データ構造とプログラミング：連結リスト

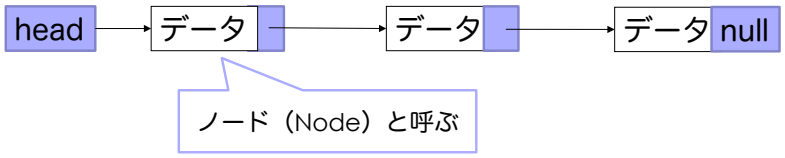
情報工学科 山本哲男

配列を利用したListの問題点

- 挿入・削除をする際に要素をずらしていく必要がある
- 配列の確保を事前にしておく必要がある
- 最初に確保した配列の大きさ以上に挿入しようとした場合，作り直す必要がある

連結リスト

- それぞれの箱（ノード）は以下の二つから構成
  - 要素のデータの値
  - 次の要素を指すポインタ



リストの宣言

- elementはデータの値
- next は次の箱を指す
- head は先頭の箱を指す

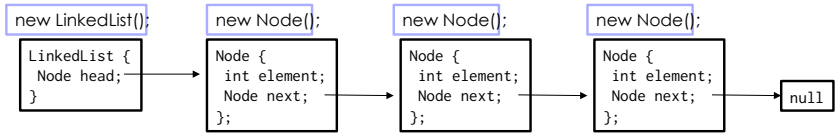
```
public class LinkedList implements List {
    Node head;
    ....
};
```

```
class Node {
    int element;
    Node next;
};
```

要素の型を指定
この例ではint

連結リスト

- それぞれの箱（ノード）は以下の二つから構成
  - 要素のデータの値 （element変数）
  - 次の要素を指すポインタ（next変数）

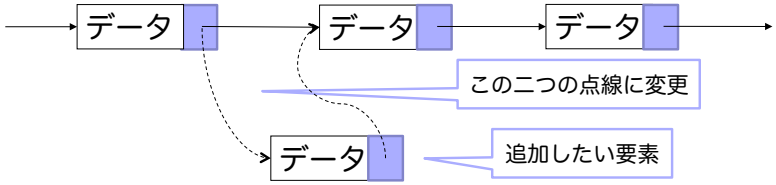


リストの参照（get）

- ランダムアクセス不可能
  - ArrayListは可能
- リストの先頭からnextをたどる必要がある
- 要素の番号がkだとするとk回たどる必要がある

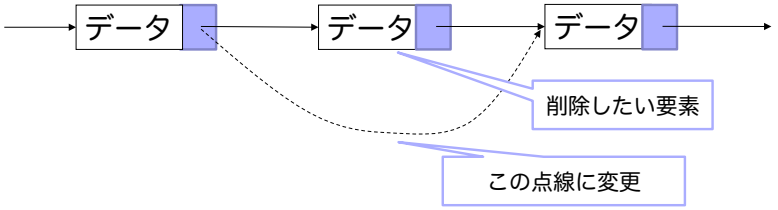
リストへのデータ挿入（add）

- 任意の場所にデータを挿入する計算量はO(1)
  - ただし事前にたどる必要がある



リストからのデータ削除（remove）

- 任意の場所のデータを削除する計算量はO(1)
  - ただし事前にたどる必要がある

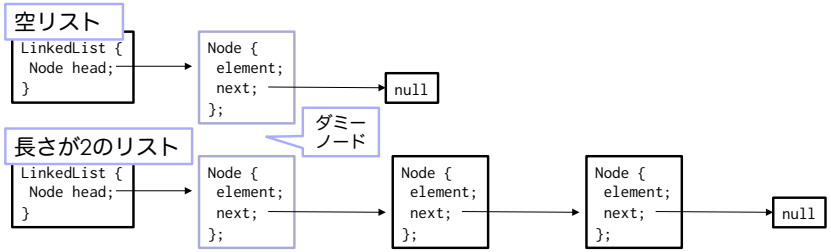


実装の指針

- k番目のノードを取得するprivateメソッドを作成
- add，remove
  - 0番目と1番目以降で処理を分ける
  - 0番目の時はheadの値が変わるため
  - nextのつなぎかたの順番に注意してつなぎかえること

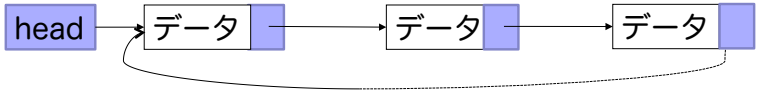
実装の指針（ダミーノード版）

- headと最初のノードの間にダミーのノードを追加
  - 0番目を特別扱いする必要がなくなる

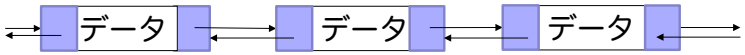


各種リスト

- 環状リスト（循環リスト，巡回リスト）



- 双方向リスト



たどる時の工夫

- データアクセスの局所性を考慮
  - 直前にアクセスしたノードへの参照を保存しておく
  - 保存しておいたノードから辿った方が速そうな場合はそのノードから辿る

最終ノード

- 最後にノードを追加・削除する処理は多用される
- 最終ノードの参照を格納する変数を利用
  - 最終ノードのaddやremoveはこの変数を利用し高速に処理

```
public class LinkedList implements List {
    ...
    Node last;
    ....
};
```


エラーコード

- 戻り値だけでは正常値とエラーコードが混在する場合がある
  - int retrieve(int p)
  - 正常なときは値，異常なときはエラーコード
- 呼び出し元でのエラーチェック処理が分散

```
int x = list.retreive(0);
if (x == -1) {
    // エラー処理
}
int x = list.retreive(1);
if (x == -1) {
    // エラー処理
}
```


例外処理

- エラー処理を分離して記述できるように例外処理という仕組みを導入

```
try {
  ...
  //エラー処理を含まない通常処理を記述
  ...
} catch (IOException e) {
  // エラー処理を記述
  // 変数eにはエラーの内容等が保存されている
} catch (Exception e) {
  // catchの後に記述した型に一致した例外のみ実行
}
```


例外のスロー

- エラーを返したい場合は

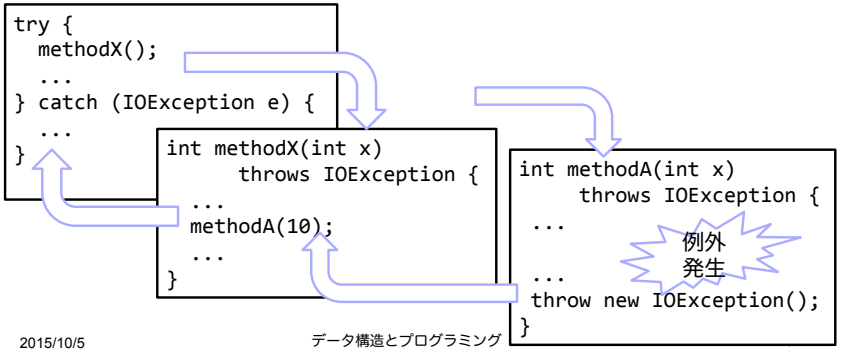
```
throw 例外クラスのインスタンス;
```

例外を投げる可能性のあるメソッドには「throws 例外名」を付けておく必要がある

```
int methodA(int x) throws IOException {
  ...
  // 入出力処理に失敗したのでエラーを返したい
  throw new IOException();
  ...
}
```

例外の伝播

- 例外がスローされると，catchされるまで呼び出し元を辿っていく
- 例外をそのメソッドでcatchする必要のない場合throwsすることで呼び出し元に伝播



例外クラスの作成

- 例外はThrowableクラスを継承したクラス
  - 実際はException（もしくはRuntimeException）クラスを継承

```
public class NoElementException extends Exception {
}
```

例外クラスに独自の情報を持たせたい場合はフィールドやメソッド追加（なくてもよい）

総称型（Generic Types）

- 型だけが異なるが，それ以外の処理は同一のクラスを作成する場合がある
  - 整数を要素にもつリスト
  - 文字を要素にもつリスト
- すべてをコピペして型名だけ変えるのはバグの原因

型Tを要素にもつリストを作成しておき要素の型名だけを与えて作成すればよい

総称型の例

クラス名の直後に<名前>を記述
慣例として大文字一文字（TとかEとか）

```
public class Sample<T> {
    private T value;
    ...

    public void setValue(T value) {
        this.value = value;
    }

    public T getValue() {
        return value;
    }
    ...
}
```

利用例

```
Sample<String> s = new Sample<String>();
s.setValue("XYZ");
```

リストの宣言（総称型）

- elementはデータの値
- next は次の箱を指す
- head は先頭の箱を指す

```
public class LinkedList<E> implements List<E> {
    Node<E> head;
    ....
};
```

```
public class Node<E> {
    E element;
    Node<E> next;
};
```


クラスパス

- どこを起点にクラスを探すか
  - 初期値は「.」だけ（カレントディレクトリ）
  - JDK標準のクラスは設定にかかわらず参照可
- 環境変数CLASSPATHかオプション-cpで指定
  - デフォルト設定が上書きされるので注意
  - ディレクトリかjarファイルを指定
    - 複数指定する場合は;(Windows)か:(UNIX)で区切る

相対パスの場合カレントディレクトリの場所に依存するので注意

例)

```
% javac -cp ~/classes:library2.jar ms/gundam/ex/*.java
% java -cp ~/classes:library2.jar:. ms/gundam/ex/MainClass
```