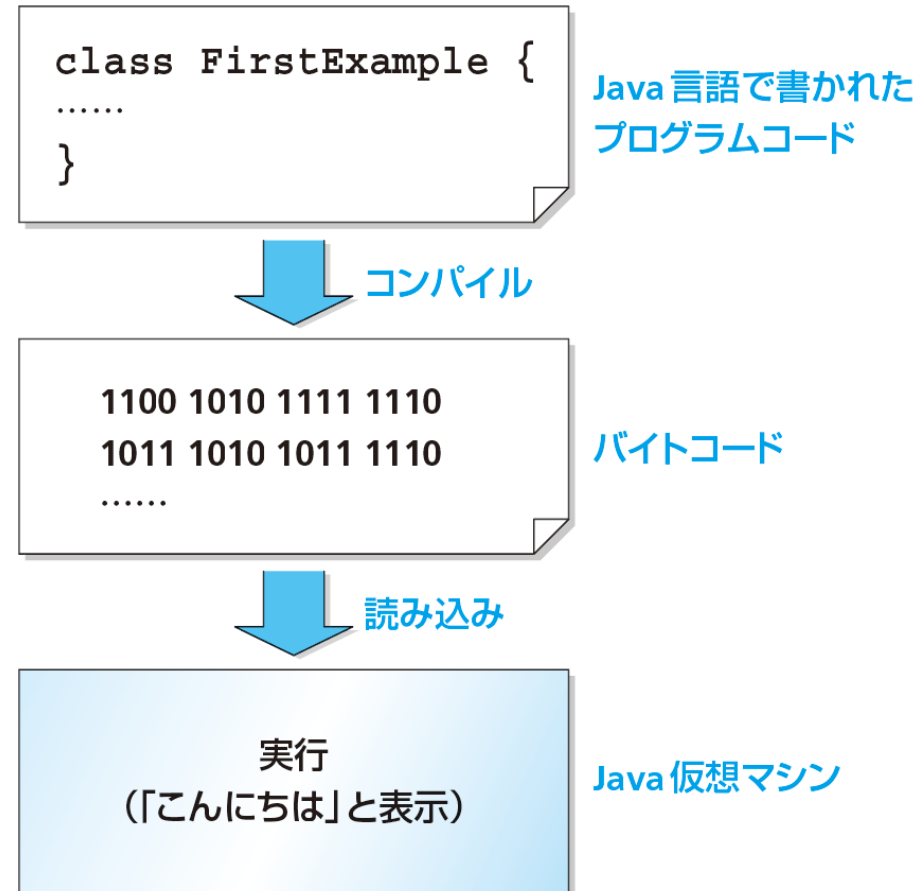
```
class FirstExample {  
    public static void main(String[] args) {  
        System.out.println("こんにちは");  
    }  
}
```

「こんにちは」という文字を画面に表示するプログラムのプログラムコード

半角英数と記号で記述する。人が読んで理解できるテキスト形式。

プログラムコードが実行されるまで

- プログラムコードがコンパイルされてバイトコードが作られる。
- バイトコードがJava仮想マシンによって実行される。



Java言語の特徴

- コンパイラによってバイトコードに変換される。
- バイトコードがJava仮想マシンによって実行されるので、WindowsやMac OS、Linuxなどの各種OS上でコンパイルし直さずに動作する。
- オブジェクト指向型言語。

C/C++言語との違い 1/2 (付録B)

- ポインタが無い
- プリプロセッサ(`#include` `#define` `#if` `#ifdef` など)やマクロが無い
- ヘッダファイルが必要ない
- 多重継承ができない
- 演算子のオーバーロードができない

※ C/C++よりもバグが含まれる可能性を低くできる

C/C++言語との違い 2/2（付録B）

- `boolean`型がある (`true/false`)
- 配列は`new`を使って確保する
例：`int[] a = new int[5];`
- ガーベッジコレクションがある (`delete`の必要が無い)
- 文字列は`String`型で扱う
例：`String str = "Hello";`

※ C/C++よりも型を厳密に扱える

Java言語のプログラム構成

```
class クラス名 {  
    public static void main(String[] args) {  
        命令文  
    }  
}
```

- クラス名は自由に設定できる。頭文字はアルファベットの大文字
例：Example
- `public static void main(String[] args) { }` の{と}の中に命令文を書く。

Java言語のプログラム構成

```
class FirstExample {  
    public static void main(String[] args) {  
        System.out.println("こんにちは");  
    }  
}
```

- 命令文の末尾にはセミコロン(;)をつける
- 空白や改行は好きなところに入れられる
- 大文字と小文字は区別される

ブロックとインデント

```
class FirstExample {  
  (インデント) public static void main(String[] args) {  
    (インデント) System.out.println("こんにちは");  
  (インデント) }  
}
```

- { と } は必ず1対1の対応を持っている
- { } で囲まれた範囲を「ブロック」と呼ぶ
- プログラムコードを見やすくするための先頭の空白を「インデント」と呼ぶ

コメント文

```
/*  
    こんにちはという文字を画面に表示するプログラム  
    作成日：2010年12月1日  
*/  
class FirstExample {  
    public static void main(String[] args) {  
        // 画面に文字を表示する  
        System.out.println("こんにちは");  
    }  
}
```

- プログラムコードの中のメモ書きを「コメント」と呼ぶ
- 方法1 /* と */ で囲んだ範囲をコメントにする
- 方法2 // をつけて、1行だけコメントにする

プログラムの作成

- 方法1

コマンドラインでコンパイルして実行する

> javac FirstExample.java

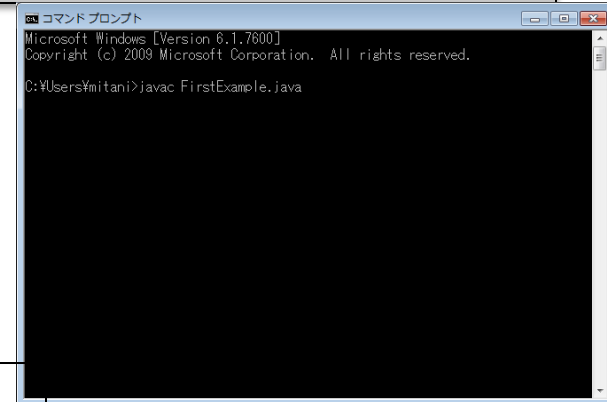
←コンパイル

> java FirstExample

←実行

こんにちは

←実行結果

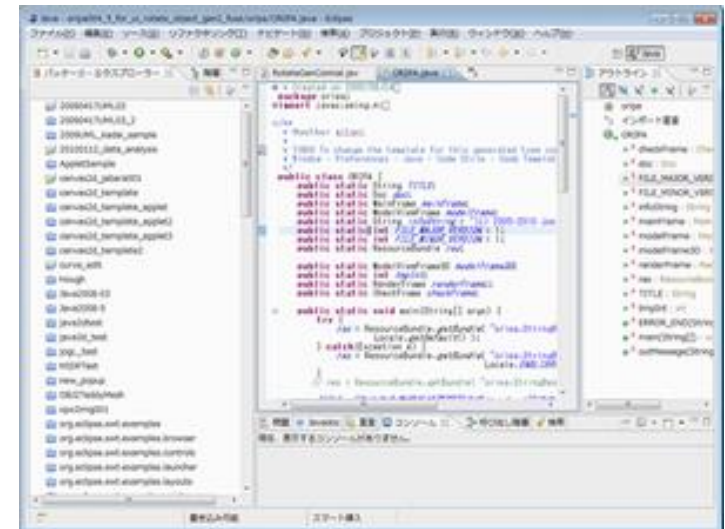


```
コマンドプロンプト
Microsoft Windows [Version 6.1.7600]
Copyright (c) 2009 Microsoft Corporation. All rights reserved.

C:\Users\mitani>javac FirstExample.java
```

- 方法2

Eclipseなどの統合開発環境を使用する



Eclipseでの実行の手順

1. プロジェクトを作成する
（[ファイル]→[新規]→[Javaプロジェクト]）
2. プログラムコードを作成する
（[ファイル]→[新規]→[クラス]）
3. プログラムの実行
（[実行]→[実行]→[Javaアプリケーション]）

エラー (Compile Error) が起きたら

- キーワードの綴りミス、文法上の誤りが原因。
- 単純なミスに気を付ける
 - 全角の文字、空白を使用しない
 - 似た文字の入力間違い
 - ゼロ(0)、小文字のオー(o)、大文字のオー(O)
 - イチ(1)、大文字のアイ(I)、小文字のエール(l)
 - セミコロン(;), コンマ(:)
 - ピリオド(.), カンマ(,)

.javaファイルと.classファイル

- プログラムコードは拡張子が.javaのファイルに保存する
例：FirstExample.java
- プログラムコードをコンパイルすると拡張子が.classのファイルが生成される
例：FirstExample.class
- Eclipseでは、最初に指定したworkspaceフォルダの中に自動生成される

演習

1. Java言語の歴史についてインターネットで調べてみる
2. 実際にJavaプログラムが使用されているシステムにはどのようなものがあるかインターネットで調べてみる

上記をレポートにまとめて提出してください
参考文献の出典も明記すること
ファイル名：Java1_学籍番号.docx

提出方法：USBメモリにコピーしてください

演習

1. `FirstExample.java` を入力し、実際に動かしてみる。
2. `.java`ファイルと`.class`ファイルがどこにあるか確認してみる。

第2章 Java言語の基本

出力

- 文字列を標準出力（Eclipseの場合はコンソールビュー）に出力する命令

```
System.out.println(出力する内容);
```

実際のコード

```
class FirstExample {  
    public static void main(String[] args) {  
        System.out.println("こんにちは");  
    }  
}
```

エスケープシーケンス

- 特別な記号や出力方法を制御するために
¥記号を使う

例：`System.out.println(`
 `"これから¥"Java言語¥"を学習します。");`

記号の組み合わせ	意味
¥'	'
¥"	"
¥¥	¥
¥n	改行 (ラインフィード: LF)
¥t	水平タブ
¥r	復帰 (キャリッジリターン: CR)

演習

1. 自分の名前を出力する
2. 複数の`System.out.println`の命令文を記して、実行結果を確認する
3. 「これから"Java言語"の学習をします」と出力する

変数

- 「変数」とは、値を入れておく入れ物。

```
int i; // 変数の宣言  
i = 5; // 値の代入  
System.out.println(i); // 値の参照
```

- 変数の宣言：変数を作成すること
- 値の代入：変数に値を入れること
- 値の参照：変数に入っている値を見ること

変数の使用

命令文

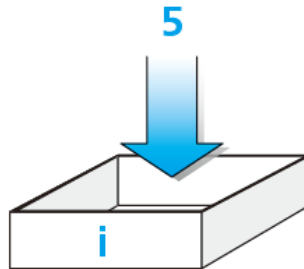
意味

```
int i;
```



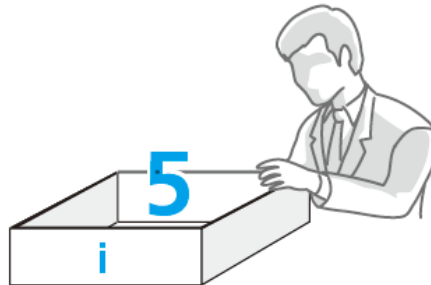
i という名前の入れ物を作成します。

```
i = 5;
```



i という名前の入れ物に5を入れます。

```
System.out.println(i);
```



i という名前の入れ物の中身を見て、その中身をコンソールに出力します。

変数の宣言と型

- 変数の宣言では、変数に入れる値のタイプ（型）をはじめに指定する。

型名 変数名;

例1 `int i;`

例2 `double d;`

例3 `boolean boo = false;`

例4 `char c = 'あ';`

Javaで利用できる型

型	値の例	格納できる値の範囲
char	'a', 'b', 'c', ... 'あ', 'い', ...	1文字 (16ビット、Unicode文字)
boolean	true, false	真偽値。true (真) または false (偽) のどちらか
byte		8ビット符号付き整数。 -2^7 (-128) $\sim 2^7-1$ (127)
short		16ビット符号付き整数。 -2^{15} (-32,768) $\sim 2^{15}-1$ (32,767)
int	整数 (... , -1, 0, 1, ...)	32ビット符号付き整数。 -2^{31} (-2,147,483,648) $\sim 2^{31}-1$ (2,147,483,647)
long		64ビット符号付き整数。 -2^{63} (-9,223,372,036,854,775,808) $\sim 2^{63}-1$ (9,223,372,036,854,775,807)
float	小数点を含む数値	32ビット符号付き浮動小数点数 (注②-8)
double	(... -0.5, ..., 0.0, ..., 0.5, ...)	64ビット符号付き浮動小数点数

演習

1. 次のプログラムコードの赤字部分を様々に変更して実行してみましょう。

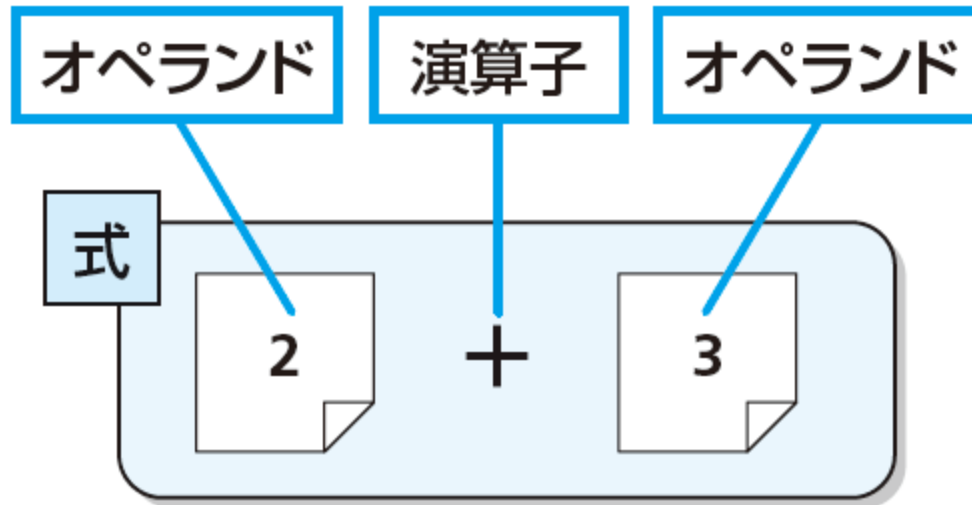
例：double型、boolean型、char型

```
class Example {  
    public static void main(String args[]) {  
        int i;  
        i = 5;  
        System.out.println(i);  
    }  
}
```

算術演算子と式

- 算術演算子を用いた計算

```
System.out.println(2 + 3);
```



式の値は5

算術演算子と優先順位

演算子	演算の内容	説明	使用例
+	加算（足し算）	左辺と右辺を足します	1 + 2（式の値は3）
-	減算（引き算）	左辺から右辺を引きます	2 - 1（式の値は1）
*	乗算（掛け算）	左辺と右辺を掛けます	2 * 3（式の値は6）
/	除算（割り算）	左辺を右辺で割ります	4 / 2（式の値は2）
%	剰余	左辺を右辺で割った余りを求めます	7 % 3（式の値は1）

- 数学と同じように、加算と減算（+, -）より乗算と除算（*, /）が優先される

```
System.out.println(3 + 6 / 3); // 5
System.out.println((3 + 6) / 3); // 3
```

演習

- 次のプログラムコードの赤字部分を変更して、様々な計算を試みましょう。
例：加算、減算、乗算、除算、剰余

```
class Example {  
    public static void main(String args[]) {  
        System.out.println(2 + 3);  
    }  
}
```

変数を含む算術演算子

```
int i = 10;  
int j = i * 2;  
System.out.println(j); // 20
```

```
int i = 10;  
i = i + 3;  
System.out.println(i); // 13
```

```
int i = 10;  
i += 3; // 短縮表現  
System.out.println(i); // 13
```

演算子	演算の内容	説明	使用例
<code>+=</code>	加算代入	左辺の変数の値と右辺の値を足した値を、左辺の変数に代入します	<code>a += 2</code> (変数 <code>a</code> の値は2 増えます。 <code>a = a + 2</code> と同じです)
<code>-=</code>	減算代入	左辺の変数の値から右辺の値を引いた値を、左辺の変数に代入します	<code>a -= 3</code> (変数 <code>a</code> の値は3 減ります。 <code>a = a - 3</code> と同じです)
<code>*=</code>	乗算代入	左辺の変数の値に右辺の値を掛けた値を、左辺の変数に代入します	<code>a *= 2</code> (変数 <code>a</code> の値は2 倍になります。 <code>a = a * 2</code> と同じです)
<code>/=</code>	除算代入	左辺の変数の値を右辺の値で割った値を、左辺の変数に代入します	<code>a /= 3</code> (変数 <code>a</code> の値は3 分の1 になります。 <code>a = a / 3</code> と同じです)
<code>%=</code>	剰余代入	左辺の変数の値を右辺の値で割った余りを、左辺の変数に代入します	<code>a %= 2</code> (変数 <code>a</code> の値はそれを2 で割った余りになります。 <code>a = a % 2</code> と同じです)
<code>++</code>	インクリメント	左辺の変数の値を1 増やします	<code>a++</code> (変数 <code>a</code> の値は1 増えます。 <code>a = a + 1</code> と同じです)
<code>--</code>	デクリメント	左辺の変数の値を1 減らします	<code>a--</code> (変数 <code>a</code> の値は1 減ります。 <code>a = a - 1</code> と同じです)

演習

次の命令文を短い表現に書き換えましょう

1. $a = a + 5;$

2. $b = b - 6;$

3. $c = c * a;$

4. $d = d / 3;$

5. $e = e \% 2;$

6. $f = f + 1;$

7. $g = g - 1;$

演習

次のプログラムコードの実行結果を予測し、確認しましょう

```
class CalcExample3 {  
    public static void main(String[] args) {  
        int i;  
        i = 11;  
        i++;  
        i /= 2;  
        System.out.println("iの値は" + i);  
  
        int j;  
        j = i * i;  
        System.out.println("jの値は" + j);  
    }  
}
```


ワン・モア・ステップ（文と式）

- 「`i = 2 + 3;`」は文
- 「`i = 2 + 3`」は式（代入式）
- 式は値を持つ
- 代入式は左辺に代入される値を持つ

```
int i;  
int j = (i = 2 + 3) * 2;  
System.out.println(i); // 5  
System.out.println(j); // 10
```

ワン・モア・ステップ（前置と後置）

- 後置「`i++;`」
- 前置「`++i;`」
- どちらも`i`の値を1だけ増やす
- 「`j=i++;`」と「`j=++i;`」では`j`の値が異なる。



```
j = i;  
i = i + 1;
```



```
i = i + 1;  
j = i;
```

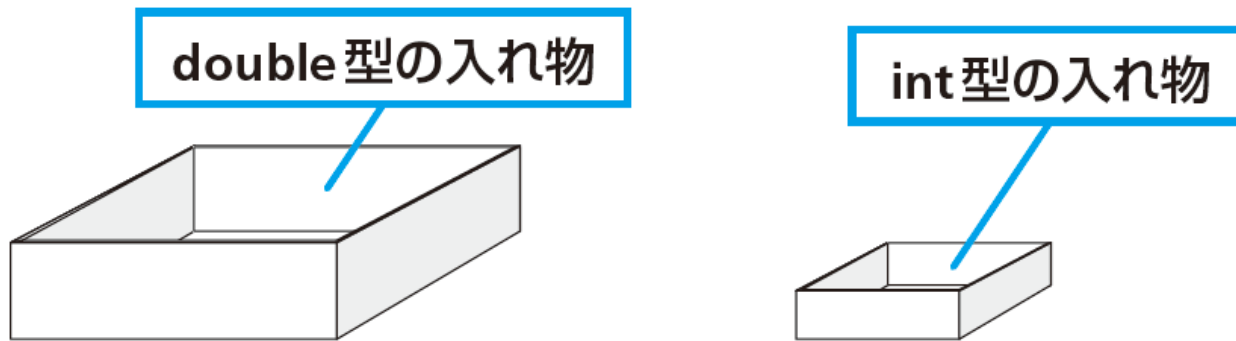
演習

次のプログラムコードの実行結果を予測し、確認しましょう

```
int i = 10;  
int j = i++;  
int k = ++i;  
System.out.println(i);  
System.out.println(j);  
System.out.println(k);
```

型と大きさ

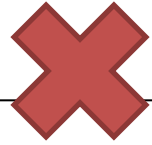
- 型によって変数の大きさが異なる



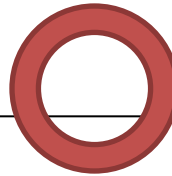
`double > int`

型変換

```
double d = 9.8;  
int i = d;
```



```
double d = 9.8;  
int i = (int)d;
```



- 大きな変数(double)の値を小さな変数(int)に代入できない
- カッコを使って型変換できる。
- 型変換を「キャスト」とも呼ぶ。

異なる型を含む演算

```
int i = 5;  
double d = 0.5;  
System.out.println(i + d); // 5.5
```

- 型の異なる変数や値の間で演算を行った場合は、最も大きい型（上の例では `double` 型）に統一されて計算される。

整数同士の割り算

```
int a = 5;  
int b = 2;  
double c = a / b;  
System.out.println(c); // 2.0
```

- 整数と整数の割り算は整数型として扱われる。
上の例では $5/2$ が 2 になる。
- 正しい値を求めるには、`double`型にキャストする必要がある。

例：`double c = (double)a/(double)b;`

演習

7÷2の計算結果が正しく3.5になるように修正しましょう。

```
class Example {  
    public static void main(String[] args) {  
        int a = 7;  
        int b = 2;  
        double d = a / b;  
        System.out.println(d);  
    }  
}
```


String 型

- 文字列はString型の変数に代入できる。

```
String s;  
s = "こんにちは";  
System.out.println(s);
```

- 文字列は「+」演算子で連結できる。

```
String s1 = "こんにちは。";  
String s2 = "今日はよい天気ですね。";  
String s3 = s1 + s2;  
System.out.println(s3);
```