ソフトウェアアーキテクチャ
第7回　JAVA仮想機械

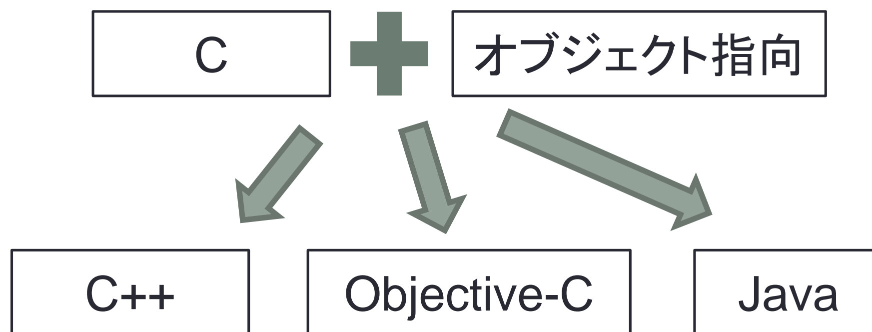
環境情報学部
萩野　達也

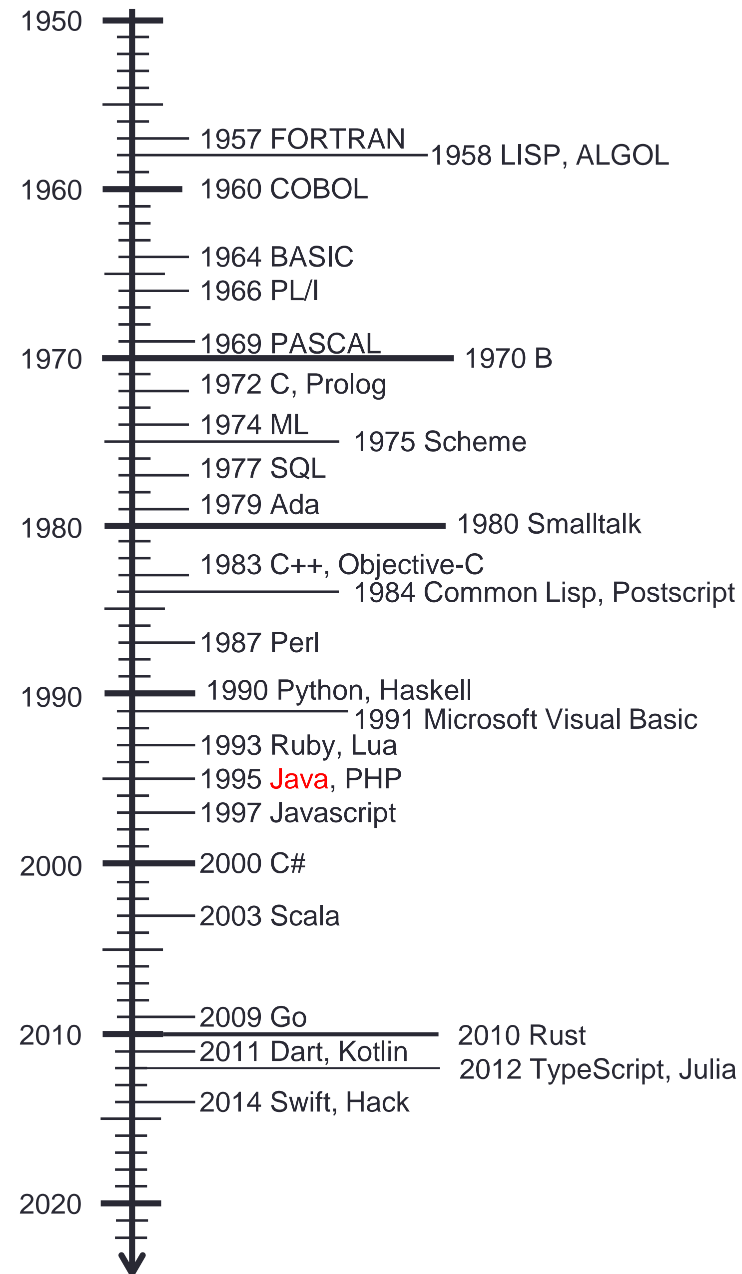
lecture URL

https://vu5.sfc.keio.ac.jp/slide/

Javaとは

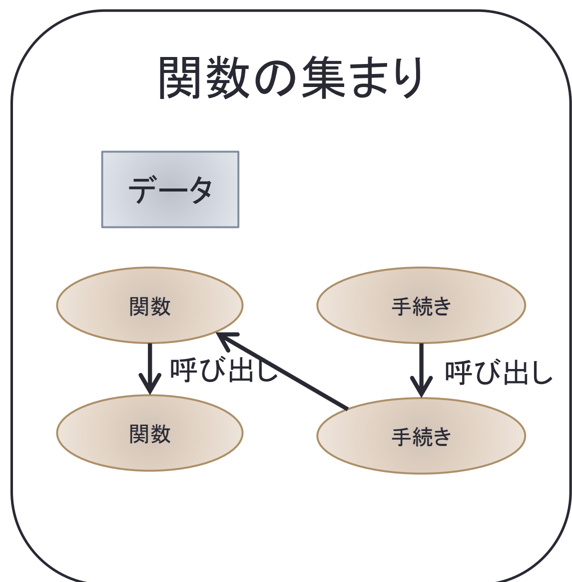
- Java 言語
  - 1995年に登場
  - 色々な機器に組み込むための言語として開発
  - オブジェクト指向言語
  - 構文はC言語に類似
    - goto文はない

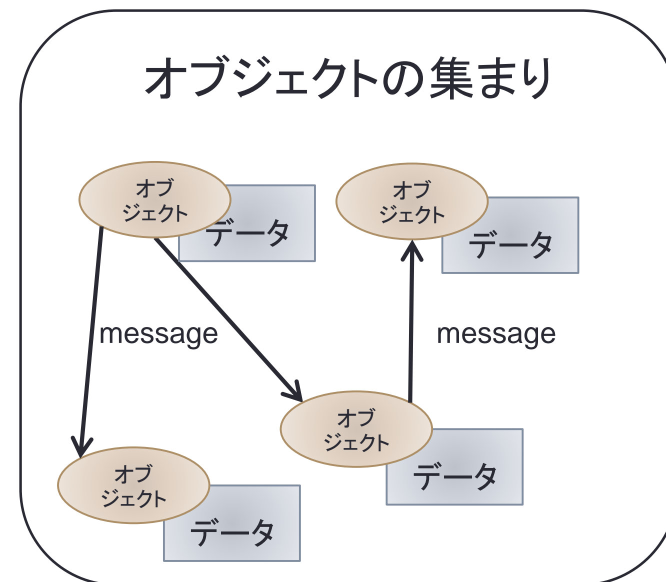




オブジェクト指向プログラミング言語とは

- オブジェクトの集まりでソフトウェアを構成する
  - オブジェクトは互いにメッセージを送りあう



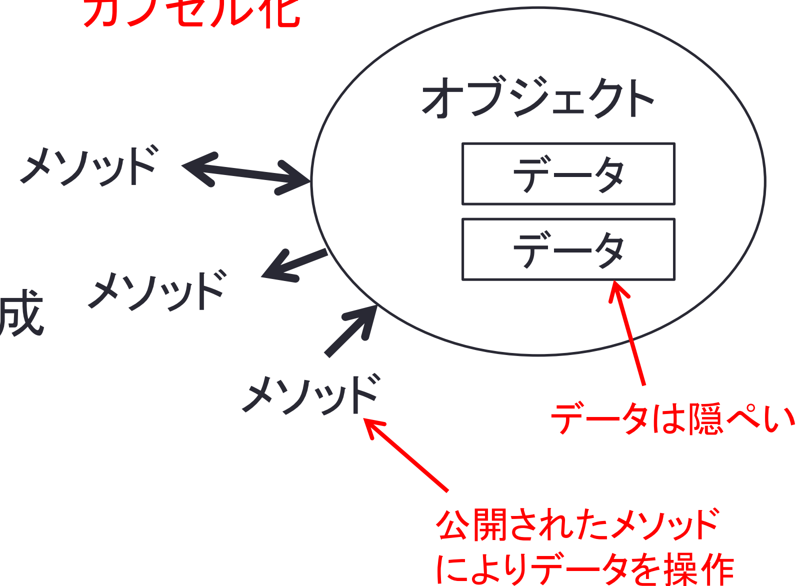


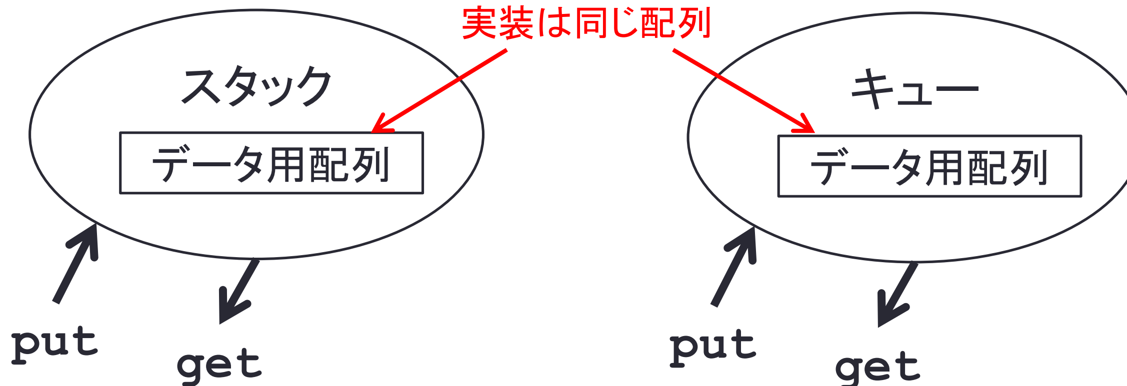
非オブジェクト指向
- 関数とデータは独立
- 複雑な処理を別関数にお願いする
- 関数・手続き中心の考え

オブジェクト指向
- オブジェクトがデータを持つ
- オブジェクトのデータの処理はメッセージで行う
- データ中心の考え

オブジェクト指向の特徴(1)

- カプセル化
  - データをオブジェクト内にカプセル化
  - メソッドによりデータにアクセス
  - オブジェクトはデータとメソッドから構成される
  - 抽象データ型に由来



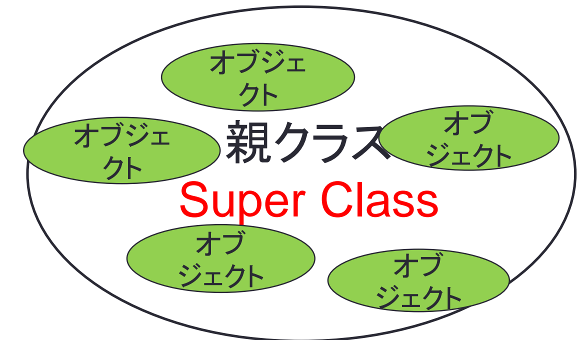


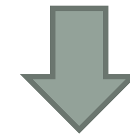
最後にputしたものがgetできる
LIFO = Last In First Out

最初にputしたものがgetできる
FIFO = First In First Out

オブジェクト指向の特徴(2)

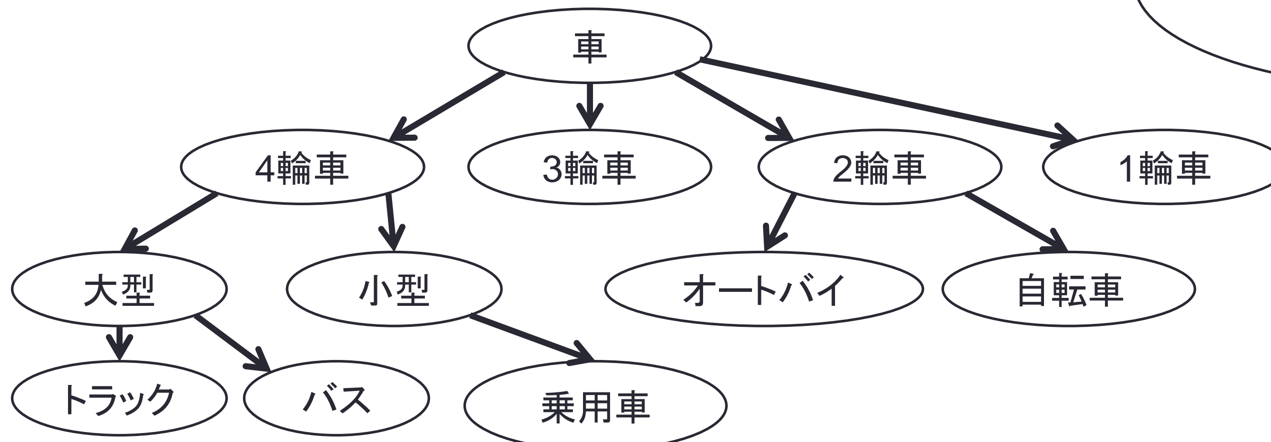
- クラスと継承
  - オブジェクトはクラスに属する
  - オブジェクトはクラスのインスタンス
  - クラスには親子関係があり，子は親を継承
    - 親のメソッドは子のメソッド
    - 親のデータは子のデータ





継承

- クラス階層
  - クラスの親子関係で階層ができる



Javaプログラムの例

クラス名
変数
オブジェクトを作るコンストラクタ

```
public class Human {
  private String name;
  public Human(String name) {
    this.name = name;
  }
  public String getName() {
    return name;
  }
}

public class Student extends Human {
  private int id;
  public Student(int id, String name) {
    this.id = id;
    Human(name);
  }
  public int getID() {
    return id;
  }
  public static void main(String[] args) {
      Student s = new Student(12345, "Hagino");
      System.out.println(s.getName());
    }
}
```

自身のオブジェクトを参照
メソッド
親クラスを指定
メソッド呼び出し
newで新しいオブジェクトを作る

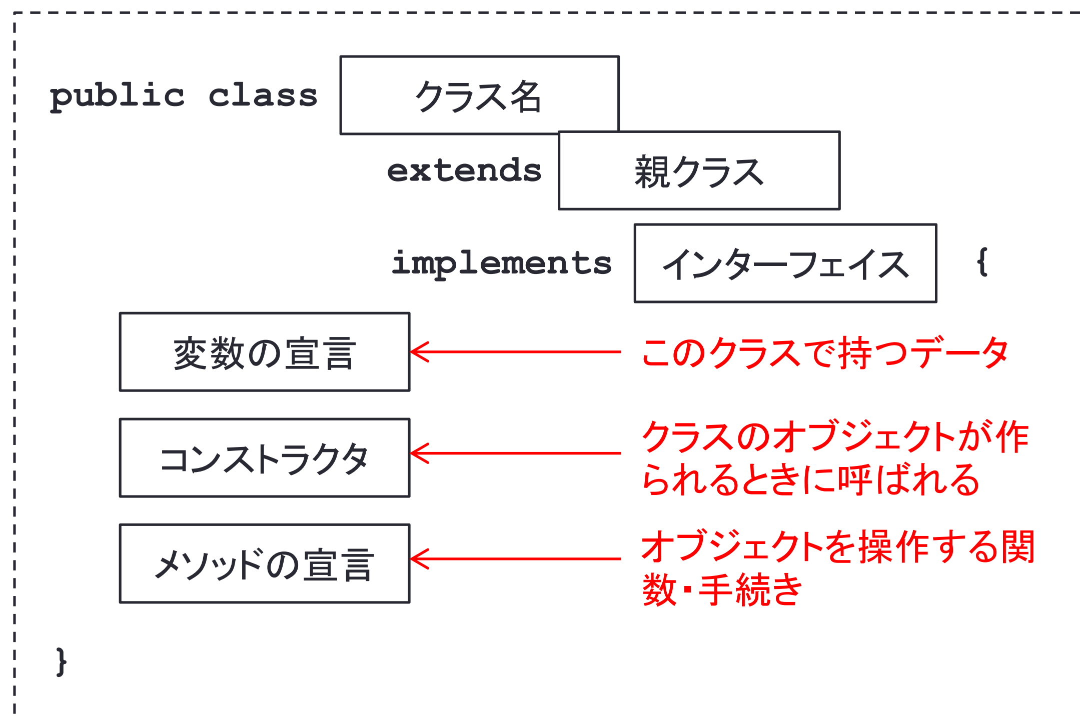
Human
- 人のクラス
- 名前をデータとして保持

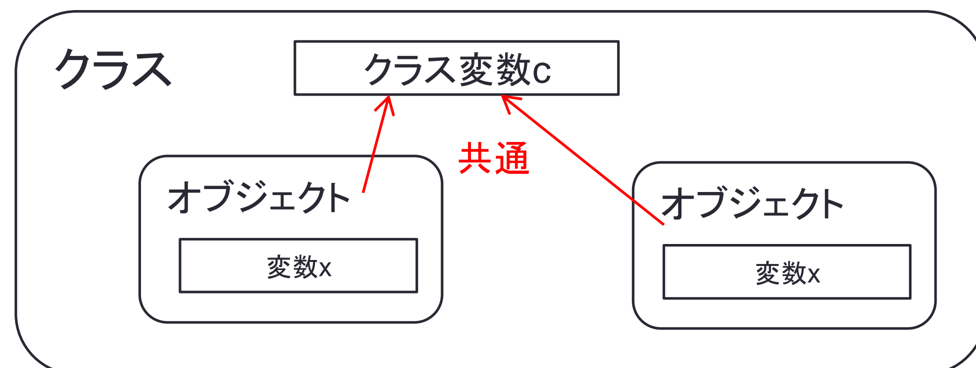
Student
- 学生のクラス
- Humanのサブクラス
- 学籍番号をデータとして保持

- Studentオブジェクトを作る
- getNameで名前を取得して出力する

Javaのクラス

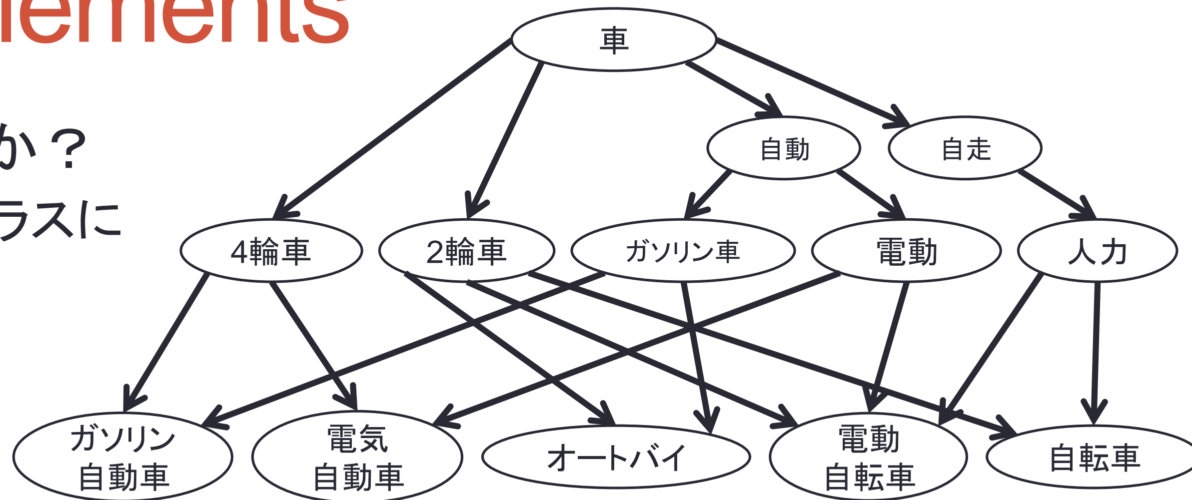
- クラスの定義
  - 親クラスの指定
  - インターフェイスの指定
  - 変数の宣言
  - コンストラクタ
  - メソッドの宣言
- クラス変数
  - クラスで共通の変数
  - **static**をつけて宣言
- クラスメソッド
  - オブジェクトを参照することができない



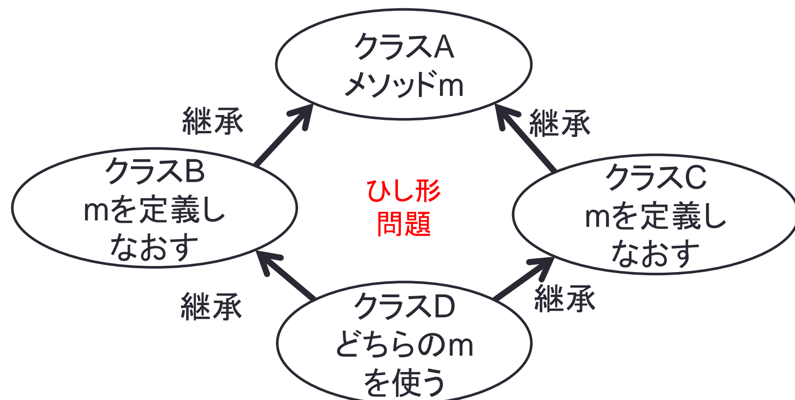


extendsとimplements

- 複数の親クラスを許すか？
  - 2つ以上のクラスの子クラスになっている
  - 複数の性質を持つ



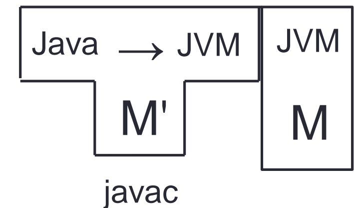
- 菱形継承（multiple inheritance）
  - 複数の親クラスがいる場合，変数やメソッドの継承が問題になる



- Javaでの解決方法
  - 親クラスは1つ（extendsで指定）
  - 複数の性質を持たせるにはインターフェイスを使う
    - インターフェイスは実装のないクラス

Java仮想機械

- スピードとポータビリティの両立
  - コンパイラ ＋ インタープリタ
  - Java仮想機械（Java Virtual Machine）の機械語に翻訳
  - 仮想機械をいろいろなアーキテクチャ上に作成

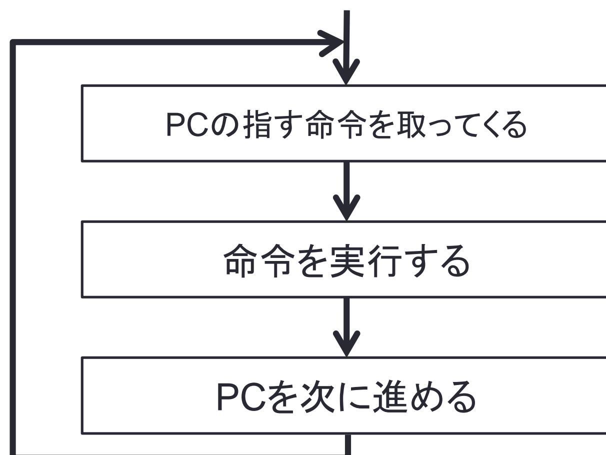


- Java仮想機械（Java Virtual Machine）
  - 仮想（理想）的に考えられたCPU
  - Javaがコンパイルしやすいように設計
  - Javaのオブジェクト指向の実装
  - メモリ管理をガーベジコレクションにより実装

仮想機械の実装

- 仮想機械(Virtual Machine)
  - 仮想的に考えられたCPU
  - 現実のCPUより分かり易い
  - 理想的なCPUをモデル
  - バイト列による命令列

- 仮想機械の実装
  - 仮想的なCPUをエミュレーションするプログラムを作成



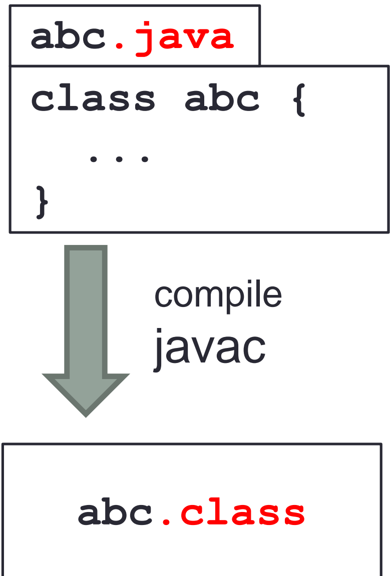

```
for (;;) {
  op = *pc++;
  switch (op) {
  case ILOAD:
    index = *pc++;
    *--sp = fp[index];
    break;
  case IADD:
    v1 = *sp++;
    v2 = *sp;
    *sp = v1 + v2;
    break;
  ....
  }
}
```

PC: プログラムカウンタ

Java仮想機械の構造

- Java仮想機械のコードはclassファイルに格納
- 実行コードはmethodsの中に格納

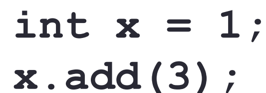
```
ClassFile {
  u4 magic;
  u2 minor_version;
  u2 major_version;
  u2 constant_pool_count;
  cp_info constant_pool[constant_pool_count-1];
  u2 access_flags;
  u2 this_class;
  u2 super_class;
  u2 interfaces_count;
  u2 interfaces[interfaces_count];
  u2 fields_count;
  field_info fields[fields_count];
  u2 methods_count;
  method_info methods[methods_count];
  u2 attributes_count;
  attribute_info attributes[attributes_count];
}
```



Java仮想機械の扱うデータ型

- プリミティブ型
  - 整数
  - 浮動小数点
  - boolean型
  - returnAddress型
- 参照型（＝ポインタ）
  - オブジェクト

プリミティブ型の整数



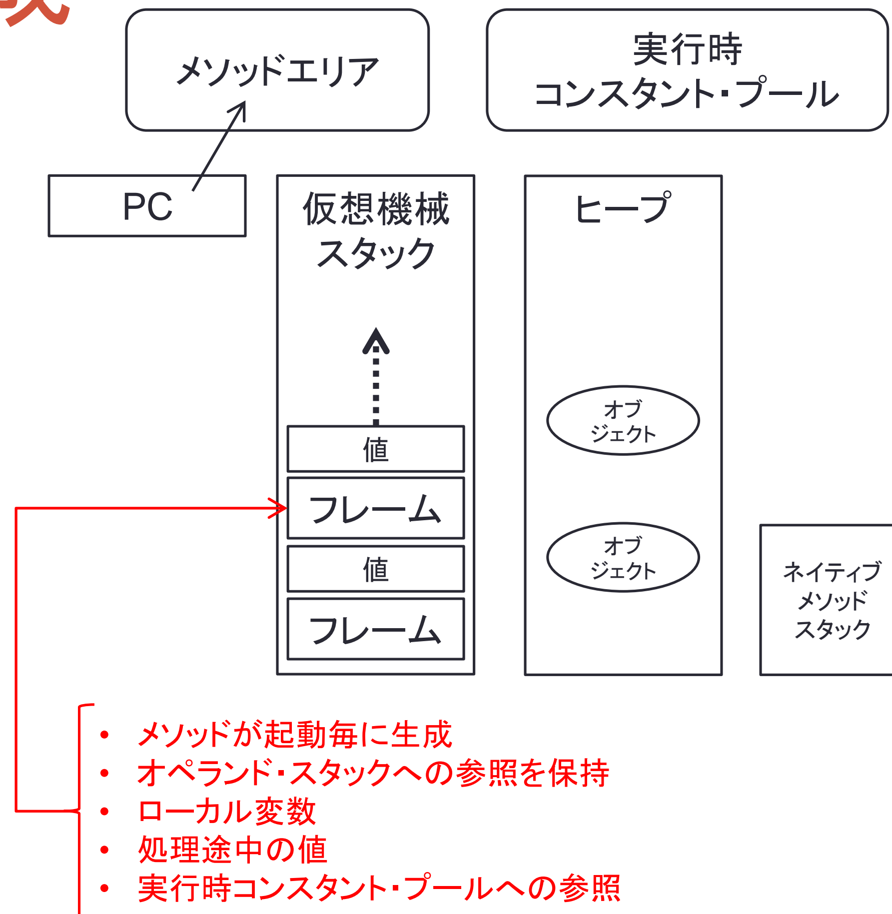
オブジェクトの整数

addは実際にはありません

- Javaは完全なオブジェクト指向ではない
  - 効率を重視
  - プリミティブ型はオブジェクトではない
  - プリミティブ型に対応したオブジェクトを準備

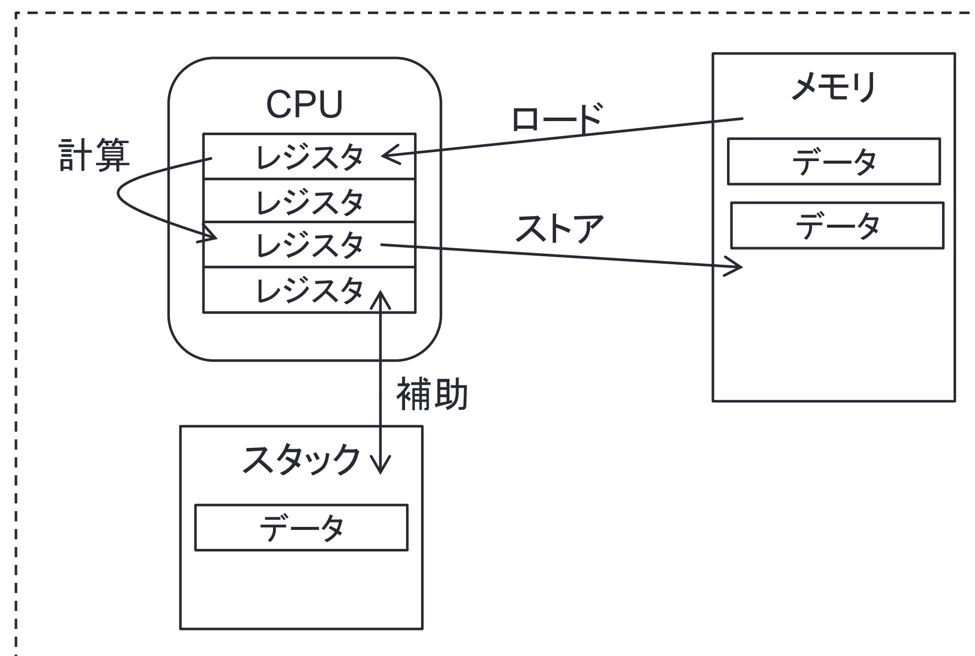
Java仮想機械の構成

- **PCレジスタ**
  - 現在実行しているメソッド内のコードを保持
  - returnAddress型
- 仮想機械スタック
  - メソッドの実行によって作られるフレームを保持
  - メソッド呼び出しでスタック上に積む
  - メソッドが終了した場合，対応するフレームは破棄
- ヒープ
  - すべてのスレッドから共有
  - オブジェクトや配列が割当られる
- メソッド・エリア
  - すべてのスレッドから共有
  - メソッドのコードなどを格納
- 実行時コンスタント・プール
  - クラスあるいはインターフェイス毎に存在
  - コンパイル時に定まるさまざまな定数，実行時に解決されなくてはならないメソッドやフィールドに対する参照などを格納
- ネイティブ・メソッド・スタック
  - Java 以外の言語で記述されたメソッドの実行時に使われるスタック

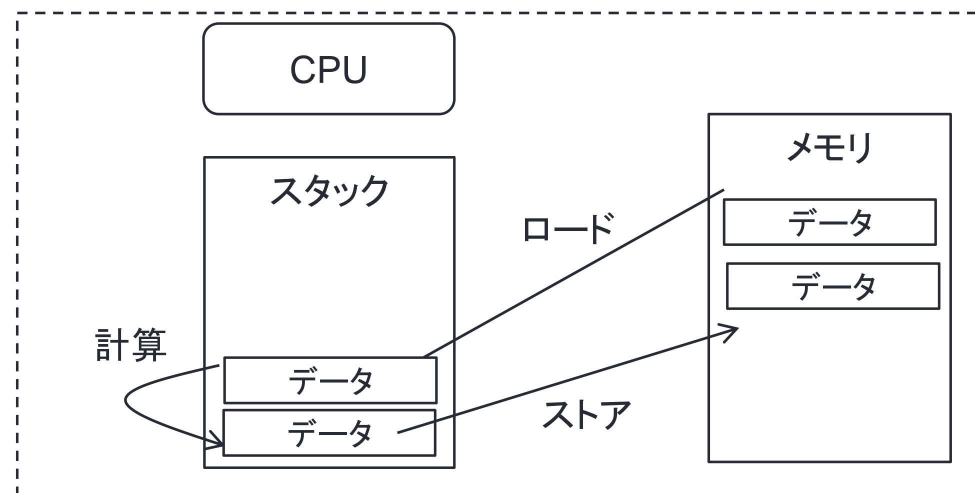


CPUの種類

- レジスタ型
  - 計算はレジスタ上で行なう.
  - アキュムレータが代表的なレジスタ
  - データを一旦CPUレジスタにロードし，計算した後，ストアする
  - 複雑な計算にはスタックを利用



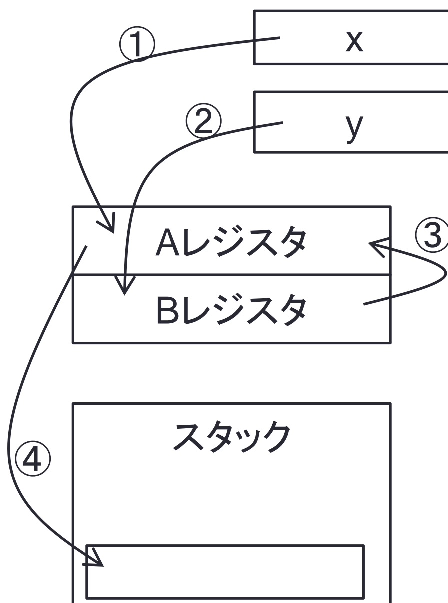
- スタック型
  - CPUは計算用のレジスタを持たない
  - 計算をすべてスタック上で行なう



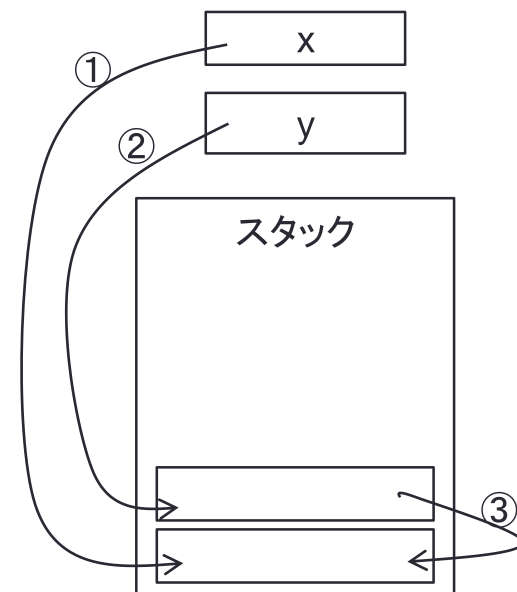
レジスタ型 vs スタック型

```
z = (x + y) * (u + w);
```

- レジスタ型
  ① load x, A
  ② load y, B
  ③ add B, A
  ④ push A
  ⑤ load u, A
  ⑥ load w, B
  ⑦ add B, A
  ⑧ pop B
  ⑨ mult A, B
  ⑩ store B, z



- スタック型
  ① load x
  ② load y
  ③ add
  ④ load u
  ⑤ load w
  ⑥ add
  ⑦ mult
  ⑧ store z

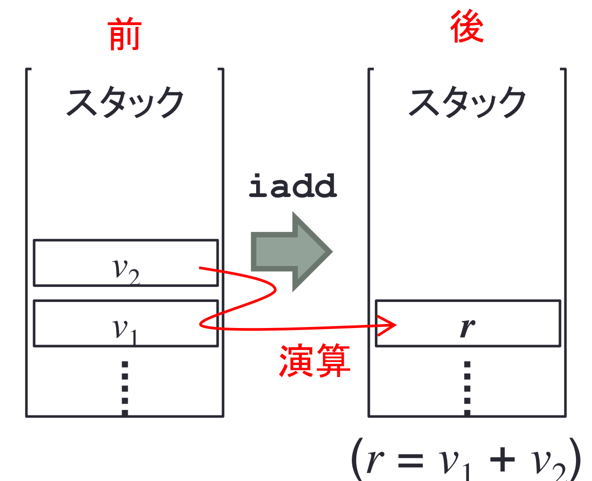


Java仮想機械の命令セット(1)

- 命令
  - オペコードは1バイト
  - 必要に応じていくつかのオペランドを持つ
  - 命令の種類は現在202個(内1つは未使用)
  - 3つが予約されている

- 整数の足し算: **`iadd`**
  - オペコード: **`iadd = 0x60`**
  - オペランド: なし
  - スタック上の操作:

`iadd` $: \ldots, v_1, v_2 \;\Rightarrow\; \ldots, r \quad (r = v_1 + v_2)$

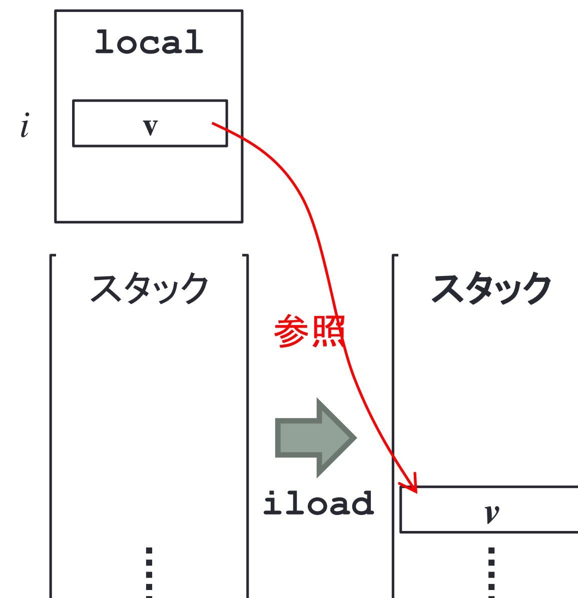


命令セット(2)

- 整数のローカル変数の参照
  - **iload** = 0x15
  - オペランド: 1バイト, インデックス

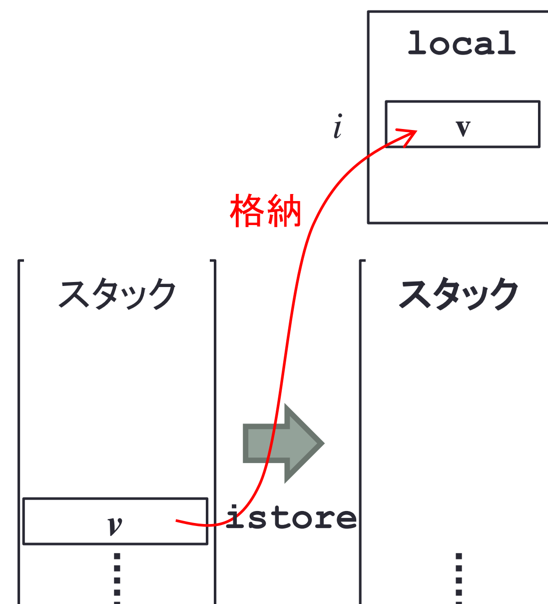
  iload, i : . . . ⇒ . . . , **local**[i]

  - **local**: ローカル変数の配列
  - インデックスが0, 1, 2, 3 の場合は, **iload_0**, **iload_1**, **iload_2**, **iload_3** の1バイトの命令も用意



- 整数をローカル変数に格納
  - **istore** = 0x36

  istore, i : . . . , v ⇒ . . .
  (**local**[i] ← v)

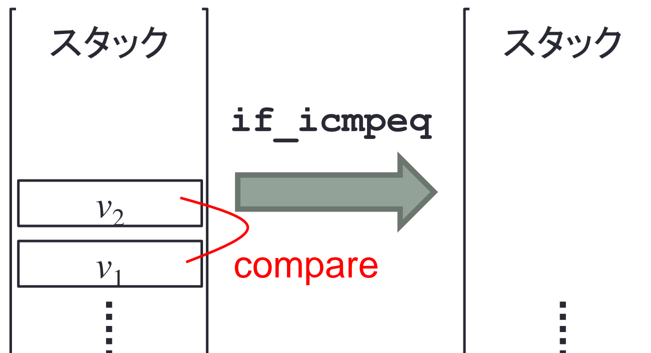


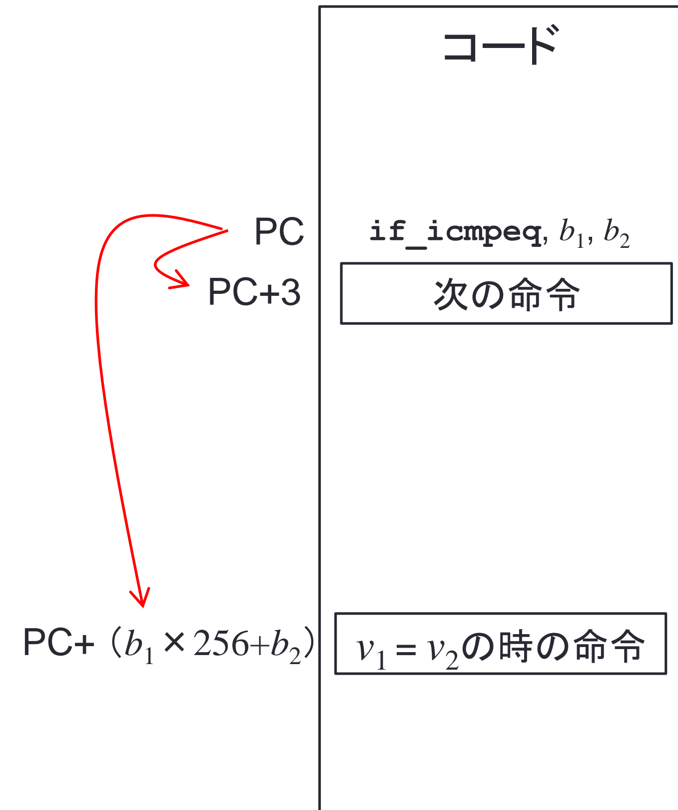
命令セット(3)

- 条件分岐
  - `if_icmpeq = 0x9f`
  - オペランド: 2バイト，分岐先へのオフセット

$$\texttt{if_icmpeq}, b_1, b_2 : \ldots, v_1, v_2 \Rightarrow \ldots$$

  - スタック上のv_1とv_2を取りだし，等しかった場合には，PCの値に($b_1 \times 256 + b_2$)を加える
  - 等しくなかった場合には，次の命令(PC+3)を実行





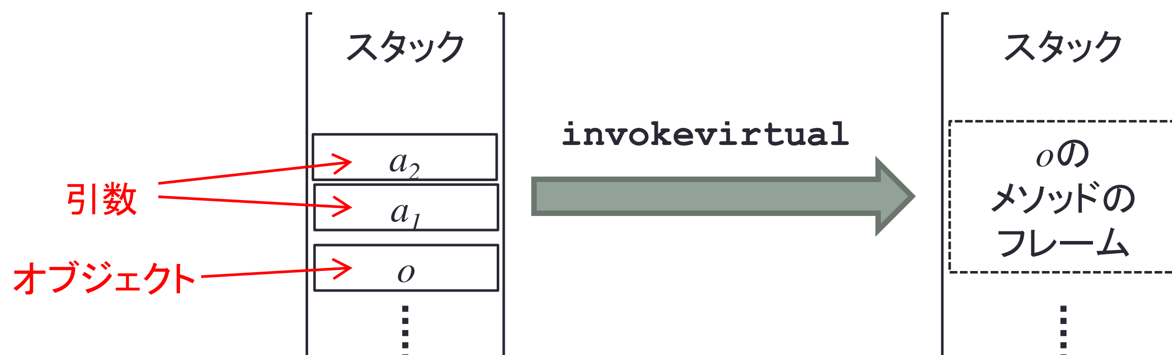
命令セット(4)

- メソッドの呼び出し
  - **invokevirtual = 0xb6**

$$\texttt{invokevirtual}, i_1, i_2 : \ldots, o, [a_1, [a_2 \ldots]] \Rightarrow \ldots$$

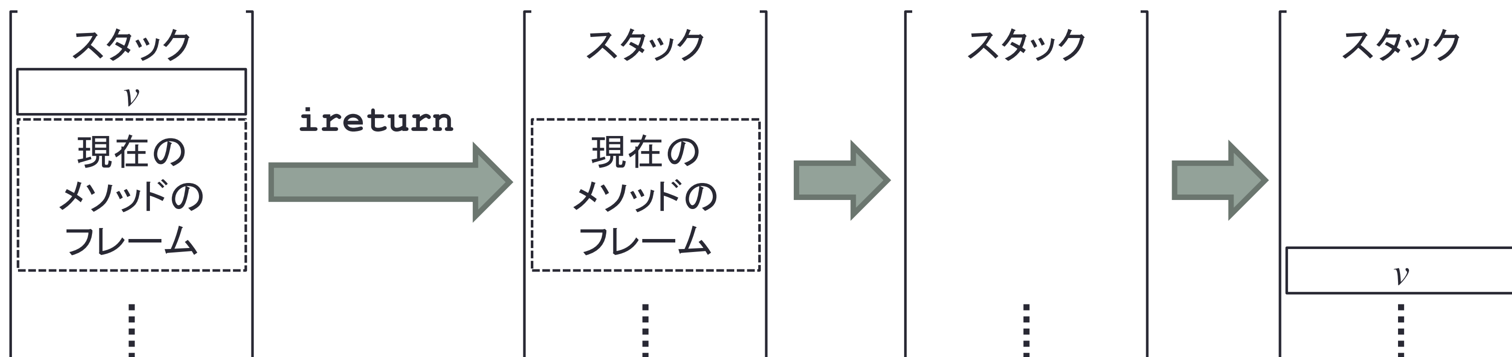
o : オブジェクト
$a_1, a_2, \ldots$: メソッドの引数
$i_1 \times 256 + i_2$: 実行時コンスタント・プールへのインデックス
メソッド名とディスクリプタ(引数の数や型), クラス名の解決

  - クラスから対応する名前とディスクリプタを持つメソッドを探しだす
  - オブジェクトo と引数$a_1, a_2, \ldots$ の新しいフレームを作成
  - PCをメソッドの最初の命令にセットしメソッドの実行を開始
  - **synchronized**の場合には, oに対するモニタを獲得



命令セット(5)

- メソッドから戻る
  - `ireturn` = `0xac` など
  - `ireturn` : . . . , v ⇒
  - 整数を返す
  - 現在のフレームから値vを取りだす
  - フレームを解放
  - 呼び出したフレームのスタックにvを入れる



コンパイル例（1）

- Java 仮想機械のコードを見るには「`javap -c`」を使う
- 単純なループ

```
void spin() {
   int i;
   for (i = 0; i < 100; i++) {
      ; // Loop body is empty
   }
}
```

```
Method void spin()
 0 iconst_0
 1 istore_1
 2 goto 8
 5 iinc 1,1
 8 iload_1
 9 bipush 100
11 if_icmplt 5
14 return
```

コンパイル例(2)

- 引数を持つメソッド

```
int addTwo(int i, int j) {
   return i + j;
}
```

```
Method int addTwo(int,int)
 0 iload_1
 1 iload_2
 2 iadd
 3 ireturn
```

- メソッド呼び出し

```
int add12and13() {
   return addTwo(12, 13);
}
```

```
Method int add12and13()
0 aload_0
1 bipush 12
3 bipush 13
5 invokevirtual #4 // Example.addTwo(II)I
8 ireturn
```

- **aload_0**: 自分自身**this**をスタックに入れる
- クラスメソッドの場合には**invokestatic**

コンパイル例(3)

- インスタンスの作成

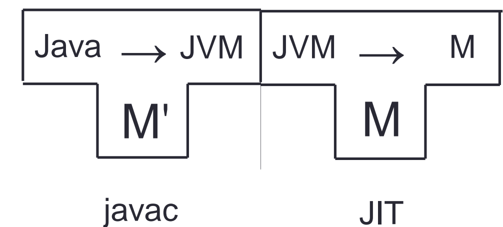
```
Object create() {
   return new Object();
}
```

```
Method java.lang.Object create()
 0 new #1 // Class java.lang.Object
 3 dup
 4 invokespecial #4 // Method java.lang.Object.<init>()V
 5 areturn
```

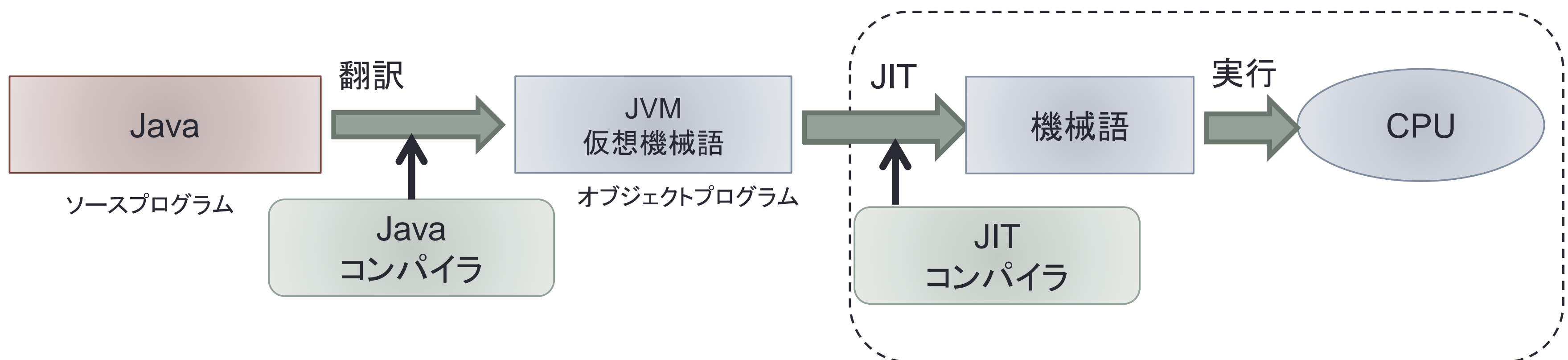
- **new**で新しいオブジェクトを作る
- 親クラスのコンストラクタを呼び出す
- **areturn**でオブジェクトを返す.

JITコンパイラ

- 仮想機械を使う場合の利点と欠点
  - ポータビリティが高くなる
  - 実行速度がネイティブ機械語より遅くなる



- JITコンパイラ
  - Just In Time
  - 仮想機械語を実行時にネイティブ機械語に変換する
  - 最初の変換に時間がかかるが，その後はネイティブと同じ速度



まとめ

- Java
- オブジェクト指向プログラミング言語
- 仮想機械
- 参考文献
  - 「Java 仮想マシン仕様 –第2版–」, Tim Lindholm, Frank Yellin, ピアソン・エデュケーション
  - https://docs.oracle.com/javase/specs/