知識工学

岡山大学大学院

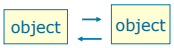
講師　竹内孔一

本日の内容

- 知識のモデル化
  - オブジェクト指向

オブジェクト指向

- オブジェクト指向
  - 60年代から70年代
    - オブジェクトという独立要素どうしの通信でプログラミング
  - ダイナブック（60年代）
    - Alan Kay 大学ノート大のパソコン構想，小学生も扱える
    - ユーザ言語 Smalltalk (のちに smalltalk-80)
      メッセージのみで対象を操作する
  - オブジェクト
    - 通信手段をもつ
    - クラスとインスタンスで整理
    - メソッドという属性を持つ

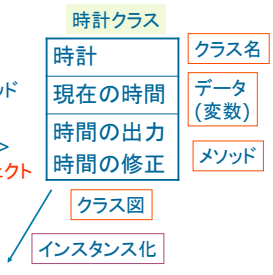


オブジェクト指向

- 考え方
  - オブジェクトという独立要素どうしの通信で全体として機能させる
- 利点
  - モノとモノどうし関係をそのまま記述
  - あとから追加しやすい（手続き型言語との違い）
  - シュミレーションできる
- 注意
  - 解を得る方法は別(探索など)
- プログラミング言語(Win/Linux/Mac/Solaris)
  - smalltalk, Java, C++,Ruby, PHP, C#(Winのみ)
- 設計のための記述
  - UML (unified modeling language)

オブジェクト指向

- 基本要素
  - クラス
    - クラス名，データ，メソッド
  - クラスとインスタンス
    - クラスを具体化したも->
      インスタンス->オブジェクト
- 性質
  - データ抽象化
  - 多相性



tokei = new 時計　(Java風の記述)

練習11

- 電波時計のオブジェクトについて
  - クラス名，データ，メソッドについて考えよ

オブジェクト指向(記述例)

- Javaを例に

```
class Doumo { //クラス名
        //変数
        public String doumodesu; //修飾子，型，変数名
        //メソッド
        void kotoba(String words) {
                doumodesu=words; //変数にセット
        }
}
```

修飾子: public, private などでアクセスを指定
→ カプセル化

```
class Hello {
        public static void main(String[] args) {
                System.out.println("Hello from Java");
                //インスタンス化↓
                Doumo takashi=new Doumo();
                //メソッド呼び出し↓
                takashi.kotoba("どうもどうも");
                //変数へのアクセス↓
                System.out.println(takashi.doumodesu);
        }
}
class Doumo {
        //変数
        String doumodesu;
        //メソッド
        void kotoba(String words) {
                doumodesu=words; //変数に引数をセット
        }}
```

main関数

画面に出力

Scalaだとこんな感じ

```
object MyHello {  //mainはオブジェクト
  def main(args: Array[String]) {
    println("Hello world")      //セミコロン不要
    val takashi= new Doumo("")
    takashi.kotoba("ハローどうもどうも")  //型は自動
    println(takashi.doumodesu)
  }
}

class Doumo (var doumodesu:String) {
    //いちいち内部変数書かなくて良い
    def kotoba(words:String) {
        doumodesu=words
    }
}
```

クラス(型)を使う良い点

- 型のチェックで人の誤りを防ぐ
  - 人が思っていた型と違う=>プログラムミス
- 型の持つメソッドを利用することでコードが減らせる

Javaの例で　takahashi.doumodesu.length()　=>6

Doumo型

String型

int型

6という値
"どうもどうも"の文字数

Rubyだと
"どうもどうも".length　と文字列に操作できる

10

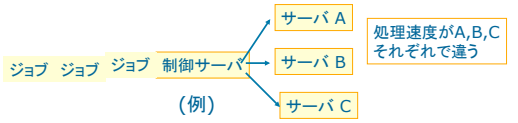
練習12

- 実行結果として「どうもどうも」と出力されるように□を埋めよ

```
class Hello {
        public static void main(String[] args) {
                System.out.println("Hello from Java");
                Doumo takahashi=new Doumo();
                takahashi.kotoba("どうもどうも");
                System.out.println(  □  );
        }
}
class Doumo {
        String doumodesu;
        void kotoba(String words) {
                doumodesu=words;
        }}
```

具体例からオブジェクトの設計

- グリッド問題
  - ジョブが次々と入ってきてサーバに渡す
  - 制御サーバ，計算サーバのクラスを考えてみよう



(例)

制御サーバ
データ: ?
メソッド: ?

サーバ
データ: ジョブID
メソッド: ジョブの処理，処理状況の報告
処理終了の報告

オブジェクト指向

- オブジェクト指向の設計　UMLに従った記述
  ユースケース図
  - 作ろうとしているシステムの機能を整理する
  - 機能に基づいてオブジェクトを設計

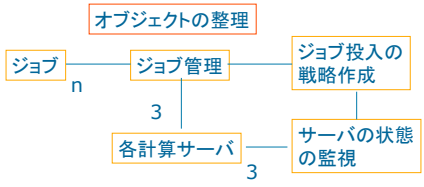
  クラス図
  - 必要なオブジェクトを書き出し，データとメソッドについて整理する

グリッド問題の例

ユースケース分析

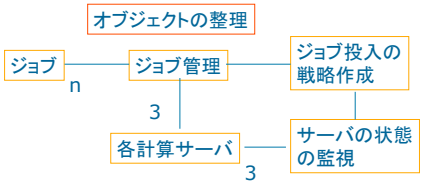
- 制御サーバの機能
  - ジョブの管理
  - ジョブ投入の戦略
  - 各サーバの特徴
  - 各サーバの状態の把握

←制御サーバの機能を
より詳細に分解



練習13

- グリッド問題を分析した下記オブジェクト図において，各オブジェクトのデータ，メソッドを記述せよ



練習13のヒント