実行時のメモリ構造(1)
Jasminの基礎とフレーム内動作

2002年5月19日

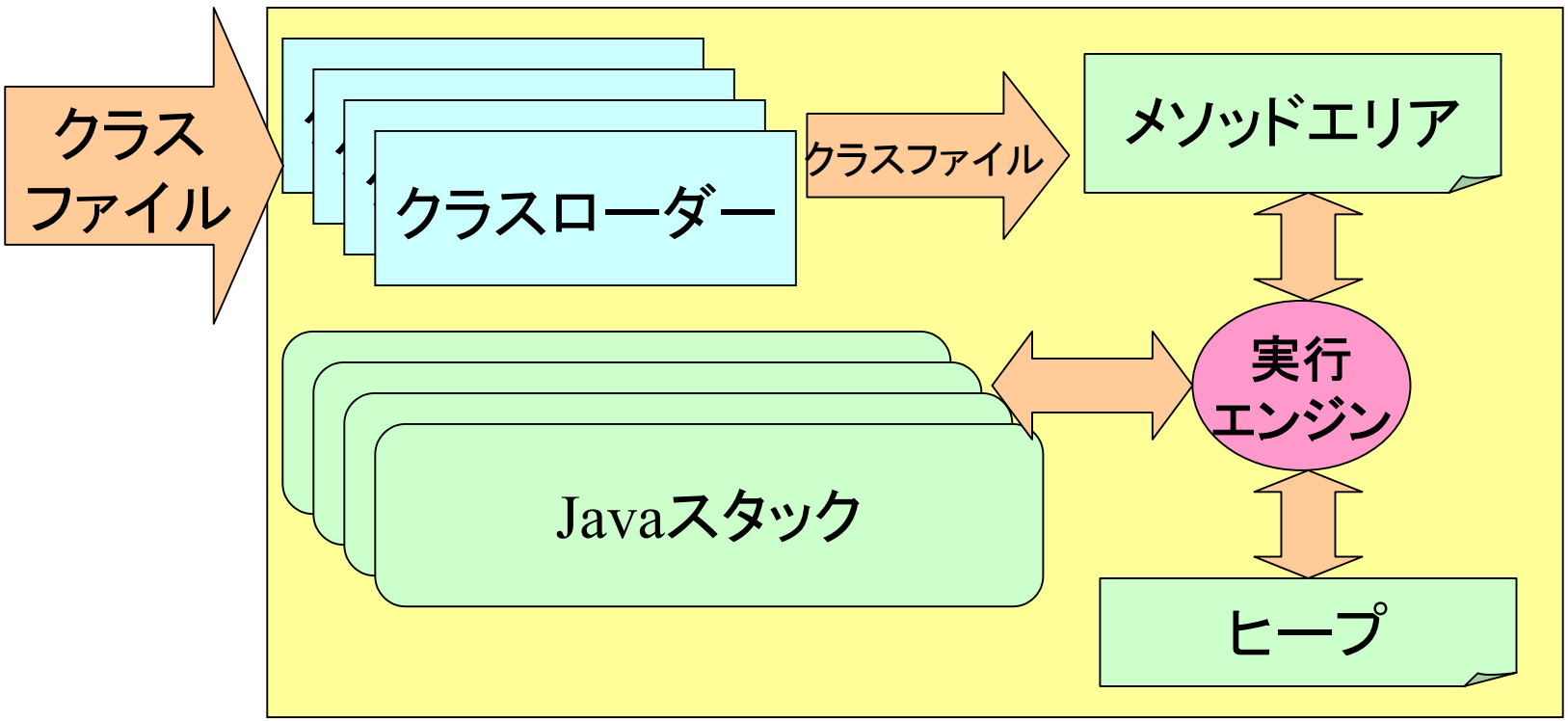
海谷 治彦

5/20改訂版

JVM内の基本構造(大雑把)

クラスファイルの
内容チェック

クラスデータを保存



各実行スレッドのローカルデータ
(実行経過)を保存

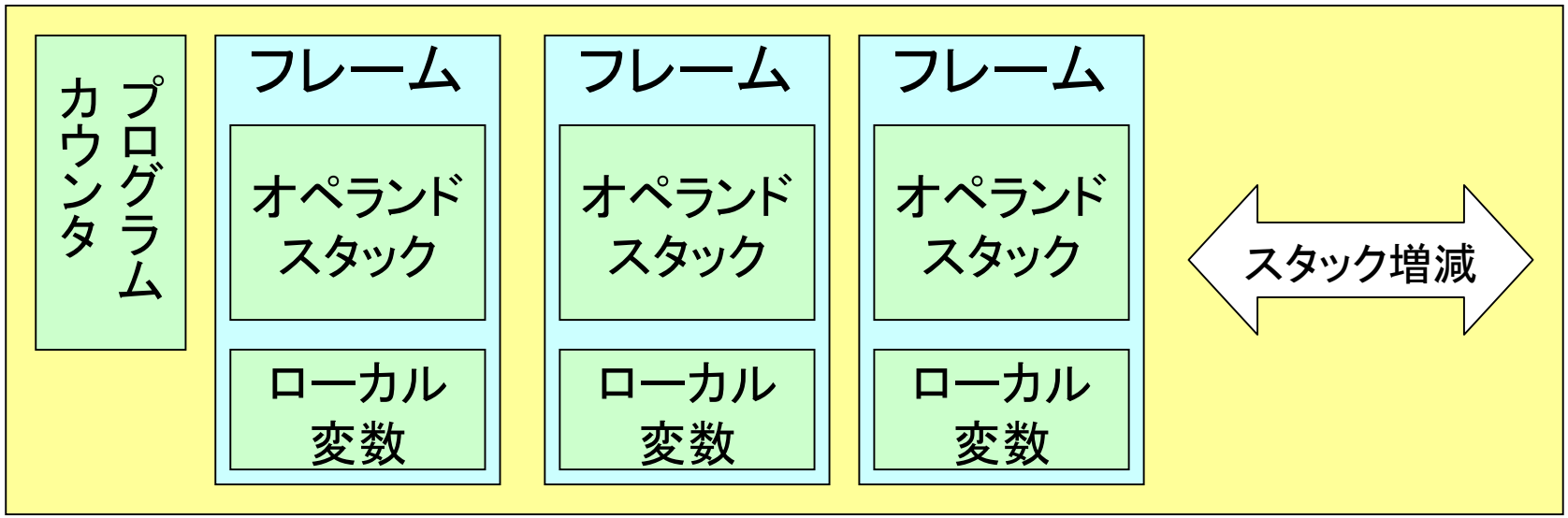
インスタンスデータ
を保存

教科書 p.15

* 原著および教科書p.15をベースに書いた.

リンク有

Javaスタック内の構造



- 「フレーム」という要素のスタック.
- フレームは，1回のメソッド呼び出しに対応.
- フレーム内の計算のためにも，スタック（オペランドスタック）が利用されている.
- 詳細は「実行時の構造」の回にて.

例えば教科書 p.20の図

本日のお題

- JVMのアセンブラ Jasminの概要を学ぶ
  - 実行時の動作を追跡するのに必要な概念
- フレーム内の動作を学ぶ
  - ーメソッド呼び出し内での計算を理解.

言語の高級･低級

- 一般にマシンが直接解釈できる言語を低級言語，人に読みやすい言語を高級言語と呼ぶ.
- Java回りの場合，
  - Java
  - Jasmin や Oolong などのアセンブラ言語
  - Javaバイトコード

  の順で低レベル化する.

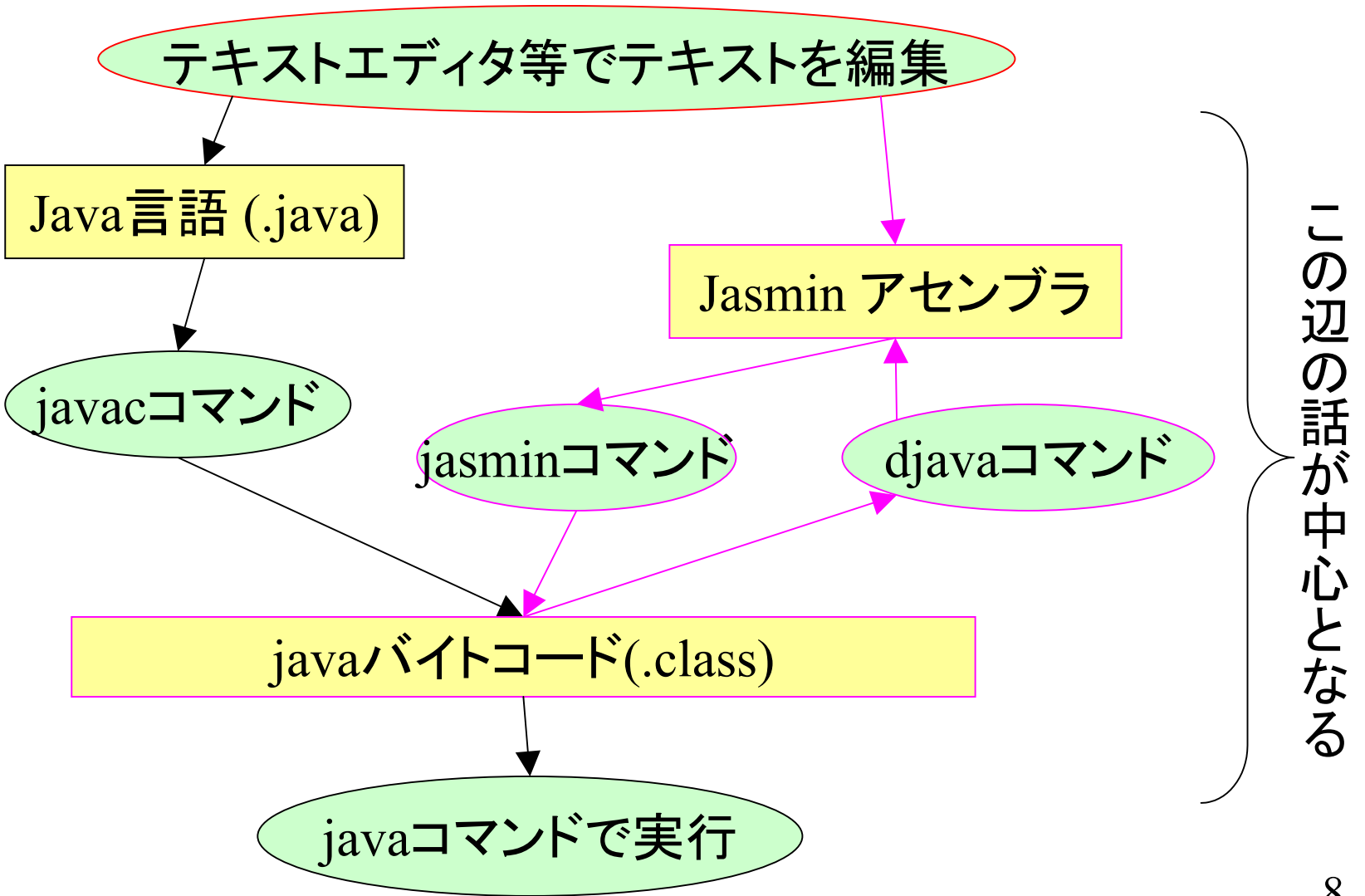
アセンブラ言語

- ほぼマシン語(ここではバイトコード)と一対一対応している多少人に読みやすい形式のプログラム記述形式.
- マシン語にある命令やアドレス，ラベル等を人の分かりやすいキーワードに置き換えているだけ.
- 本授業では，Jasminというアセンブラ言語を使う.

処理系と言語の関係

- 言語: プログラミング言語の種類のこと
  - ソースコード: 主に高級言語で書いたプログラムのことを言う.
  - アセンブラ: アセンブラ言語で書いたプログラム.
- 処理系: 言語処理系のこと.
  - コンパイラ: ソースコードからマシン語への変換プログラム.
  - アセンブラ: アセンブラからマシン語への変換プログラム

処理と言語の関係図


テキストエディタ等でテキストを編集
Java言語 (.java)
Jasmin アセンブラ
javacコマンド
jasminコマンド
djavaコマンド
javaバイトコード(.class)
javaコマンドで実行
この辺の話が中心となる

djavaコマンド

- 逆アセンブルツール: すなわちマシン語からアセンブラに変換するプログラム.
- 実体はjavaで書かれたプログラム
- 典型的な使い方

  `djava なんとか.class`

  で標準出力にjasminアセンブラを出力.
- 教科書p.7付近参照. (ちょっと違う)
- 実演 (Iadd.java)

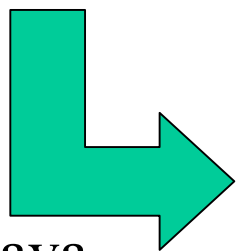
javaとjasminの関係

- メソッド・属性ほぼ一対一対応している.
- jasminでは省略したデフォルト・コンストラクタの記述も出てくる. (ここは対応してない.)
- jasminでも型がある, すなわち, バイトコードレベルでの型がある. (p.65)

最も簡単なjasminの例

javaソース(一部)

```
int add(int a, int b){
    return a+b;
}
```



djava

jasminソース(一部)

```
.method add(II)I
    .limit stack 2
    .limit locals 3
    iload_1
    iload_2
    iadd
    ireturn
.end method
```

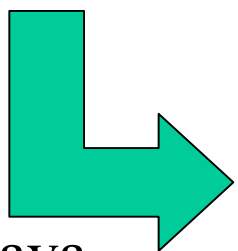
全体比較は,
iadd/cmp.html を参照

本日はフレーム内計算しかしないので,
他の部分には言及しない

良く似たjasminの例

javaソース(一部)

```
int add2(int a, int b){
  int c;
     c=a+b;
     return c;
  }
```



djava

jasminソース(一部)

```
.method add2(II)I
    .limit stack 2
    .limit locals 4
    iload_1
    iload_2
    iadd
    istore_3
    iload_3
    ireturn
.end method
```

jasminメソッドの基本構造(例)

```
.method add(II)I
    .limit stack 2
    .limit locals 3
    iload_1
    iload_2
    iadd
    ireturn
.end method
```

- 青字部分は予約語
- フレーム内のオペランドスタックの長さを指定.
- 同ローカル変数のサイズを指定.
- 残り4行が命令文

jasminメソッドの引数(例)

```
.method add(II)I
    .limit stack 2
    .limit locals 3
    iload_1
    iload_2
    iadd
    ireturn
.end method
```

- (II)Iは引数と返り値の型を規定.
  - II 整数が二つ
  - I 整数が1つ

```
int add(int a, int b){
    return a+b;
}
```

jasminの基本型+α

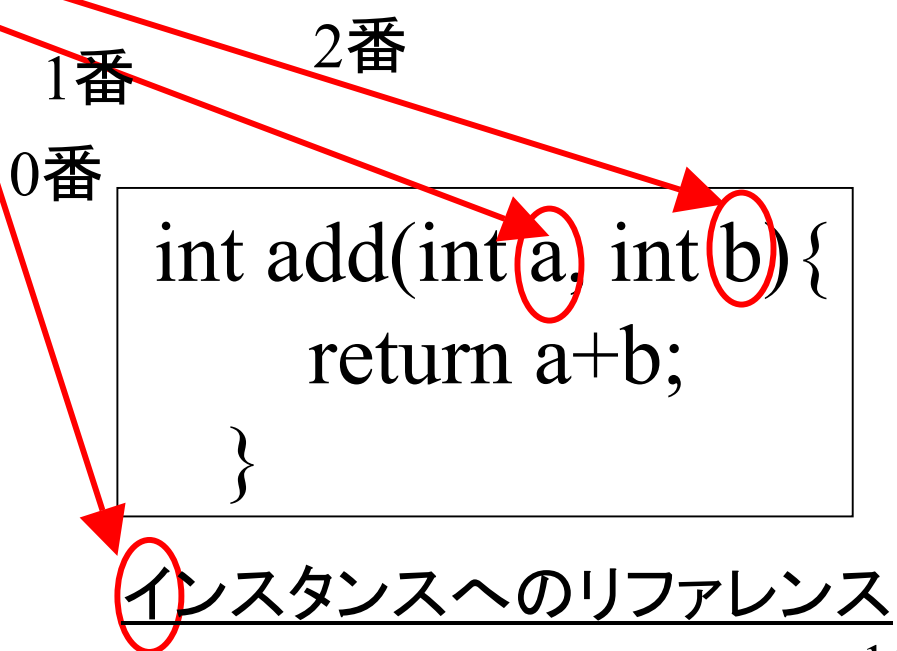
ディスクリプタ	Java言語の型
B	byte
C	char
D	double
F	float
I	int
J	long
S	short
Z	boolean
V	(void)

教科書 p.65 p.216 参照

ローカル変数の対応

```
.method add(II)I
    .limit stack 2
    .limit locals 3
    iload_1
    iload_2
    iadd
    ireturn
.end method
```

- 3つあるという意味.
- 0から順に番号付けされている.



他の例のローカル変数の対応

```
.method add2(II)I
    .limit stack 2
    .limit locals 4
    iload_1
    iload_2
    iadd
    istore_3
    iload_3
    ireturn
.end method
```

```
int add2(int a, int b){
    int c;
        c=a+b;
        return c;
    }
```

0番 インスタンスへのリファレンス

1番

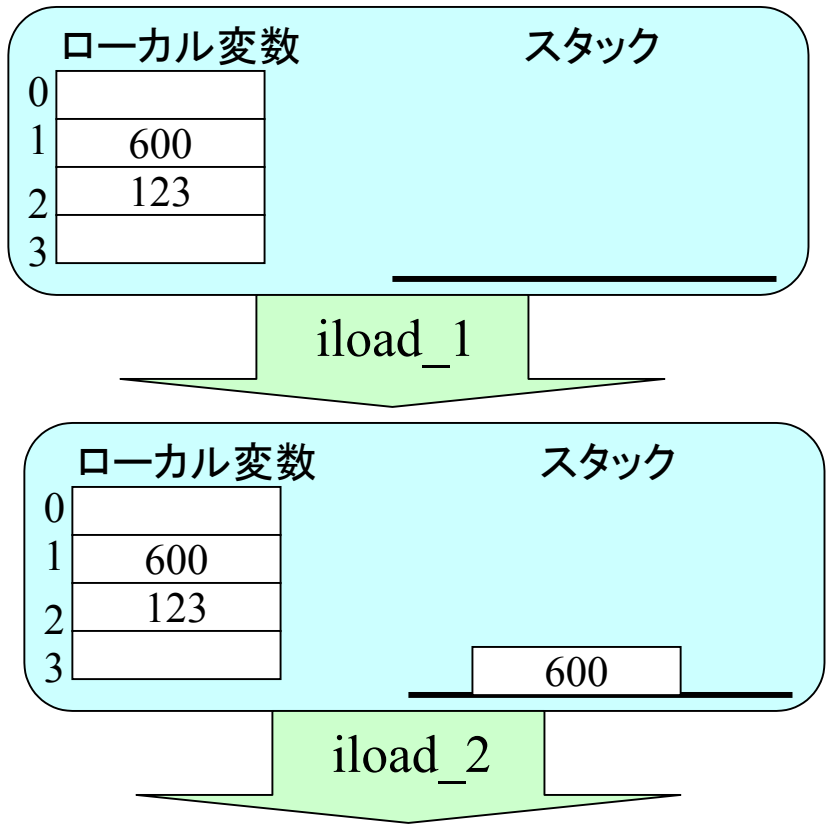
2番

3番

フレーム内計算の例 1/3

add2(600, 123) を呼ぶ場合.

```
.method add2(II)I
    .limit stack 2
    .limit locals 4
    iload_1
    iload_2
    iadd
    istore_3
    iload_3
    ireturn
.end method
```

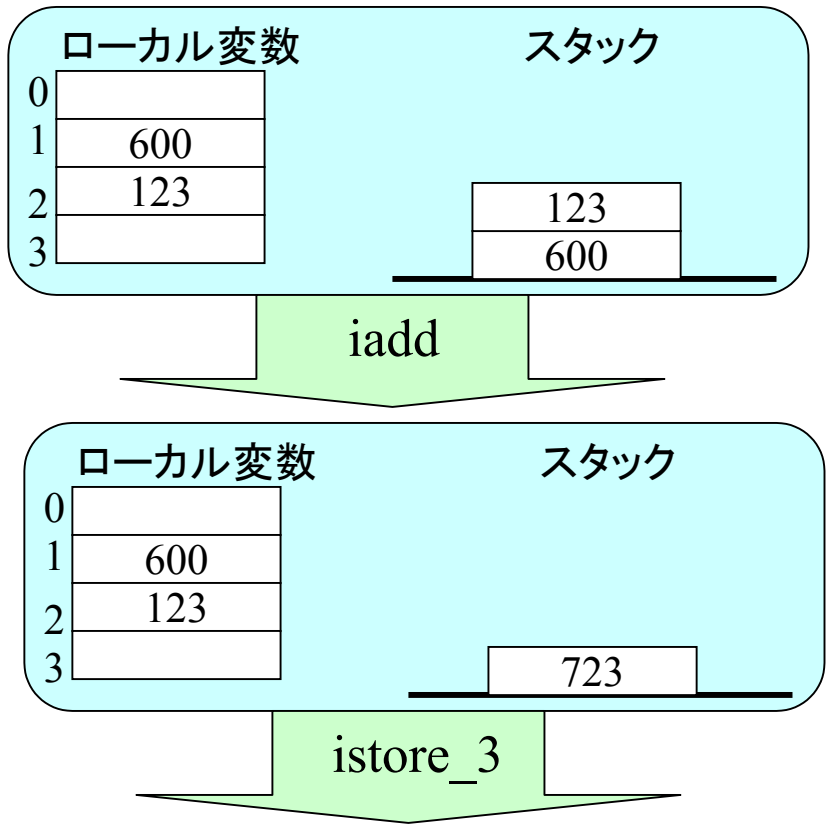


教科書p.29とほぼ同じ

フレーム内計算の例 2/3

add2(600, 123) を呼ぶ場合.

```
.method add2(II)I
    .limit stack 2
    .limit locals 4
    iload_1
    iload_2
    iadd
    istore_3
    iload_3
    ireturn
.end method
```

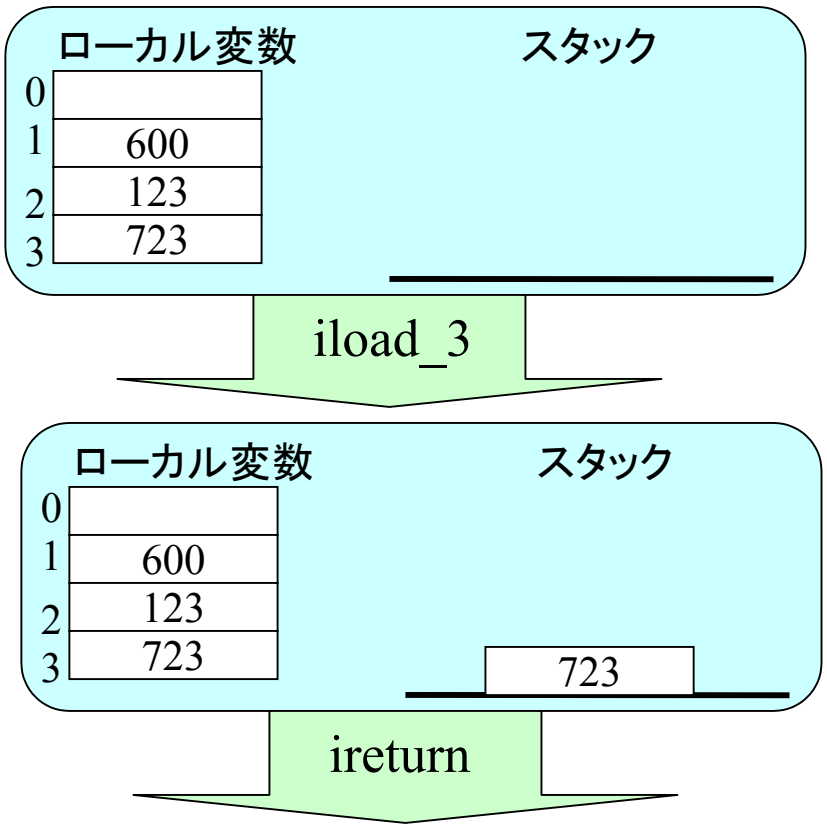


教科書p.29とほぼ同じ

フレーム内計算の例 3/3

add2(600, 123) を呼ぶ場合

```
.method add2(II)I
    .limit stack 2
    .limit locals 4
    iload_1
    iload_2
    iadd
    istore_3
    iload_3
    ireturn
.end method
```

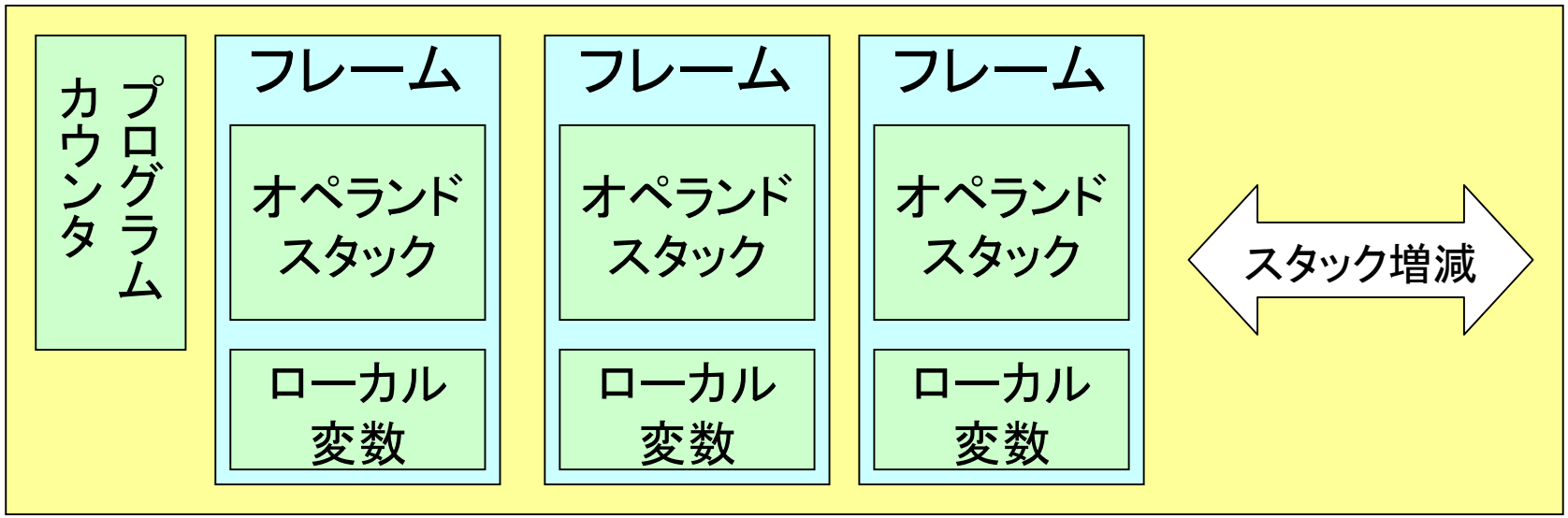


(フレーム自体が消滅，来週の話)

教科書p.29とほぼ同じ

再掲載

Javaスタック内の構造



- 「フレーム」という要素のスタック.
- フレームは，1回のメソッド呼び出しに対応.
- フレーム内の計算のためにも，スタック（オペランドスタック）が利用されている.
- 詳細は「実行時の構造」の回にて.

例えば教科書 p.20の図

フレーム内計算のテキスト表現

add2(600, 123) を呼ぶ場合

```
.method add2(II)I
    .limit stack 2
    .limit locals 4
    iload_1
    iload_2
    iadd
    istore_3
    iload_3
    ireturn
.end method
```

```
local=[this, 600, 123,      ] stack=[]
iload_1
local=[this, 600, 123,      ] stack=[600]
iload_2
local=[this, 600, 123,      ] stack=[600, 123]
iadd
local=[this, 600, 123,      ] stack=[723]
istore_3
local=[this, 600, 123, 723] stack=[]
iload_3
local=[this, 600, 123, 723] stack=[723]
ireturn
```

stack長は必要最低限である

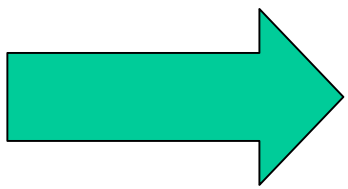
add2(600, 123) を呼ぶ場合

```
.method add2(II)I
    .limit stack 2
    .limit locals 4
    iload_1
    iload_2
    iadd
    istore_3
    iload_3
    ireturn
.end method
```

```
local=[this, 600, 123,      ] stack=[]
iload_1
local=[this, 600, 123,      ] stack=[600]
iload_2
local=[this, 600, 123,      ] stack=[600, 123]
iadd
local=[this, 600, 123,      ] stack=[723]
istore_3
local=[this, 600, 123, 723] stack=[]
iload_3
local=[this, 600, 123, 723] stack=[723]
ireturn
```

if文の展開例

```
int larger(int a, int b){
  int c;③      ①      ②
    if(a>b) c=a;
    else c=b;
    return c;
  }
```

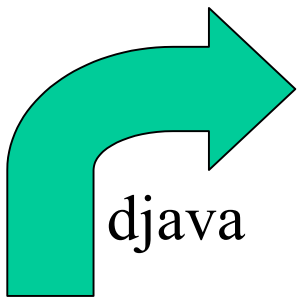


```
.method larger(II)I
    .limit stack 2
    .limit locals 4
    iload_1
    iload_2
    if_icmple Label1
    iload_1
    istore_3
    goto Label2
Label1:
    iload_2
    istore_3
Label2:
    iload_3
    ireturn
.end method
```

フレーム内の計算過程は，ifelse/exe.html を参照.

ループの展開の例

単に0からn-1まで
加算するルーチン




```
int forLoop(int n){
  int s=0;
     for(int c=0; c<n; c++)
          s+=c;
     return s;
  }
```

①
②
③

```
.method forLoop(I)I
    .limit stack 2
    .limit locals 4
    iconst_0
    istore_2
    iconst_0
    istore_3
    goto Label2
```

```
Label1:
    iload_2
    iload_3
    iadd
    istore_2
    iinc 3 1
Label2:
    iload_3
    iload_1
    if_icmplt Label1
    iload_2
    ireturn
.end method
```

ループの実行例

- 長いのでwebページ参照
  - loop/exe.html
- 実はforもwhileもほとんど同じ形に展開される
  - loop/cmp.html