Reference, 配列，アクセスフラグ，jdbでの観察，アセンブラ上での編集

2002年6月6日

海谷 治彦

目次

- Reference型
  - 要はインスタンスを指すデータ型
- 配列型
  - Java配列の復習とJVMでの扱い.
- アクセスフラグ
  - クラス，メソッド，属性，それぞれ.
- jdbでの観察
  - 主にJavaスタック内部の様子を実際に見る.
- アセンブラ上での編集
  - jasmin上でプログラムをいじくる

リファレンス型

- インスタンス(そして配列)を指すデータ構造.
- 直感的にはヒープ内のアドレスとインスタンスの型(すなわちクラス)のデータと思ってよい.
- 型の表記法が妙だけど覚えよう.
  Lパッケージを含めたクラス名 ;
  - 上記Lと;が予約語.
  - 例 `Ljava/lang/String;`

例

```
class Refs{
String name=null;
    void setName(String s){
        name=s;
    }

    String getName(){
        return name;
    }

    void shareName(Refs r){
        name=r.getName();
    }
}
```

```
.method setName(Ljava/lang/String;)V
    .limit stack 2
    .limit locals 2
    aload_0
    aload_1
    putfield Refs/name Ljava/lang/String;
    return
.end method

.method shareName(LRefs;)V
    .limit stack 2
    .limit locals 2
    aload_0
    aload_1
    invokevirtual Refs/getName()Ljava/lang/String;
    putfield Refs/name Ljava/lang/String;
    return
.end method
```

注意

- パッケージ名もつけた形で展開されます.
  - Stringなら，java/lang/String
  - Vectorなら，java/util/Vector
- import文をつかっていても，それも展開されます.
- パッケージ名は，. で区切られてますが，/ に変換されます.

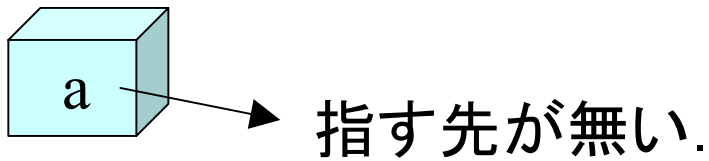
Javaの配列を復習

- 以下の3通りに分けると整理しやすい.
  - 基本データ型の一次元配列
  - インスタンスの一次元配列
  - 多次元配列
- でも，本質的には全て同じ原理.
- (一次元)配列自体，Javaではオブジェクト(インスタンス)である.
- 教科書 p.143辺りからが関連.

基本データ型の一次元配列

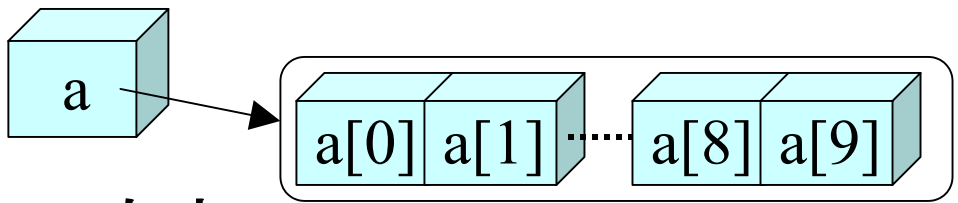
- 配列宣言，この時点で記憶実体は無い．

```
int[] a;
```



- メモリを割り当てる，この例では10個．

```
a=new int[10];
```



- 上記をまとめて書いても良い．

```
int[] a=new int[10];
```

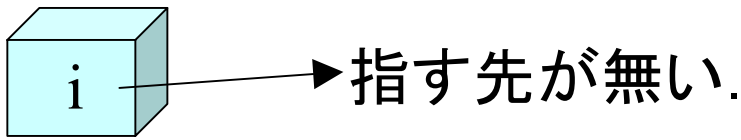
- []の位置を変えても良い．

```
int a[]=new int[10];
```

インスタンスの一次元配列 1/2

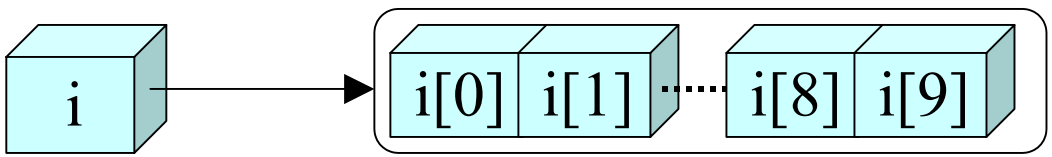
- 配列宣言，この時点で記憶実体は無い．

```
Inst[] i;
```



- メモリを割り当てる，この例では10個．

```
i=new Inst[10];
```



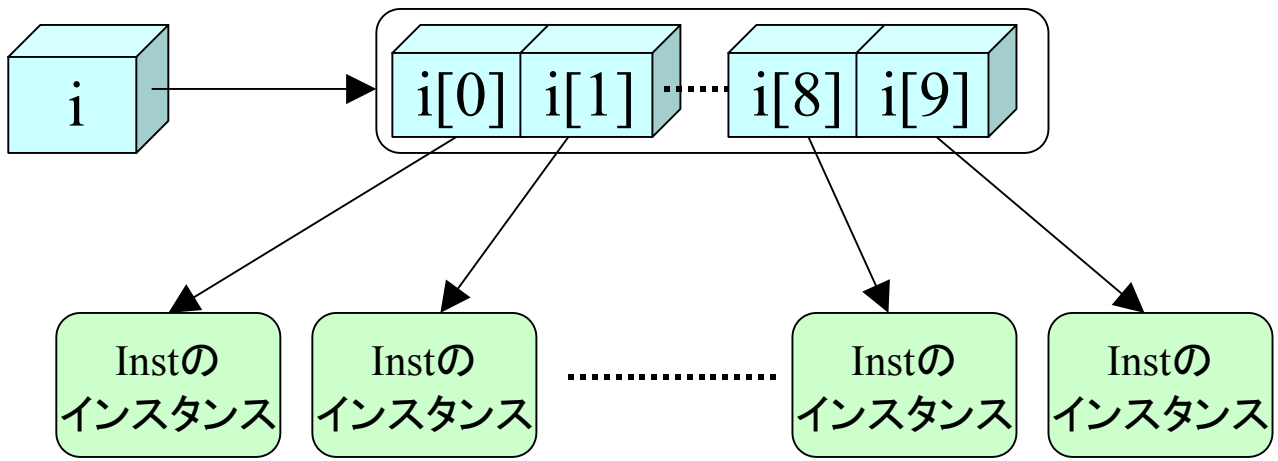
インスタンスの実体はこの時点では無い！

- 無論，`Inst[] i=new Inst[10]`も有り．
- インスタンスの実体を作る．（次頁へ）

インスタンスの一次元配列 2/2

- インスタンスの実体を作る．例えば，以下．

```
for(int c=0; c<i.length; c++){
        i[c]=new Inst();
}
```

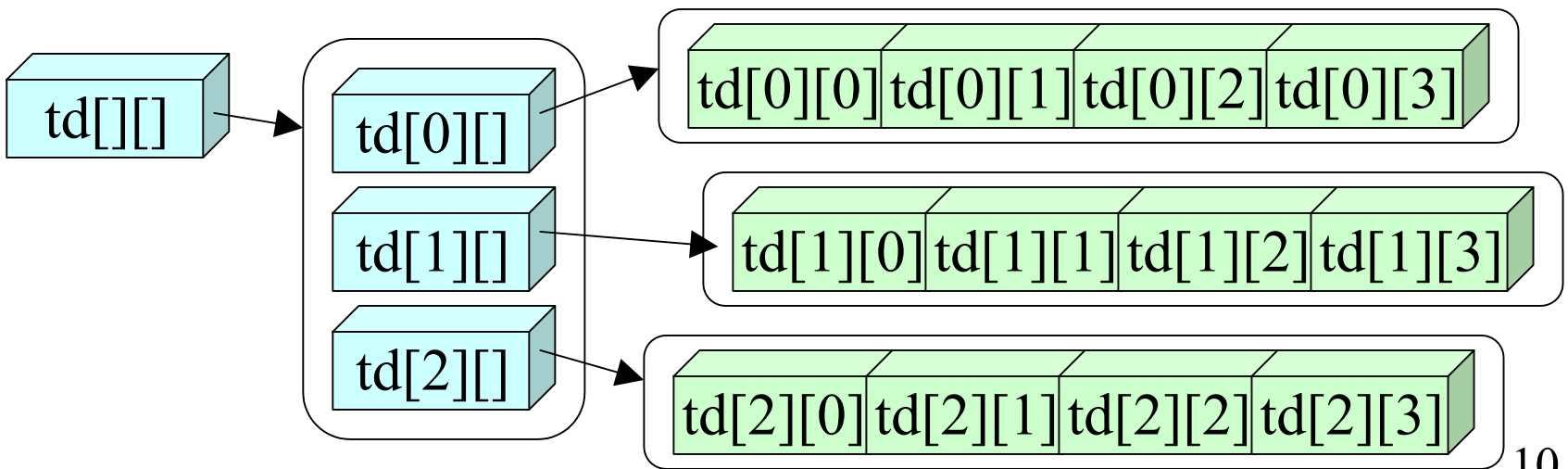


多次元配列

- 例えば二次元の場合，以下のように書く.

```
int[][] td=new int[3][4];
```

- 構文からも察しがつくが，Javaでは，一次元配列の配列にて二次元配列を構成.
  - 要は，一次元配列がプリミティブ.



JVMでの配列型の表現

- 一次元の場合
  [型名
    - 例 `[I` や `[Ljava/lang/String;` など
- 二次元の場合
  [[型名
    - 例 `[[B` や `[[java/util/Vector;` など.
- なんか誤植みたいだけど，こういう表記.
- mainの引数等が代表例.

JVMでの基本型一次元配列

実体の無い場合

```
void onlyDefIntA(){
int[] a;
}

void allocIntA(){
int[] a=new int[4];
}
```

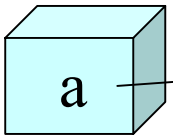
実体の有る場合

p.439 newarray の説明を参照

```
.method onlyDefIntA()V
    .limit stack 0
    .limit locals 2
    return
.end method

.method allocIntA()V
    .limit stack 1
    .limit locals 2
    iconst_4
    newarray int
    astore_1
    return
.end method
```

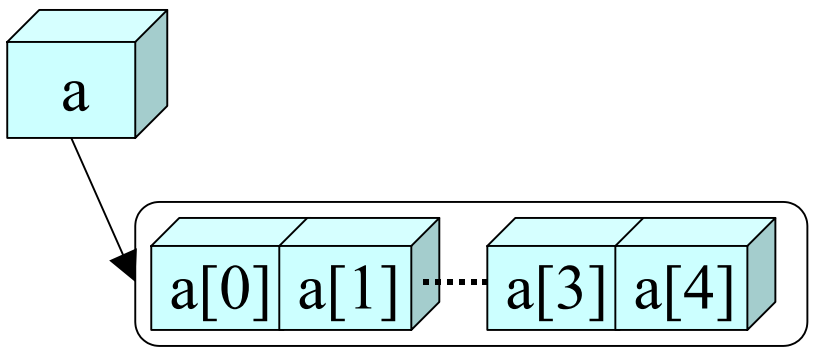
JVMでのインスタンス型配列 1/2



指す先が無い.

```
void onlyDefObjRefA(){
Array1[] a;
}

void allocObjRefA(){
Array1[] a=new Array1[5];
}
```




```
.method onlyDefObjRefA()V
    .limit stack 0
    .limit locals 2
    return
.end method

.method allocObjRefA()V
    .limit stack 1
    .limit locals 2
    iconst_5
    anewarray Array1
    astore_1
    return
.end method
```

p.228 anewarray

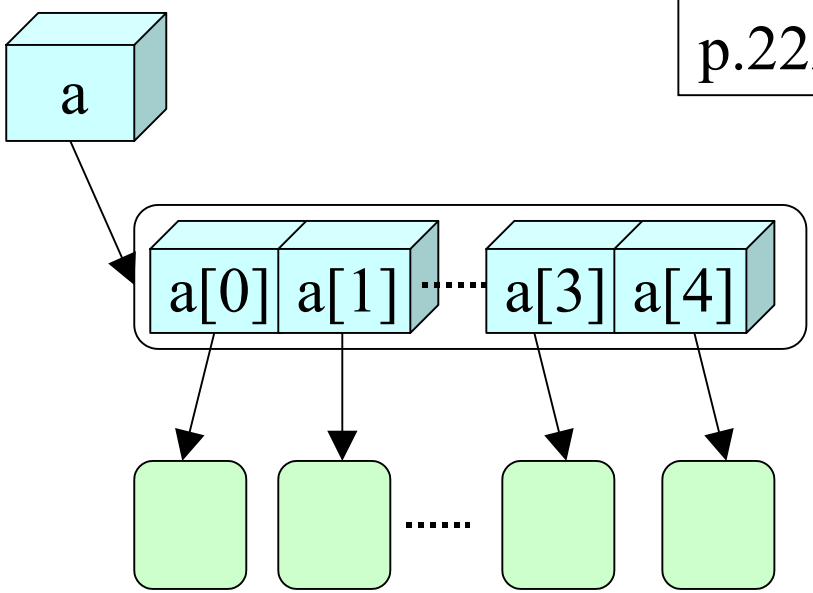
JVMでのインスタンス型配列 2/2

```
void allocObjA(){
Array1[] a=new Array1[5];
    for(int i=0; i<a.length; i++)
        a[i]=new Array1();
}
```

```
.method allocObjA()V
    .limit stack 4
    .limit locals 3
    iconst_5
    anewarray Array1
    astore_1
    iconst_0
    istore_2
    goto Label2
```

p.222 aastore

```
Label1:
    aload_1
    iload_2
    new Array1
    dup
    invokespecial Array1/<init>()V
    aastore
    iinc 2 1
Label2:
    iload_2
    aload_1
    arraylength
    if_icmplt Label1
    return
.end method
```



多次元配列

```
void intArray2(){
    int[][] a=new int[3][4];
        a[2][3]=5;
}
```

p.432
multianewarray

p.220
aaload

p.321
iastore

```
.method intArray2()V
    .limit stack 3
    .limit locals 2
    iconst_3
    iconst_4
    multianewarray [[I 2
    astore_1
    aload_1
    iconst_2
    aaload
    iconst_3
    iconst_5
    iastore
    return
.end method
```

JVMでの配列生成のまとめ

- 基本データ型の配列 (boolean, char, float, double, byte, short, int, long)
  - p.439のnewarray
- インスタンスの配列
  - p.228のanewarray
- 多次元配列
  - p.432のmultianewarray

JVMでの配列取り出しのまとめ

- 基本データ型の配列
  - iaload(p.318)やfaload(p.284)などを使って値をとりだします.
- インスタンスの配列
  - aaload(p.220)を使う.
- 多次元配列
  - aaload(p.220)を複数回使って最終的な1次元配列にたどりつき，最後に，iaload(p.318)やfaload(p.284)などを使って値をとりだす.

JVMでの配列格納のまとめ

- 基本データ型の配列
  - iastore(p.321)やfastore(p.285)などを使って値を格納します.
- インスタンスの配列
  - aasotre(p.222)を使う.
- 多次元配列
  - aasotre(p.222)を使い，aaloadと同様に，最後は，iasotre(p.321)や，fastore(p.285)を使って値を格納する.

配列の要素数

- arraylength (p.233)
- バイトコードのプリミティブとして用意されている.

アクセスフラグ

- クラス p.58 or p.195
  - 明示的に指定がないと，ACC_SUPERのみ立つ．(同パッケージ内でpublic)
- フィールド p.62 or p.209
  - 明示的指定がないと，全bitは寝てる．同じパッケージ内のみからアクセス可．
- メソッド p.73 or p.210
  - 同上．

jdb

- Java Debugger
- Javaプログラムのデバッグを行うためのツール.
- C言語用のgdbやdbxに似ている.
- この授業では，JVMの内部観察に使う.
- Java2以降，ちょっぴり仕様が変わった.

jdbによる観察に必要な技 1/2

- プログラムの実行を途中(breakpoint)で止める.
  - stop in クラス名.メソッド名
  - stop at クラス名.行番号
- 次のbreakpointまで実行を継続.
  - cont
- 一行実行，一マシン語命令実行
  - step, stepi

jdbによる観察に必要な技 2/2

- Javaスタック内のフレームを表示
  - where
- 存在するスレッドを列挙
  - threads
- ロードしてるクラスを列挙
  - classes
- 現フレームのローカル変数を表示
  - locals

その他はマニュアル見てね.

観察の手順

- 観察するプログラムを –g オプションをつけてコンパイル
  - 例 **`javac -g SomeClass.java`**
- jdb クラス名 引数
- 必要なコマンドを入れる.
- jdb –launch クラス名 引数 でもよい.

観察の例

- 前回の演習問題等

アセンブラ上での編集

- アセンブラ言語(jasmin)もプログラム言語には違いないので，直接，エディタでプログラムを書ける.
- アセンブラプログラムをクラスファイルに変換するコマンド

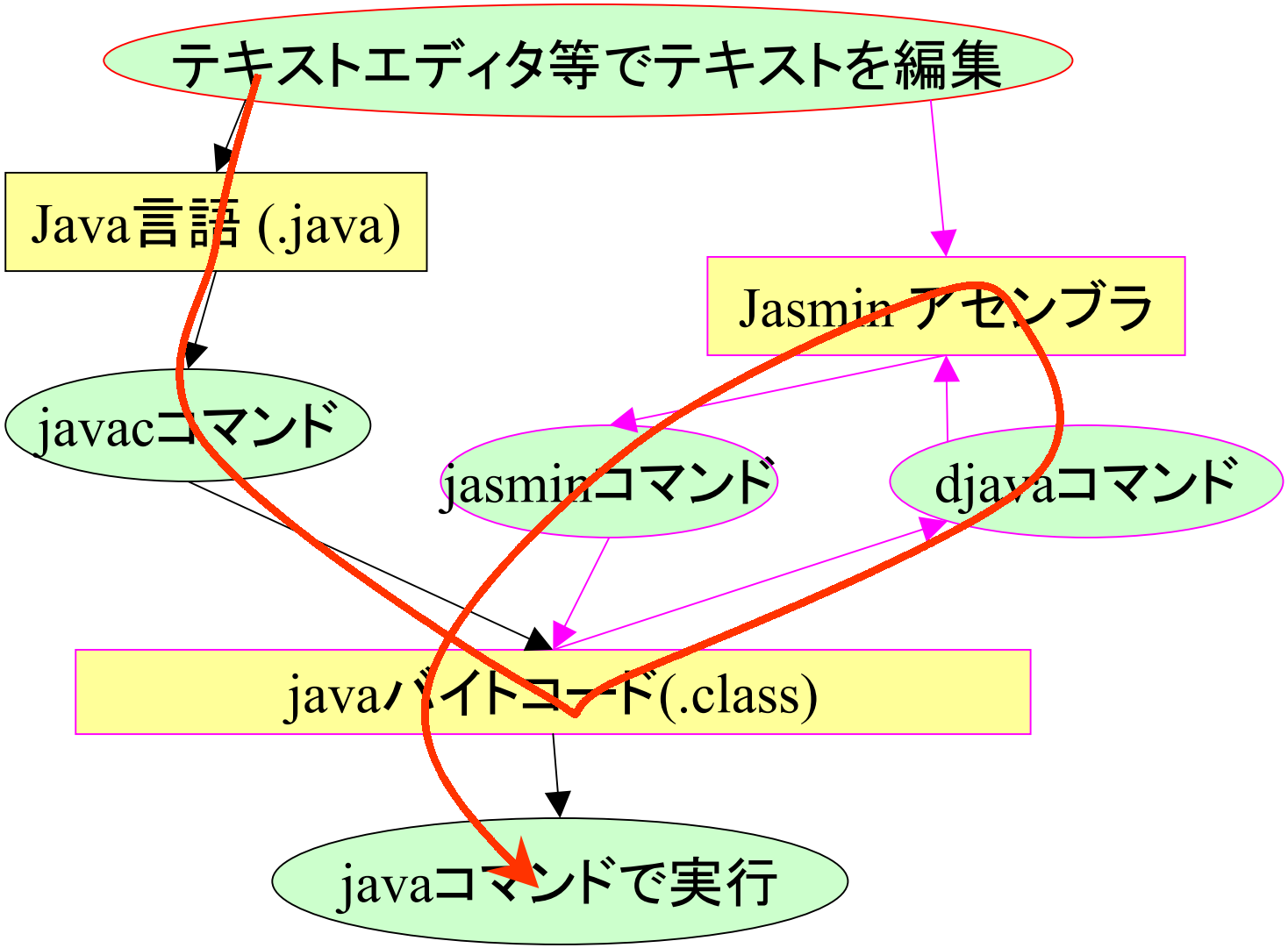
  `jasmin`
- 使い方

  `jasmin ナントカ.j`
  - で，指定されたクラスファイルが現ディレクトリに作成される.

クラスファイルの改造

- 以下のステップで改造
  - クラスファイルを得る.
  - djavaで逆アセンブル (なんとか.j)
  - なんとか.j をエディタで編集.
  - jasminでアセンブル (新しい, なんとか.class)
- ゼロからアセンブラで書くより楽.
- ソースコード無しでもプログラムを改造可能
- 不正アクセス.... とか.

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

改造例

- 出力文字列の変更.
- ちょっとした最適化.
- 二引数メソッドの雛型.
- privateフィールドの解除