

第9回のキーワード

1

アルゴリズム関係

- 連結リスト (linked list)
- 単方向連結リスト
(single linked list)
- ノード (node) / セル (cell)
- リンク (link)
- 再帰データ型
(recursive data type)
- 番兵法 (sentinel method)

Java関係

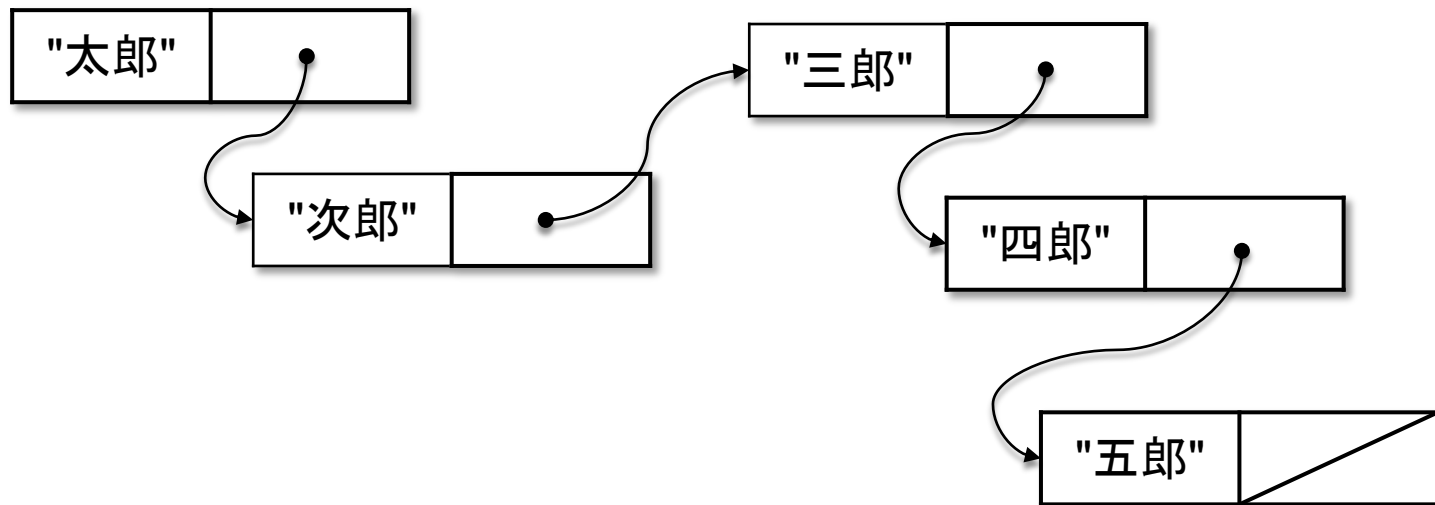
- 参照型 (reference type)
- 自己参照クラス
(self-referential class)
- `node1.next = node2`
- `head = null`
- `for (Node n = head;
n != null; n = n.next)`

連結リスト(linked list)

2

- データを次々と数珠つなぎにしたデータ構造

先頭



末尾

- 利点: 登録数が無制限 & 途中に挿入・削除が容易

ノード(セル)

3

- 連結リストでデータを入れるための箱
 - ▣ 「データ」と次のノードへの「つながり」を持つ



「つながり」を英語で
言えば「リンク」

- クラスによる表現(自己参照クラス / 再帰データ型)

```
class Node {  
    String data;  
    Node next;  
}
```

え!? ノードの中
にノードが入って
いるの?

そうではなくて、
次のノードへの
「つながり」

“参照型”の性質

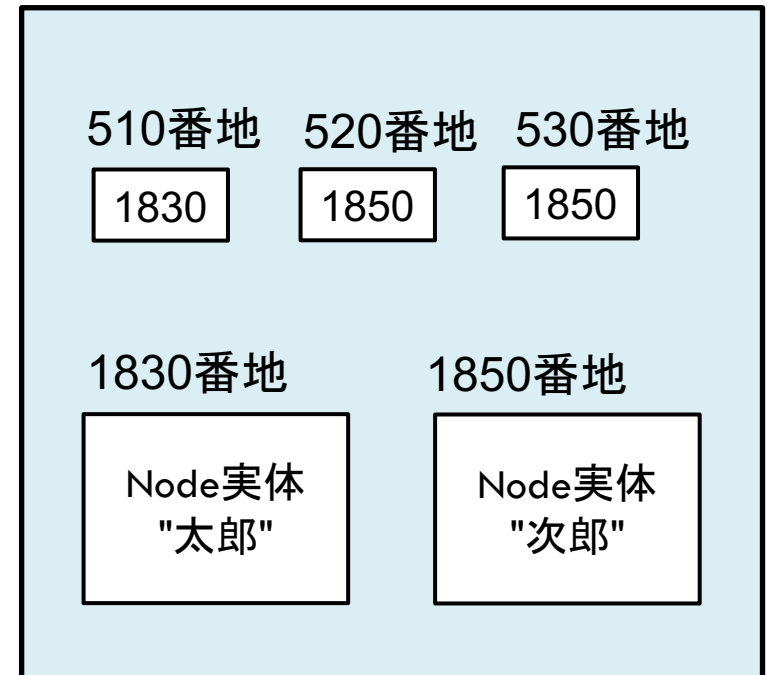
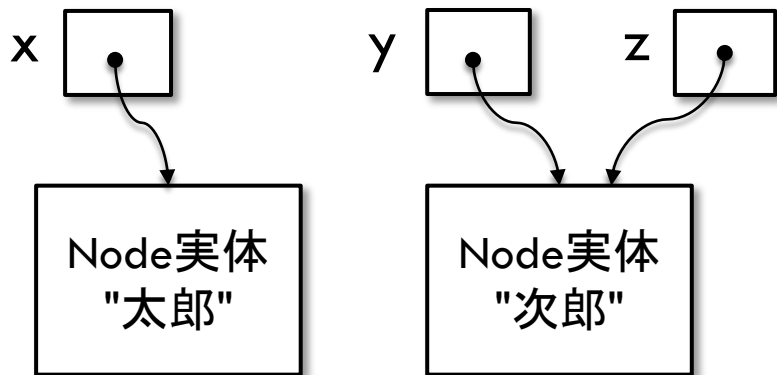
4

- Javaでは、クラス型や配列型の変数やフィールドは、インスタンス(実体)への参照(=つながり)しか持たない

```
Node x = new Node("太郎");
```

```
Node y = new Node("次郎");
```

```
Node z = y;
```



メモリ内でのイメージ(番地は適当)

ノードの連結

5

- 参照型の性質により，代入操作によってノードの連結を実現できる

```
Node node1 = new Node("太郎");
```

```
Node node2 = new Node("次郎");
```

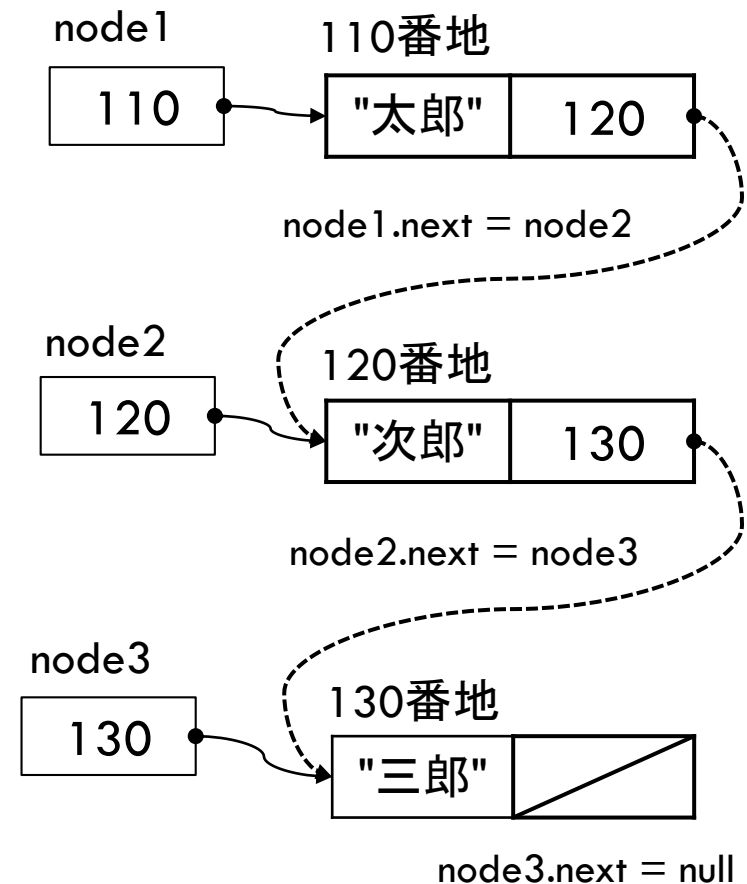
```
Node node3 = new Node("三郎");
```

```
node1.next = node2;
```

```
node2.next = node3;
```

```
node3.next = null;
```

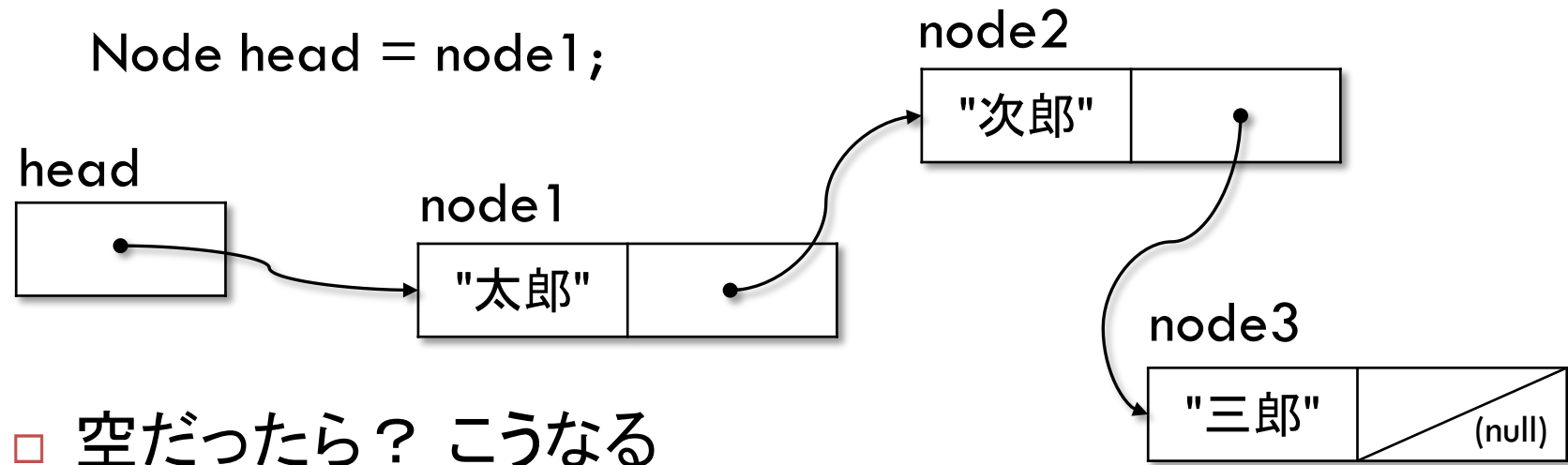
「終」のマークとして
nullを使う



スタート地点は必要

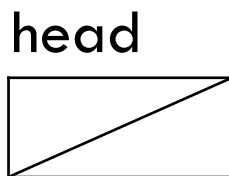
6

- リストの要素を順にたどれるようにするためにはいつも先頭ノードを指す変数 (head) が必要



- 空だったら？ こうなる

Node head = null;

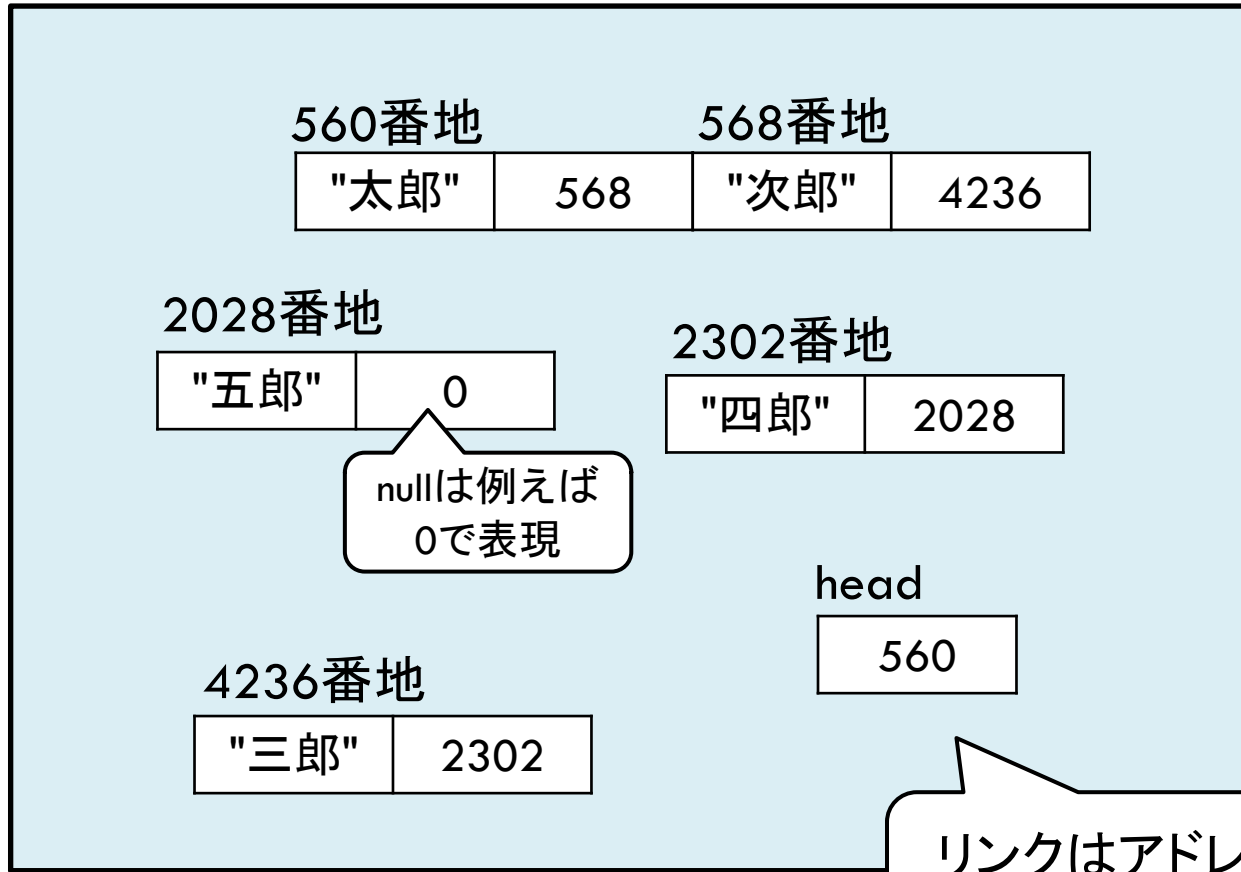


nullが入っていることを
図では斜線で表す

メモリ内でのイメージ例

7

アドレス
0番地
1000番地
2000番地
3000番地
4000番地
5000番地



nullは例えば
0で表現

head

560

メモリマップ

リンクはアドレス
(番地)で実現

リンクをたどる

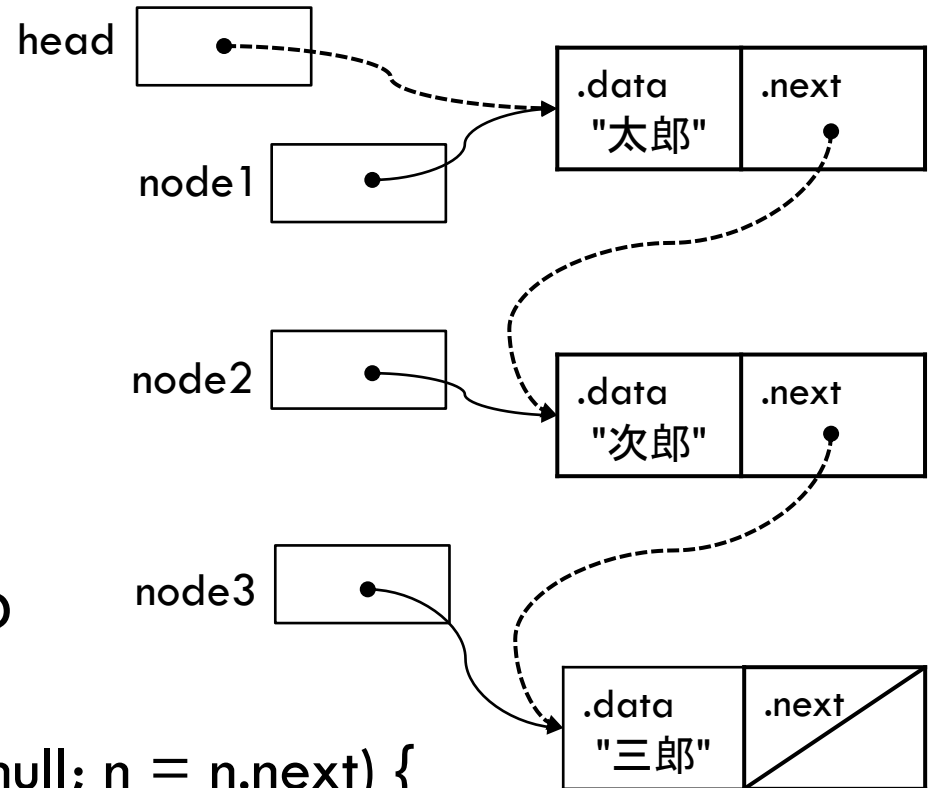
8

□ こういうふうにも書ける

```
head = node1;  
head.next = node2;  
head.next.next = node3;  
head.next.next.next = null;
```

□ for文を使ってたどれる

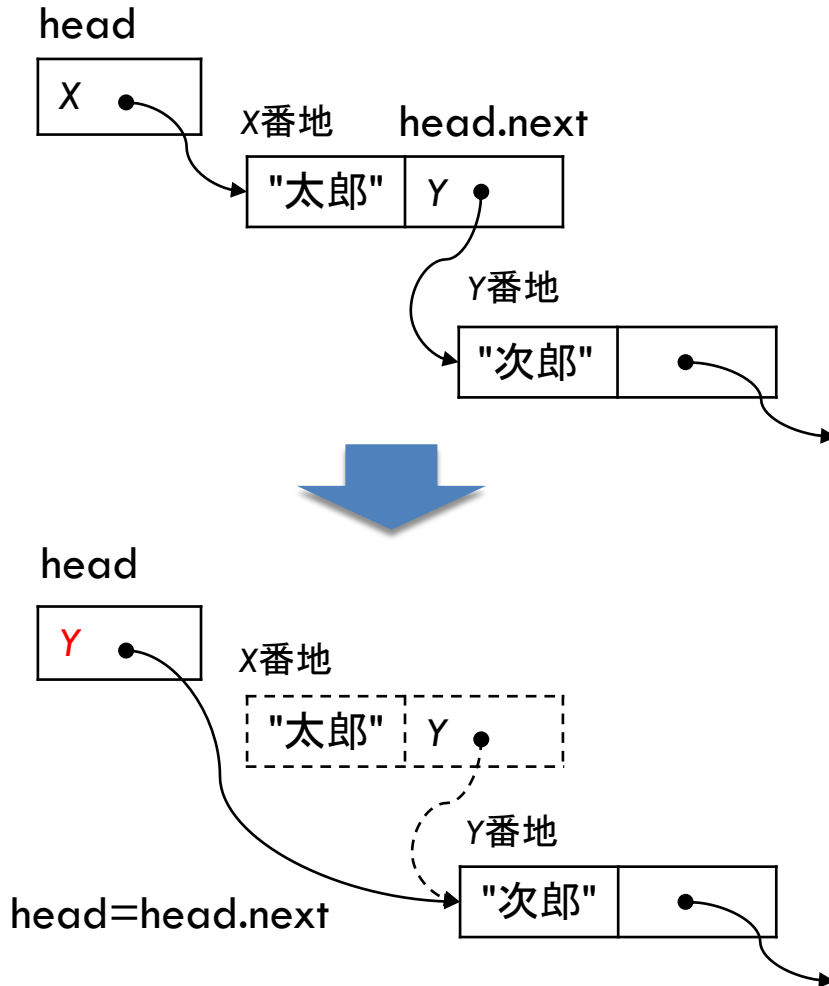
```
for (Node n = head; n != null; n = n.next) {  
    System.out.println(n.data);  
}
```



リスト先頭での削除と挿入

9

□ 先頭ノードの削除 (pop)



□ 先頭ノードの挿入 (push)

