

# アルゴリズムとデータ構造

第10回 連結リストの応用

# 第10回のキーワード

2

## アルゴリズム関係

- 連結リストへの挿入
- 連結リストからの削除
- 双方向連結リスト  
(doubly linked list)
- 循環リスト  
(circular linked list)
- $O(1)$
- 連結リストの探索と整列

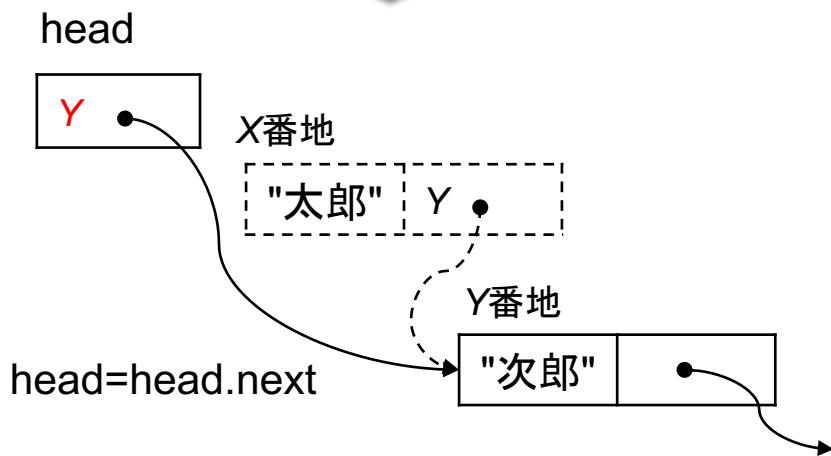
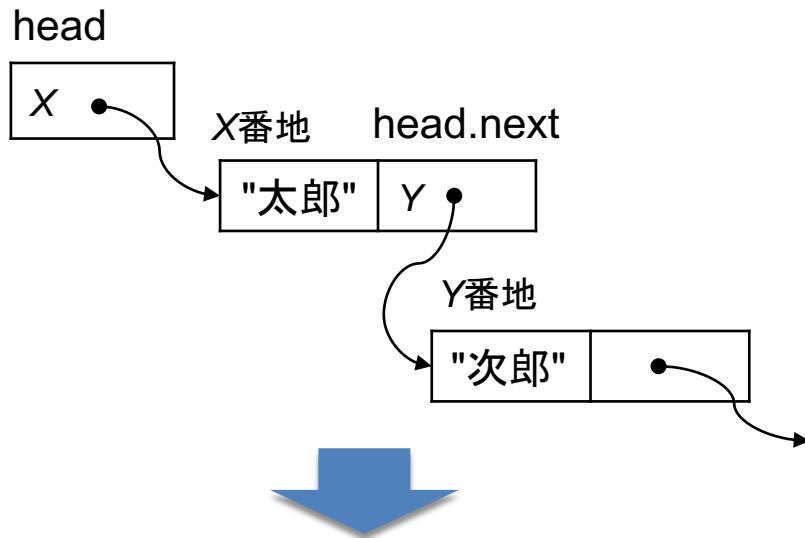
## Java関係

- ジェネリクス / 総称型  
(generics)
- ジェネリッククラスの定義  
class クラス名<E> { ... }
- LinkedList<E>

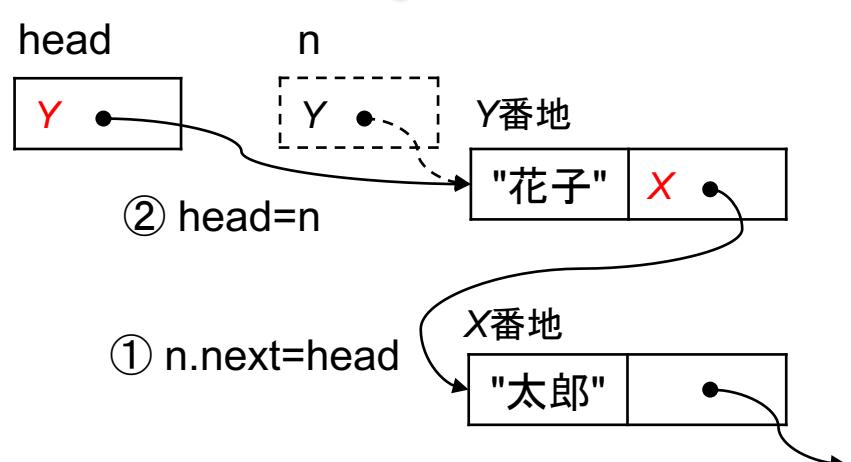
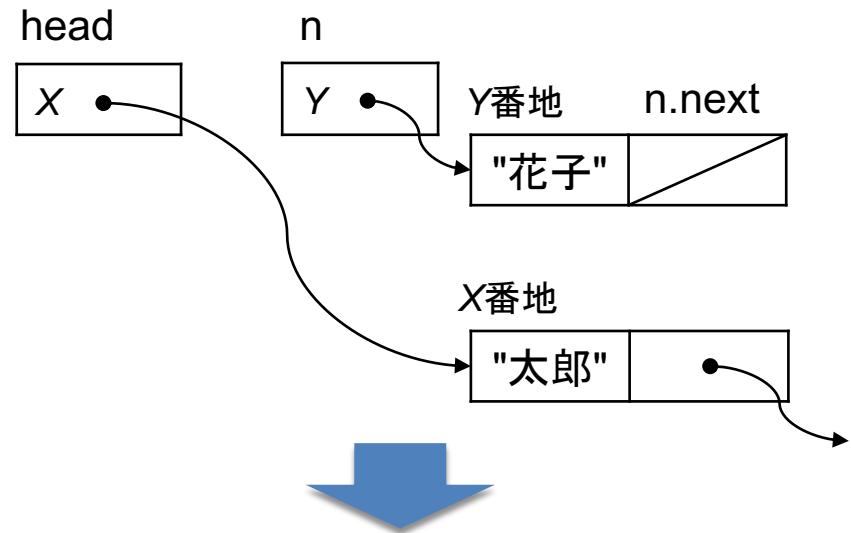
# リスト先頭での削除と挿入

3

## □ 先頭ノードの削除(pop)



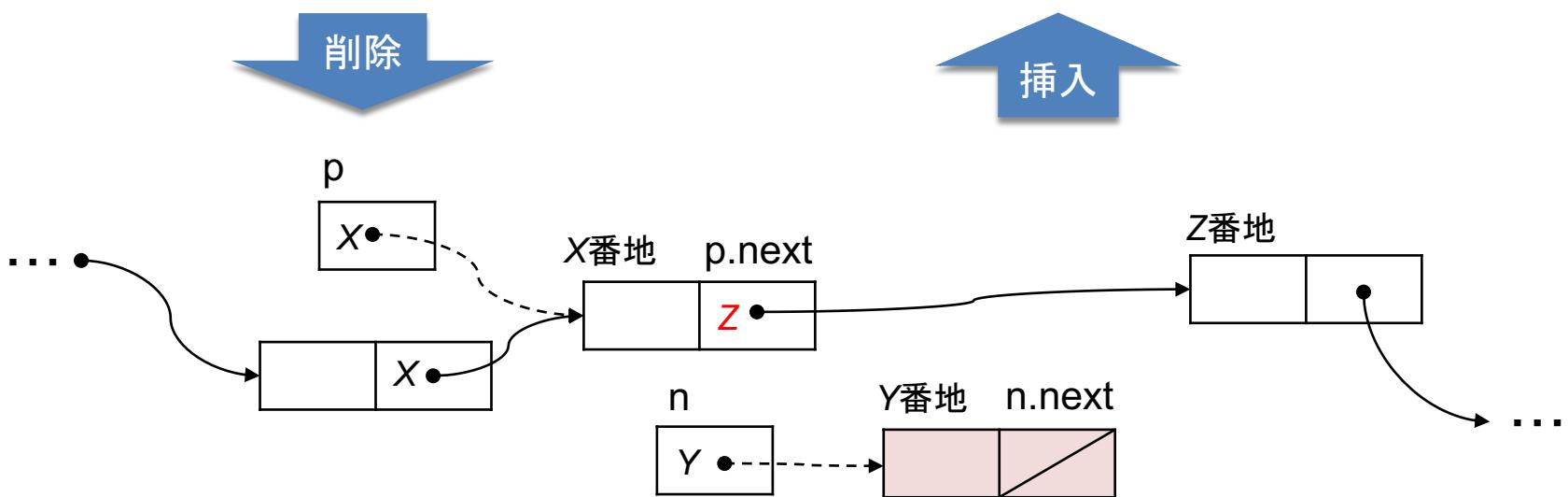
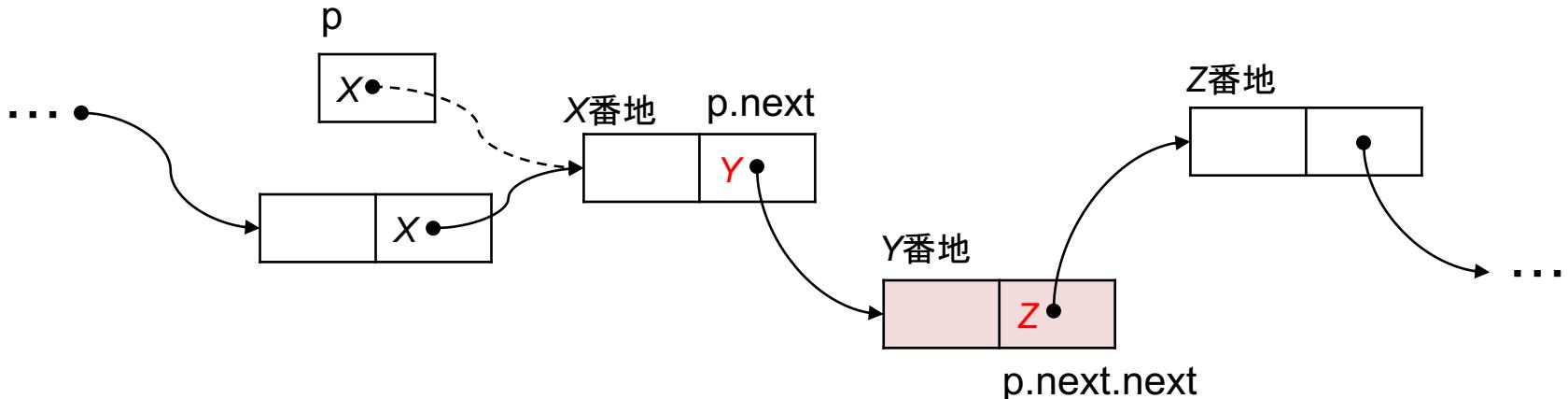
## □ 先頭ノードの挿入(push)



# リスト途中での削除と挿入

4

- リストの途中(や末尾)でノードを削除・挿入する



# 確認問題

5

## □ 削除と挿入の実装

- 下記は、連結リストにおけるノードの削除と挿入の実装例である。空欄を埋めてメソッドを完成させよ。

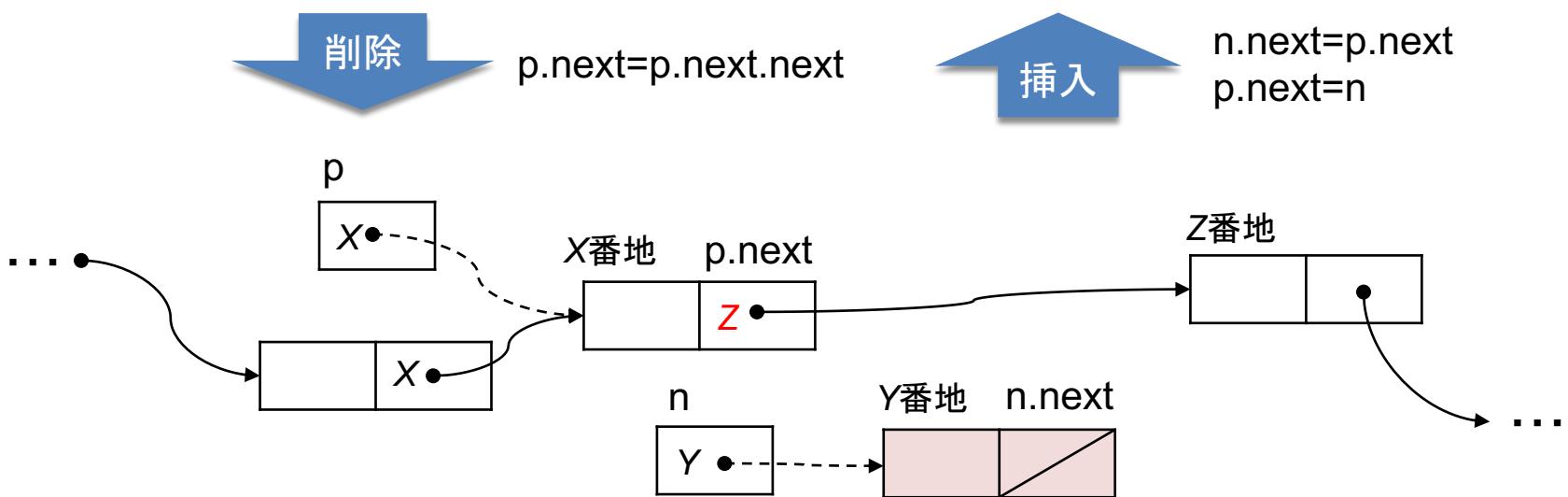
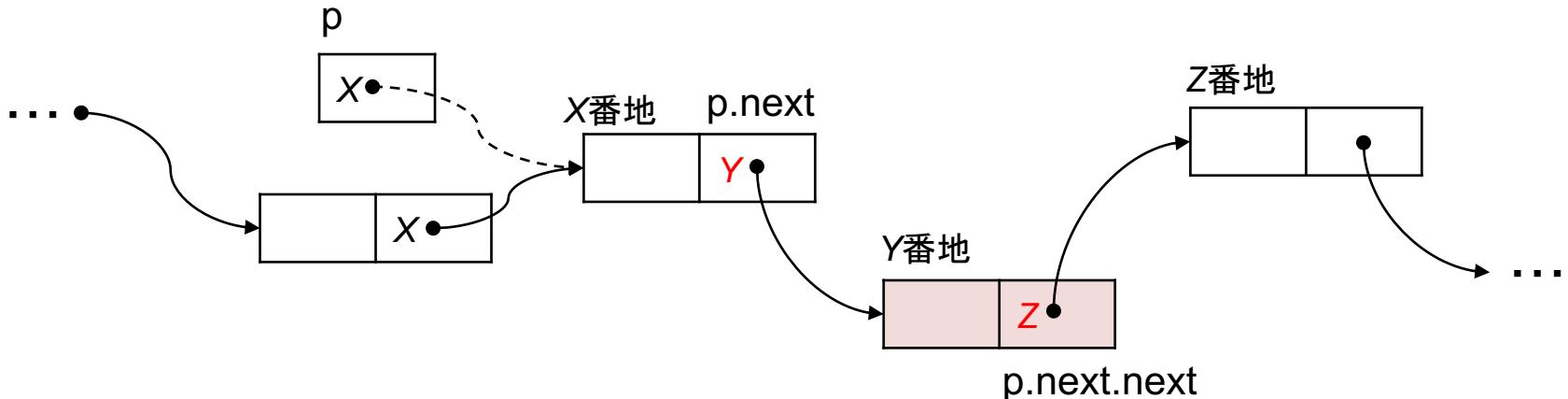
```
// ノードpの次のノードを削除
public void
removeNext(Node p) {
    if (p == null) {
        // pがnullのときは先頭を削除
        if (head != null) {
            head =
        }
    } else if (p.next != null) {
        // そうでないときは通常の削除
        p.next =
    }
}
```

```
// ノードpの次に新しいノードnを挿入
public void
insertNext(Node p, Node n) {
    if (p == null) {
        // pがnullのときは先頭に挿入
        n.next =
        head =
    } else {
        // そうでないときは通常の挿入
        n.next =
        p.next =
    }
}
```

# リスト途中での削除と挿入

6

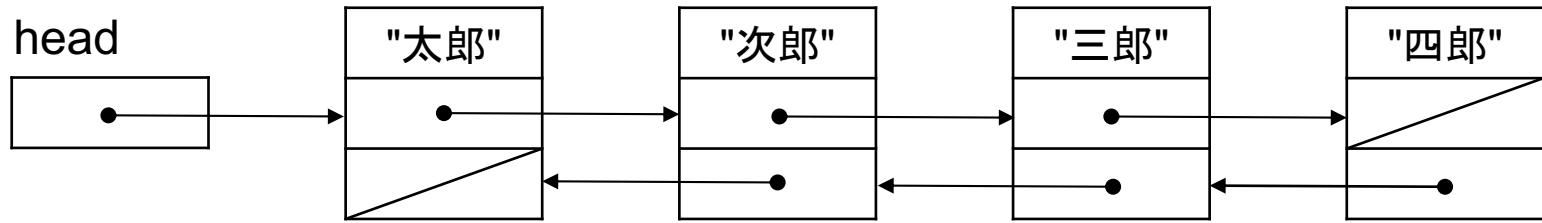
- リストの途中(や末尾)でノードを削除・挿入する



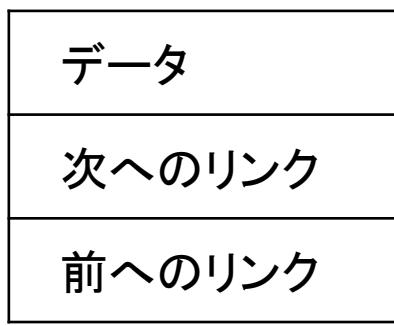
# 双方向連結リスト

7

- 前後のノードへのリンクを保持
  - 前から後ろだけでなく、後ろから前にも、たどれる
  - 現在指しているノードを削除する処理が実装できる



- ノードの構造



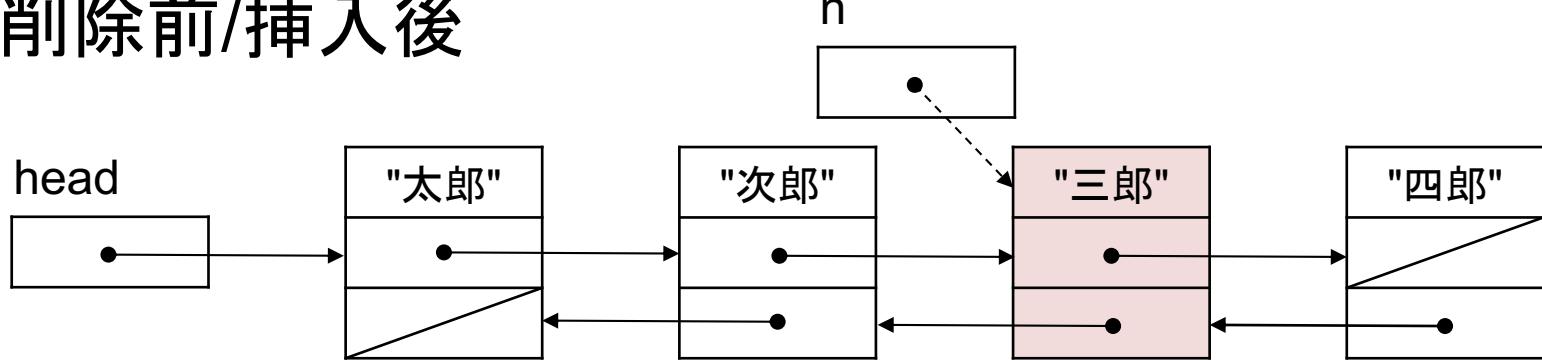
```

class Node {
    String data;
    Node next; // 次へのリンク
    Node prev; // 前へのリンク
}
  
```

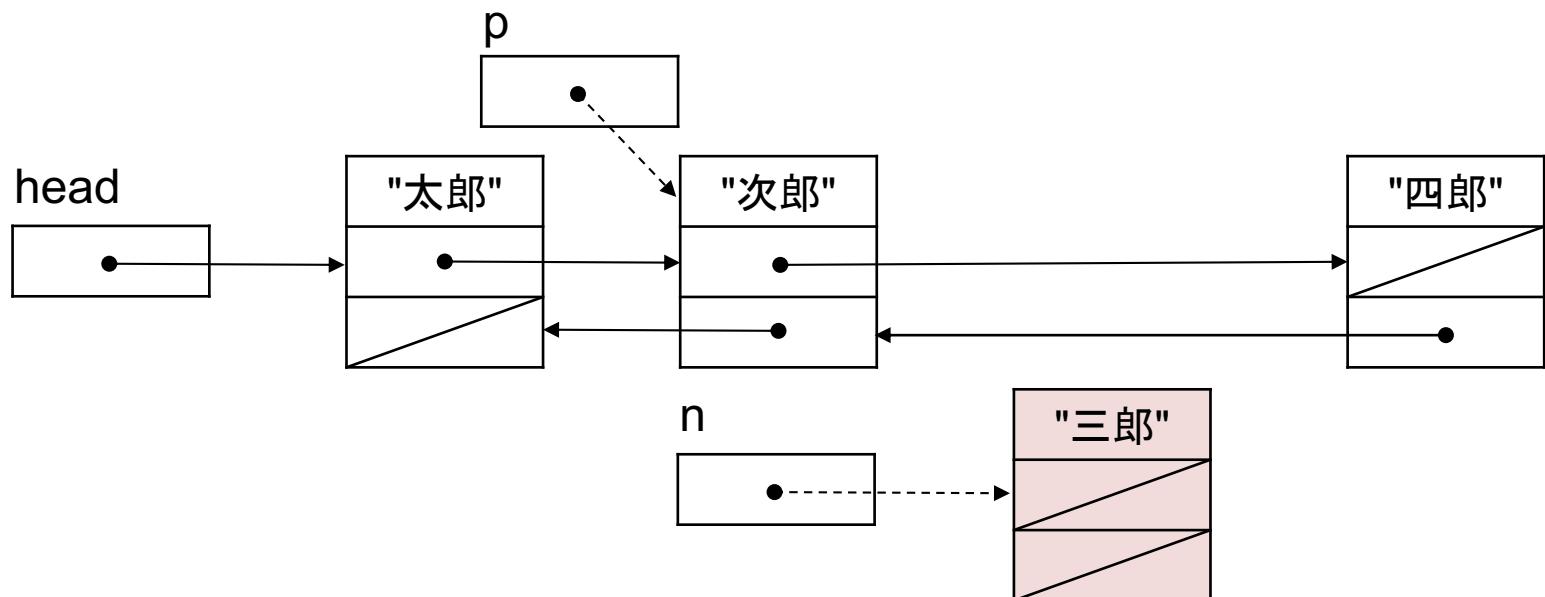
# 双向連接リストの挿入と削除

8

## □ 削除前/挿入後



## □ 削除後/挿入前



# 確認問題

9

## □ 双方向連結リスト

- 前ページの図に削除と挿入の処理の手順を書き加えよ。
- 下記はそれらの実装例である。空欄に入る処理を考えよ。

```
// リストの中にあるノードnを削除
public void remove(Node n) {
    if (n == head) {
        // 先頭を削除する場合はheadを調整
        head = n.next;
    }
    if (n.prev != null) {
        n.prev = n.next;
    }
    if (n.next != null) {
        n.next = n.prev;
    }
}
```

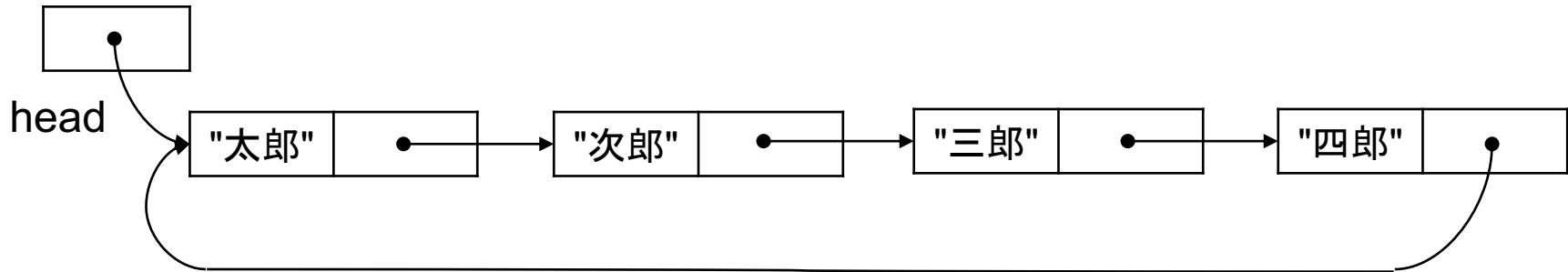
双方向連結リストの細かい実装は発展課題だが、要点は理解すること

```
// ノードpの次に新しいノードnを挿入
public void insert(Node p, Node n) {
    if (p != null) {
        n.prev =
        n.next =
        p.next =
    } else {
        // pがnullなら先頭に挿入
        n.prev = null;
        n.next = head;
        head = n;
    }
    if (n.next != null) {
        n.next.prev =
    }
}
```

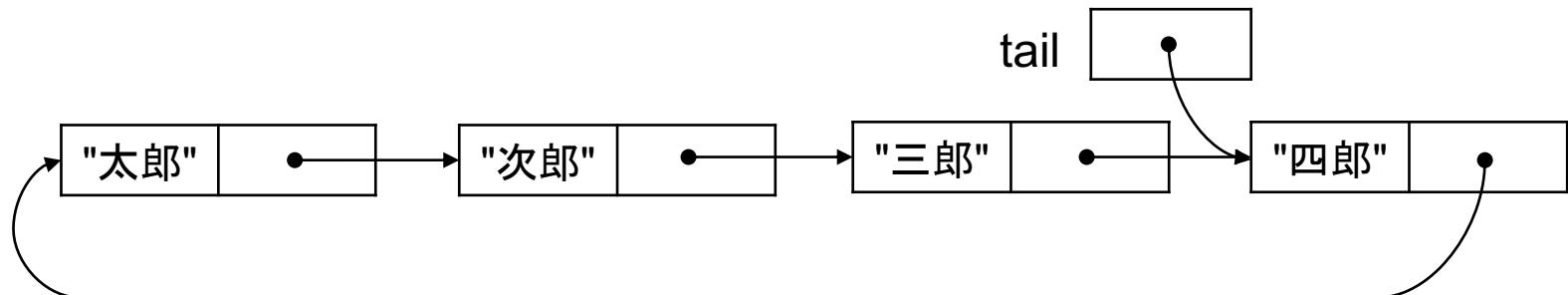
# 循環リスト

10

- 末尾から先頭につなげる
  - ▣ リストを「回転」し、順に繰り返して処理することが容易



- 効率的なキューの実装に利用できる
  - ▣ 末尾(tail)だけ保持すれば、先頭はその次で求められる



# 連結リストの探索と整列

11

- 配列との構造の違い
  - 配列は、ランダムアクセス（途中の要素を直接参照可能）
  - 連結リストは、シーケンシャルアクセス（先頭から順にたどる）
- 連結リストの探索
  - 基本的には、先頭から順にたどって線形探索をする
  - 高速化したい場合は、リストではなく木構造などを用いる
- 連結リストの整列
  - 値のコピーの代わりにノードの移動を使える方法が効率的
  - 挿入ソート：挿入位置を空けるためのコピーが不要になる
  - マージソート：マージ処理における退避領域が不要になる

# Javaジェネリクス

12

- ジェネリック(総称的)プログラミング
  - 特定のデータ型(クラス)に依存しないようにコードを記述可能にして、アルゴリズムを再利用可能にする手法
  - 利用するときには、必要なデータ型の名前を当てはめる指示を書くと、そのデータ型の対応版が実現される
- Javaジェネリクス
  - クラスまたはメソッドの定義では、「<仮クラス名>」という書式によって、実際のクラスが入れ替えられることを示す
  - 定義の中では、仮クラス名(例:E)を使ってコードを書く
  - インスタンス生成時には、実際のクラス名を当てはめる
  - 例: class Node<E> { ... } → new Node<String>();