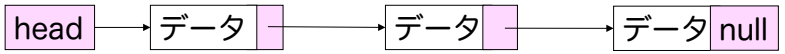
データ構造とプログラミング：リスト

情報工学科 山本哲男

リスト

- データを入れた箱（記憶場所）を並べたもの



- データの値を「要素」と呼ぶ
- 要素の個数を「リストの長さ」と呼ぶ

リスト（抽象データ型）

リストを抽象データ型で表すと

- 値
  - あるデータ型Tの要素を０個以上並べたもの
- 操作
  - 挿入：任意の位置に要素を追加
  - 削除：任意の位置の要素を削除
  - 取得：任意の位置の要素を取得

・・・

リストの操作

- 挿入：p番目に要素xを追加

```
void add(int p, T x)
```

- 削除：p番目の要素を削除

```
T remove(int p)
```

- 取得：p番目の要素を取得

```
T get(int p)
```

Tは任意のデータ型を表す

インタフェース

```
修飾子 interface インタフェース名 {
    // フィールド
    型名 フィールド名 = 式;
    ...
    // インタフェース
    戻り値の型 メソッド名(引数);
    ...
}
```

定数になる

メソッドの実装は記述しない

classとは違いメソッドの実装がないためインスタンス化できない
あくまで操作の一覧を記述しているだけ

インタフェースの例

```
public interface List {
    // インタフェース
    void add(int p, T x);
    T remove(int p);
    T get(int p);
}
```

Tは実在するデータ型にする
（数値ならint，オブジェクトならその型名など）

インタフェースの実装

- インタフェースだけでは動作しない
- インタフェースで定義されたメソッドの実装が必要
- インタフェース以外のメソッド等を追加して実装することも可能

```
public class ArrayList implements List {
    // フィールド
    private T elements[];
    ...
    // インタフェースの実装
    public void add(int p, T x) { ... }
    public T remove(int p) { ... }
    ...
    private int find(T x) { ... }
}
```


インタフェースを利用したプログラミング

- インタフェース型の変数には，そのインタフェースを実装したクラスのインスタンスが代入可能

```
List list = new ArrayList();
list.add(0, 100);
list.add(1, 10);
list.remove(0);
```

実装に対してではなく
インターフェースに対してプログラミングする

名前空間

- よく利用する名前はあちこちで作成されるため区別する必要がある
- 名前がどの集合（空間）に属するか指定することで一意に決める

山本さん？
どこの山本さん？
「日大工学部情報工学科の教員の」

集合（空間）

List？
どこのList？
「java.utilの」
とか
「org.example.testの」

パッケージ

- Javaにパッケージというメカニズムがある
- ソースファイルの先頭に
  `package パッケージ名;`
  と記述すると，どの空間のクラス（インタフェース）か指定可能

- 正式な名前は
  - パッケージ名.クラス名
  - パッケージ名.インタフェース名

  になるので注意

パッケージの例

```
package org.example.test;

public interface List {
    void add(int p, T x);
    T remove(int p);
    T get(int p);
}
```

org.example.test.Listが
正式なインタフェース名

パッケージ名はドメイン名を逆順にして利用することを推奨
（Javaの標準クラスはjava.で始まる）

正式名を省略して書くことも可能
ソースファイルにimport文を記述

import

- 先ほどのソースコードすべてに以下のパッケージ文が記述されていたとすると

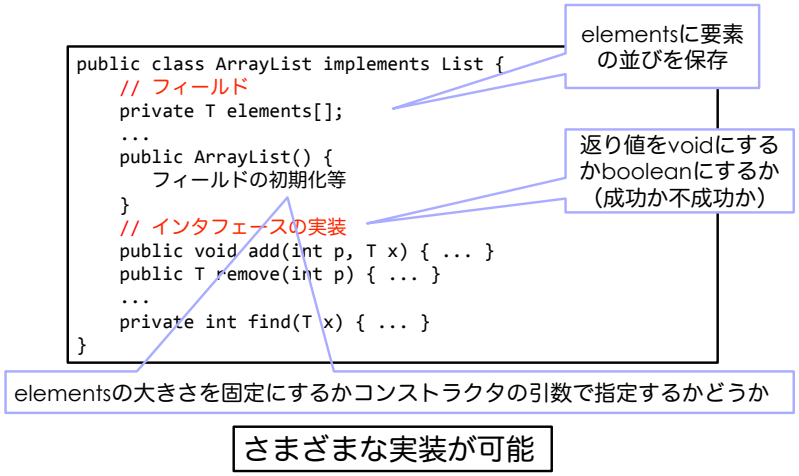
```
package org.example.test;
```

- 以下のように記述しなければならない

```
org.example.test.List list = new org.example.test.ArrayList();
list.insert(0, 100);
list.insert(1, 10);
list.remove(0);
```

正式名を省略して書くことも可能．ソースファイルにimport文を記述．
上記の例だと
```
import org.example.test.List;
import org.example.test.ArrayList;
```
を追加

配列を利用したListの実装



add

- インスタンス変数elementsに要素を追加していくだけ
  - 今いくつ保存しているかを表すインスタンス変数size等を利用すると便利
- リストの間に要素を追加する場合，追加する場所から後ろを一つずつずらす必要がある

addする際の注意点

- sizeがelements配列の大きさを超えた場合どうするか
- 配列は動的に大きさを変えられない
- sizeより大きい配列を確保し，要素をすべてコピーすることで実現
  - 倍の大きさを確保するのが一般的

remove

- 消したい要素より後ろを一つ前につめていくだけ

JAR

- JAR (JavaARchive) という圧縮ファイル
  - 圧縮ファイルの形式はzipと同じ

圧縮

```
% jar cf jarファイル名 ファイル名 or ディレクトリ
```

伸張

```
% jar xf jarファイル名
```

実行

```
% java -cp jarファイル名 mainクラス名
```

MANIFESTファイルを作成しjarにすることで自動実行も可

MANIFEST

- 以下のようなテキストファイルを作成し，jarファイル作成時に指定

```
Main-Class: ms.gundam.ex1.Main
```

mainメソッドがあるクラス名を指定

作成

```
% jar cfm jarファイル名 作成したマニフェストファイル名 ファイル名 or ディレクトリ
```

実行

```
% java -jar jarファイル名
```

jarファイル名のみが分かればよいので，jarファイルをクリックするだけで自動実行することも可能

submitするJARファイル

- 実行可能なJARファイルを作成すること
  - classファイルだけではなく，javaのソースファイルも一緒にまとめること