

# データ構造とプログラミング： 抽象データ型

情報工学科 山本哲男

## 目標

- 抽象データ型の概念についての理解
- 連結リスト・マップ・木構造およびそれ  
を操作するためのアルゴリズムについて  
理解し、Java言語を用いて実装
- 上記で実装したデータ構造を利用したプ  
ログラムをJava言語を用いて実装

## データ構造とプログラミング および演習

- 「データ構造とプログラミングおよび演習」は、  
講義と演習がセットとなっている3単位の必修科目
- 【講義】
  - 曜日と校時：月曜日 4校時
  - 場所：61号館 2階演習室
- 【演習】
  - 曜日と校時：月曜日 5校時
  - 場所：61号館 2階演習室
- 担当教員
  - 山本 哲男 (61号館403室)
    - [tetsuo@cs.ce.nihon-u.ac.jp](mailto:tetsuo@cs.ce.nihon-u.ac.jp)
    - @ytetsuwo

## 演習の概要

- 授業では講義を行いその後講義で学んだ  
ことを応用できるようになるため演習を  
実施
- 演習問題で指示されるプログラムの作成

<http://dsap.cse.ce.nihon-u.ac.jp/> 提出サイトを利用

Registration Code は 61403

Usernameは演習室のユーザ名 (u256xxx)

Emailは任意のメールアドレス

## 授業計画（予定）

変更の可能性あり

1. 抽象データ型
2. リスト構造（配列）
3. リスト構造（連結リスト）
4. 単体テスト
5. スタック・キュー
6. ハッシュその1
7. ハッシュその2
8. 中間試験
9. 木構造
10. 二分探索木その1
11. 二分探索木その2
12. その他の木構造（赤黒木）その1
13. その他の木構造（赤黒木）その2
14. まとめ
15. 授業内試験

## 進め方

### ■ 講義の進め方

- スライドを用いて通常の講義形式で行う
- 講義中に練習問題を出すこともある
- 中間試験（筆記）行う

### ■ 演習の進め方

- 数題の課題をプログラミングしテスト
- プログラムを提出
- 中間試験（実技）を行う

## 成績評価

- **レポートを全て提出**していることが単位修得のための必要条件  
レポート内容に不足等があったら再提出を求める
- **4回以上欠席**すると成績評価を行わない
- レポートを10点，中間試験（筆記と実技）を30点，定期試験を60点として，100点満点中60点以上が単位修得のための必要条件

## 履修に関する注意

- 『WWWとJavaプログラミングおよび演習』の単位を取得している必要がある
- 基本的なJavaプログラムが記述可能であること
  - クラス，オブジェクトの概念
  - 継承，例外処理の概念
  - 複数のクラスを利用したプログラム

## 復習

## 変数

- 変数：データを入れておく箱
  - 整数型変数
    - int型
    - 整数だけを格納できる
  - 浮動小数点型変数
    - double型
    - 小数点のある値も格納できる

2015/9/15

データ構造とプログラミング

9

## データ型（プリミティブ）

	データ型	ビット長	範囲
真偽	boolean	1	true, false
整数	byte	8	-128 ~ 127
	short	16	-32768 ~ 32768
	int	32	-2147483648 ~ 2147483648
	long	64	-9223372036854775808 ~ 9223372036854775807
文字	char	16	\u0000 ~ \uFFFF
浮動小数点数	float	32	単精度浮動小数点数 -3.4028235E+38 ~ -1.401298E-45
	double	64	倍精度浮動小数点数 -1.79769313486231570E+308 ~ -4.94065645841246544E-324
型なし	void		

2015/9/15

データ構造とプログラミング

10

## 配列

- 配列：同じ種類のデータをまとめて扱う
- データの型がintなら（xは変数名）  
int[] x;
- データの型がStringなら（strは変数名）  
String[] str;  
と宣言して利用可能

この時点で配列の実態は作成されていない事に注意

2015/9/15

データ構造とプログラミング

11

## 配列の確保

- 配列を使うときには配列自体を作成する必要がある（※後から要素数変更不可）

例：  
x = new int[10]; ← int型で要素数10個の配列xを作成  
str = new String[size]; ← String型で要素数size個の配列strを宣言

配列もオブジェクト

str.lengthで配列の要素数  
を取得可能

## 多次元配列

- 配列は「多次元」にできる

[0] [1] [2] [3] [4]					new int[5][4];																	
[0]					[0]	[1]	[0]	[2]	[0]	[3]	[0]	[4]										
[1]					[1]	[1]	[1]	[2]	[1]	[3]	[1]	[4]										
[2]					[2]	[1]	[2]	[2]	[2]	[3]	[2]	[4]										
[3]					[3]	[1]	[3]	[2]	[3]	[3]	[3]	[4]										

1次元配列

2次元配列

「1次元配列」は「ベクトル」  
「2次元配列」は「行列」  
をイメージすると良い

注意：[2, 3]のようには書かない

3次元以上も可能

## クラス

```
修飾子 class クラス名 {  
    // フィールド  
    修飾子 型名 フィールド名;  
    ...  
    // コンストラクタ  
    修飾子 クラス名(引数) {  
        ...  
    }  
    ...  
    // メソッド  
    修飾子 戻り値の型 メソッド名(引数) {  
        ...  
        return 式;  
    }  
    ...  
}
```

## 抽象データ型

- データ構造とそれを直接操作する手続きをまとめたデータ型
  - データ構造
  - 操作する手続き
- 抽象データ型を利用する側は「操作する手続き」のみを利用する

## 抽象データ型の注意点

- データ構造の情報内容と操作する手続きのみを気にすればよい
  - どのような型定義や実装によって実現されているか
  - を気にしてはいけない、気にする必要もない
- 操作する手続き（インタフェース）のみに依存

## スタックの例（C言語）

```
#define STACK_SIZE 100

typedef int data_t; /* スタックに貯えるデータの型 */
data_t stack[STACK_SIZE]; /* スタック本体 */
int stack_num; /* データ数 */

int push(data_t data) {
    if (stack_num < STACK_SIZE) {
        stack[stack_num] = data;
        stack_num++;
        return 0;
    } else {
        return -1;
    }
}

data_t pop() {
    if (stack_num > 0) {
        stack_num--;
        return stack[stack_num];
    } else {
        return -1;
    }
}
```

データ構造

操作する手続き

## スタックの例（Java言語）

```
class stack {
    private static final int STACK_SIZE = 100
    private int stack[STACK_SIZE]; /* スタック本体 */
    private int stack_num; /* データ数 */

    public int push(data_t data) {
        if (stack_num < STACK_SIZE) {
            stack[stack_num] = data;
            stack_num++;
            return 0;
        } else {
            return -1;
        }
    }

    public int pop() {
        ...
    }
}
```

## Javaにおける抽象データ型

- Javaのクラスは抽象データ型
  - データ構造 = フィールド
  - 操作する手続き = メソッド
- 単なるメソッドの集合がクラスではない
- 抽象データ型と考えることで適切なクラス設計を！

## カプセル化

- あらゆる物を隠蔽すること
  - 実装
  - 型
  - . . .
- データ構造の「型」や操作する手続きの「実装」を隠す
- カプセル化 ≡ 情報隠蔽

## 修飾子

- アクセス修飾子
  - private 自クラス
  - protected サブクラスか同一パッケージ
  - public すべてのクラス

## コンストラクタ

- オブジェクト生成時に実行される処理
  - フィールドの初期設定等

```
private int num;
private int gas;

public Car() {
    num = 0;
    gas = 0;
}

public Car(int num, double gas) {
    this.num = num;
    this.gas = gas;
}
```

コンストラクタの  
オーバーロード

## デフォルトコンストラクタ

- コンストラクタの記述を省略すると引数のない空（処理がない）のコンストラクタが自動生成

```
private int num;
private int gas;

public Car() {
}
```

## クラス変数, クラスメソッド

- フィールドはインスタンス毎に生成
- ただし, **static**修飾子をつけるとクラス変数となり読み書きする場所は一つ
  - クラス全体で共通に扱うデータ
- メソッドに**static**をつけるとクラスメソッドと呼び, インスタンスを生成しなくても呼び出し可能

## 標準入出力

- 標準出力 `System.out`
- 標準入力 `System.in`

Systemクラスのoutフィールドとinフィールド型はそれぞれPrintStream型とInputStream型

## 標準入力 (その1)

- `System.in.read`メソッドで一文字 (バイト) 毎の入力が可能
- バイト単位ではなく文字列や数値として読み込みたい
  - `InputStreamReader` バイトから文字へ変換
  - `BufferedReader` バッファリングすることで効率よく

```
BufferedReader in =  
    new BufferedReader(new InputStreamReader(System.in));  
String str = in.readLine(); // 一行読み込み
```

## 標準入力 (その2)

- `java.util.Scanner`クラスを利用すると, プリミティブ型や文字列の入力が容易に
  - `nextInt()` `int`型
  - `next()` `String`型
  - . . .

```
Scanner sc = Scanner(System.in);  
  
int i = sc.nextInt(); // scanf("%d", &i) に似ている  
String str = sc.next(); // scanf("%s", str) に似ている
```

## Javadoc

- JavaのソースコードからHTML形式のAPI仕様書を生成
- クラス名やメソッド名の直前に

`/** */`

形式のコメントを記述

```
javadoc -private -d doc ooooo.java
```

docというディレクトリ内にHTML形式のドキュメントが生成される

## Javadoc形式のコメント例

```
/**
 * 長方形を表すクラス
 * 複数行のコメントが記述可能
 * @author Tetsuo Yamamoto
 */
public class Rectangle {
    ...
    /**
     * 長方形のサイズ設定
     * @param width 幅
     * @param height 高さ
     */
    public void setSize(int width, int height){
        ...
    }
    /**
     * 長方形の高さ取得
     * @return 高さ
     */
    public int getHeight() {
        ...
    }
}
```

## モデルとビューの分離

- モデル（ロジック・ビジネスロジック）
  - アプリケーションが扱う領域のデータと処理を表す要素
    - 計算処理・データの加工/保存など
- ビュー
  - モデルの処理の結果を適した形で出力する要素
    - CLIの出力・GUIの画面など

モデルとビューのクラスを分離することで

- ・ ソースコードが見やすくなる
- ・ UIの入れ替えが容易

その結果、保守性が向上する