プログラミング応用
http://bit.ly/kosen02

Week5@後期
2016/10/26

本日の内容

- 講義：
  - C++によるオブジェクト指向プログラミング (クラス宣言、継承)
  - UMLを用いたソフトウェアモデリング (クラス図、ユースケース図)
  - デザインパターン(Composite Pattern)
- 演習：
  - C++によるオブジェクト指向家計簿アプリの作成
  - Astahを用いたUML記述演習

本日の内容

- 講義：
  - C++によるオブジェクト指向プログラミング (クラス宣言、継承)
  - UMLを用いたソフトウェアモデリング (クラス図、ユースケース図)
  - デザインパターン(Composite Pattern)

- 演習：
  - C++によるオブジェクト指向家計簿アプリの作成
  - Astahを用いたUML記述演習

オブジェクト指向プログラミング

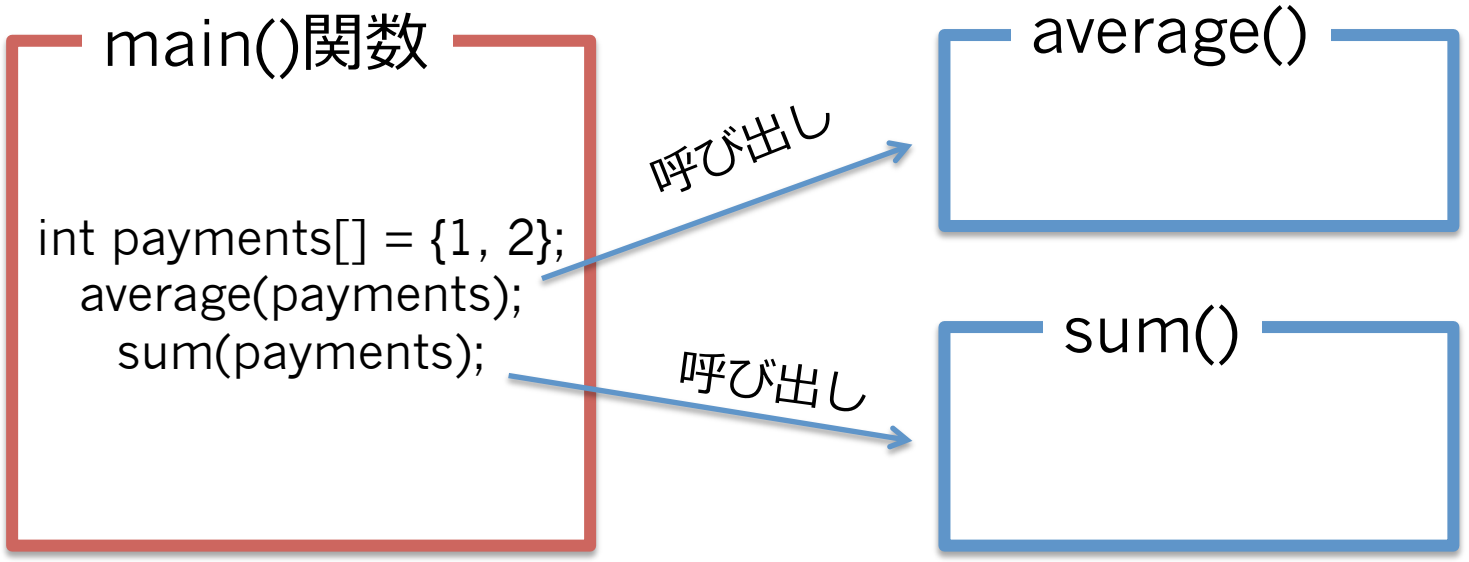
- オブジェクト指向プログラミング
  - クラスを組み合わせてプログラムを作る技法

    ※これまでは関数を組み合わせてプログラムを作っていた

- クラス
  - 関数＋変数をまとめて記述する仕組み

例) 家計簿アプリ

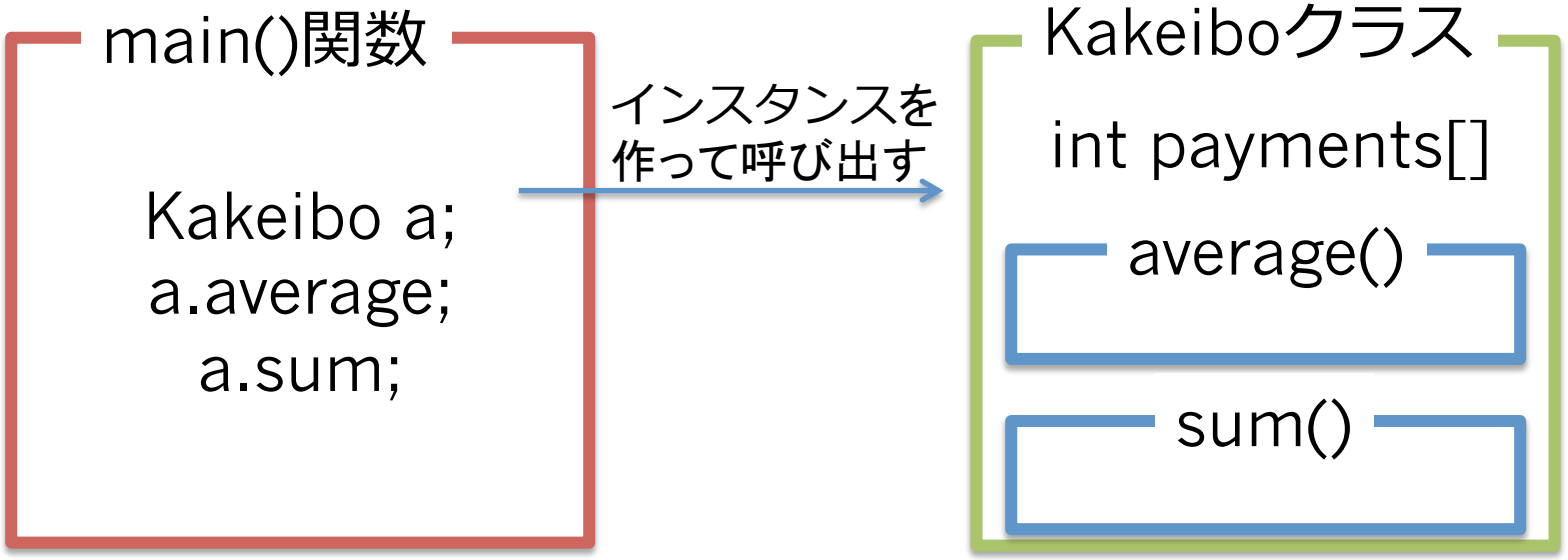
- これまでの家計簿アプリ
  (関数を組み合わせて作る設計)



大規模なソフトウェアで変数や関数の数が
増えると管理が大変になる

例）家計簿アプリ

- オブジェクト指向
  (=クラスを組み合わせて作る設計)



クラスはC言語の構造体と関数がセットになったものとも考えられる

C++によるクラス実装と利用

- C++：クラスが使えるCの拡張言語
- プログラムの拡張子は.cpp
  例) kakeibo.cpp
- コンパイルは以下のコマンドで行う
  $ gcc kakeibo.cpp

C++によるクラスの記述例

```
 1 #include <stdio.h>
 2
 3 class Kakeibo {
 4   // ここにクラスの定義を書く
 5   public:
 6     // 1. メンバ変数宣言
 7     int payments[1000]; // 支出を記録する配列
 8     // 2. コンストラクタ宣言
 9     Kakeibo() {
10       // クラスを呼び出した際に1度だけ実行される処理
11     }
12     // 3. メンバ関数宣言
13     void pay(int amount) {
14       // payが呼び出されたときの処理
15       printf("%d円支出しました\n");
16     }
17 };
18 int main()
19 {
20     Kakeibo a;     //  Kakeiboクラスを呼び出しaという名前にする
21                    // このaをインスタンスという
22     a.pay(100); // pay()メンバ関数を呼び出す
23     return 0;
24 }
```

C++によるクラスの記述例

クラスの宣言部分

クラスの呼び出し部分

```
1  #include <stdio.h>
2
3  class Kakeibo {
4    // ここにクラスの定義を書く
5    public:
6      // 1. メンバ変数宣言
7      int payments[1000]; // 支出を記録する配列
8      // 2. コンストラクタ宣言
9      Kakeibo() {
10       // クラスを呼び出した際に1度だけ実行される処理
11     }
12     // 3. メンバ関数宣言
13     void pay(int amount) {
14       // payが呼び出されたときの処理
15       printf("%d円支出しました\n");
16     }
17 };
18 int main()
19 {
20     Kakeibo a;     //  Kakeiboクラスを呼び出しaという名前にする
21                    // このaをインスタンスという
22     a.pay(100); // pay()メンバ関数を呼び出す
23     return 0;
24 }
```

C++によるクラスの記述例(クラス部分)

```
1  #include <stdio.h>
2
3  class Kakeibo {
4    // ここにクラスの定義を書く
5    public:
6      // 1. メンバ変数宣言
7      int payments[1000]; // 支出を記録する配列
8      // 2. コンストラクタ宣言
9      Kakeibo() {
10       // クラスを呼び出した際に1度だけ実行される処理
11     }
12     // 3. メンバ関数宣言
13     void pay(int amount) {
14       // payが呼び出されたときの処理
15       printf("%d円支出しました\n");
16     }
17 };
```

1. メンバ変数定義

2コンストラクタ

3. メンバ関数定義

メンバ変数定義はC言語での方法と同じ。この変数はクラス内のどの関数からも呼び出すことが出来る。
コンストラクタはmain()関数からクラスを呼び出したときに1度だけ実行される処理。変数の初期化などを行う。
メンバ関数定義ではC言語での関数定義と同じように関数を記述。

C++によるクラスの呼び出し例 (呼び出し部分)

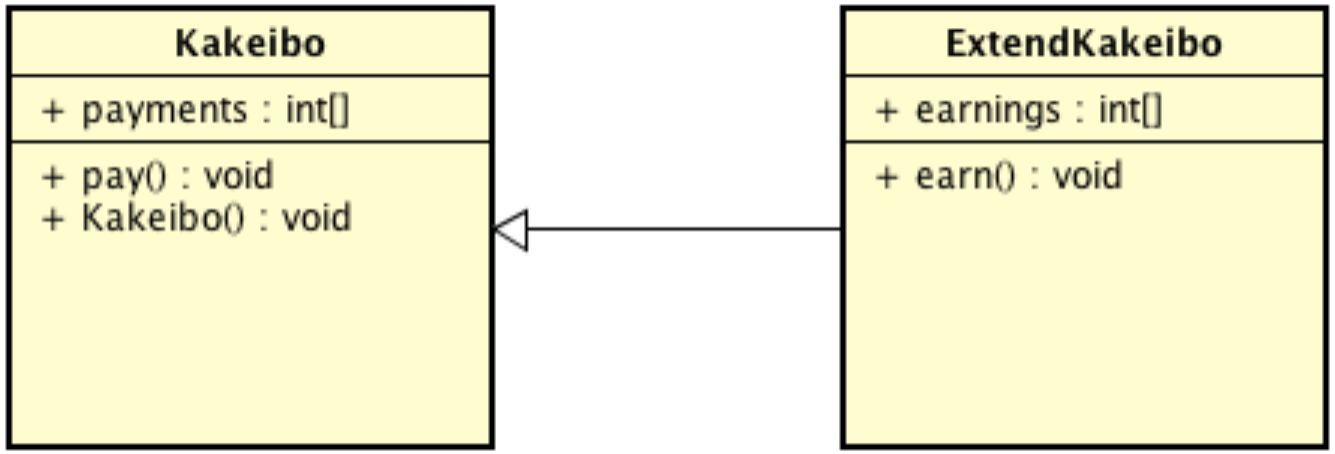
```
1  #include <stdio.h>
2
3  class Kakeibo {
4    // ここにクラスの定義を書く
5    public:
6      // 1. メンバ変数宣言
7      int payments[1000]; // 支出を記録する配列
8      // 2. コンストラクタ宣言
9      Kakeibo() {
10       // クラスを呼び出した際に1度だけ実行される処理
```

C言語で構造体を使うときと同じように以下の形式で宣言
クラス名 任意の名前(インスタンス名);
宣言するとまず初めにコンストラクタが実行される

```
17 };
18 int main()
19 {
20     Kakeibo a;     //  Kakeiboクラスを呼び出しaという名前にする
21                    // このaをインスタンスという
22     a.pay(100); // pay()メンバ関数を呼び出す
23     return 0;
24 }
```

メンバ関数やメンバ変数を呼び出すときには
インスタンス名.関数名or変数名

継承



- 継承
  - クラスが提供する機能の1つ
  - Kakeiboクラス(親)からExtendKakeiboクラス(子)を作ることができる
  - 親クラスのメンバ関数/変数が子クラスへ引き継がれ、ソースコードの再利用ができる

C++での継承の記述例

```
class ExtendKakeibo : public Kakeibo {
  public:
    int earnings[1000];
    void average() {
      // 支出の平均値を求めて表示する処理を書く
      int i=0;
      int sum = 0;
      while(payments[i] != -1) {
        sum += payments[i];
        i++;
      }
      printf("支出平均：%d\n", sum/i);
    }
};
```

C++での継承の記述例

```
class ExtendKakeibo : public Kakeibo {
```

class 子クラス名 : public 親クラス名　{

```
    void average() {
      // 支出の平均値を求めて表示する処理を書く
      int i=0;
      int sum = 0;
      while(payments[i] != -1) {
        sum += payments[i];
        i++;
      }
      printf("支出平均：%d\n", sum/i);
    }
};
```

C++での継承の記述例

```
class ExtendKakeibo : public Kakeibo {
  public:
    int earnings[1000];
    void average() {
      // 支出の平均値を求めて表示する処理を書く
      int i=0;
      int sum = 0;
      while(payments[i] != -1) {
        sum += payments[i];
        i++;
      }
      printf("支出平均：%d\n", sum/i);
    }
};
```

この部分は通常のクラス定義と同様

本日の内容

- 講義：
  - C++によるオブジェクト指向プログラミング (クラス宣言、継承)
  - UMLを用いたソフトウェアモデリング (クラス図、ユースケース図)
  - デザインパターン(Composite Pattern)

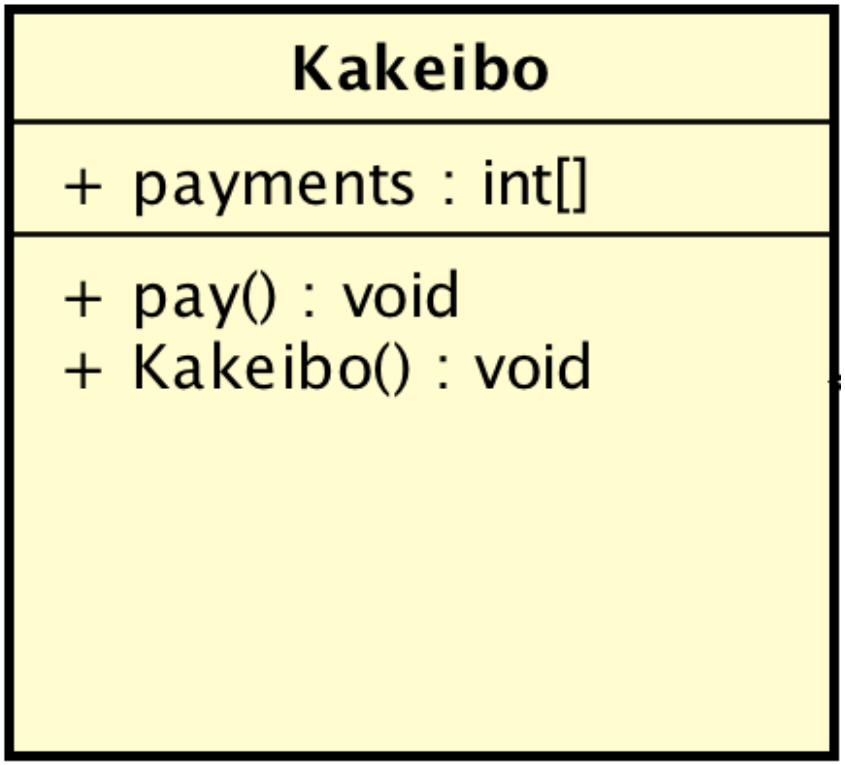
- 演習：
  - C++によるオブジェクト指向家計簿アプリの作成
  - Astahを用いたUML記述演習

UML

- UML; Unified Modeling Language
  - ソフトウェアの振る舞いを図として表現するための枠組み
  - UMLでは多くの図があるが、本講義ではクラス図、ユースケース図にしぼって説明
- Astahをいうソフトを使いUMLを記述する演習を行う

クラス図(1/2)

- クラス設計を表現する図

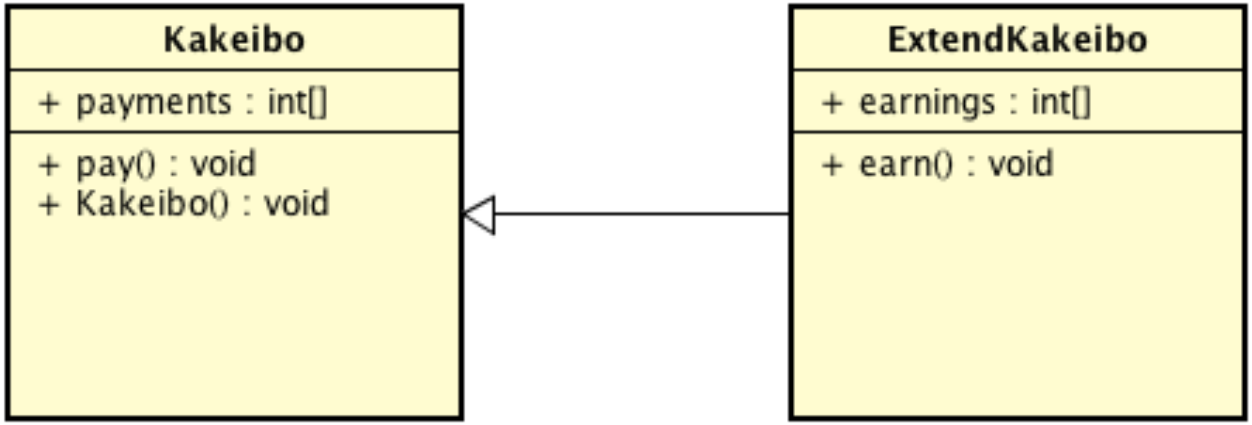


1. クラス名
2. + メンバ変数：型名
3. + メンバ関数：戻り値の型

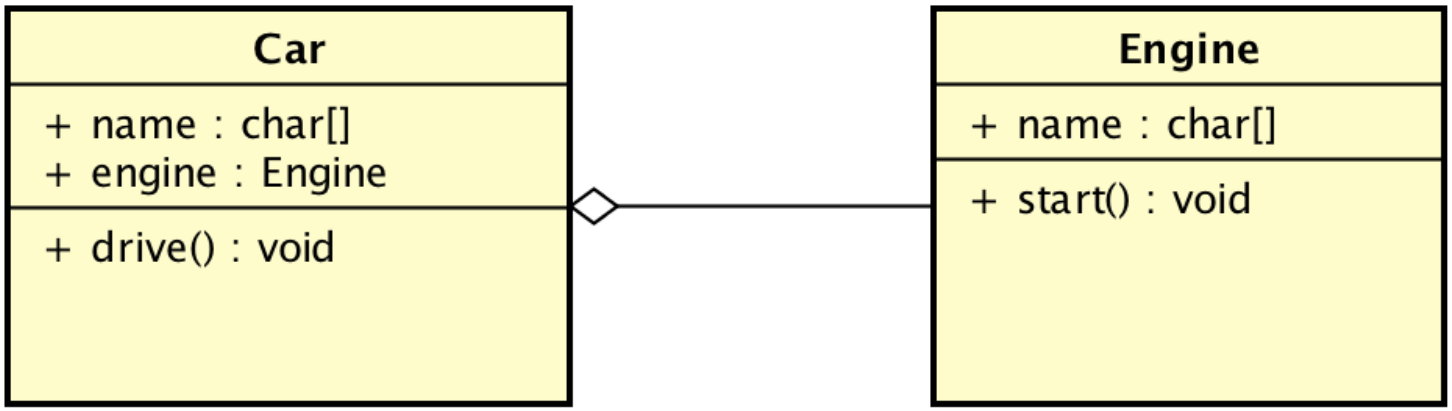
+記号については4/5年
選択科目で学習する

クラス図(2/2)

- クラス間の関係を表現する方法
  - 継承(端点が三角形の矢印)

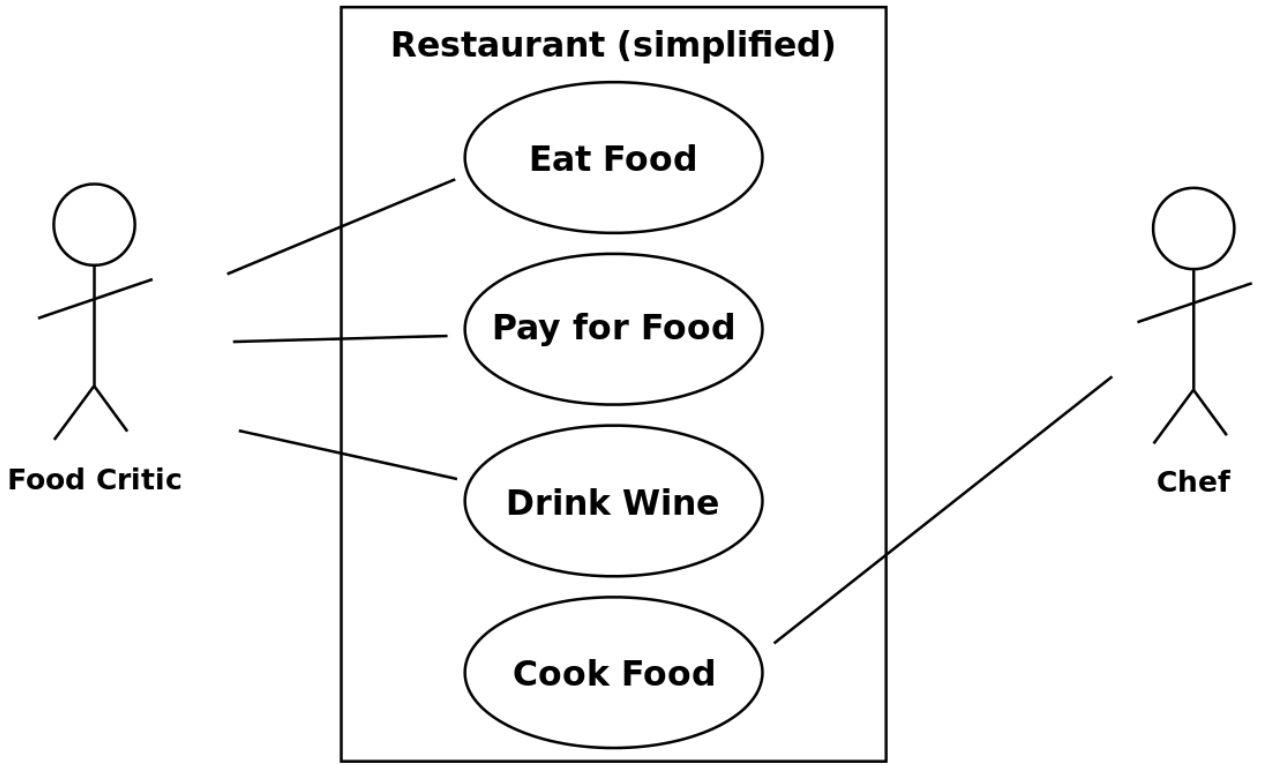


  - コンポジション(xはyを持っている関係)



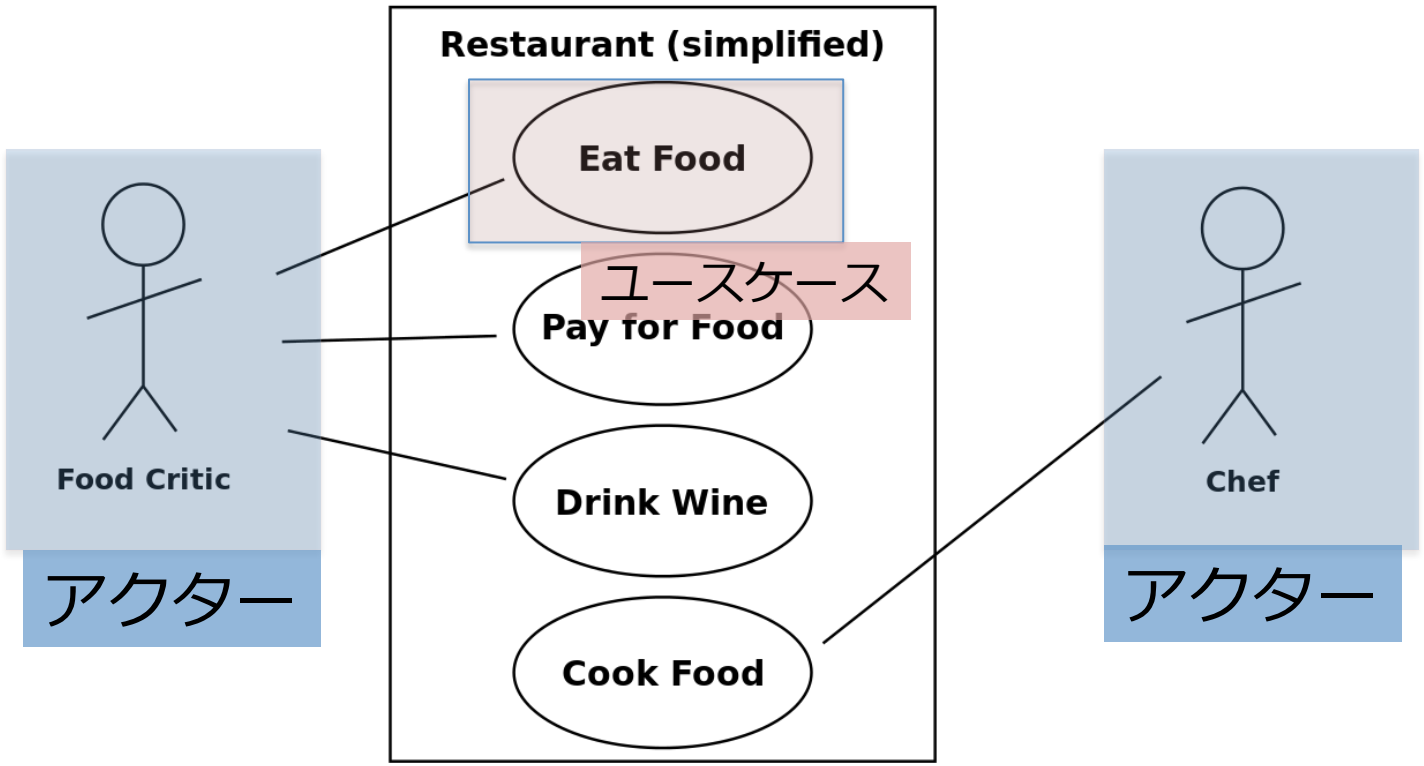
ユースケース図

- ユーザーから見たソフトウェアの振る舞いを表現する図



ユースケース図

- ユーザーから見たシステムの振る舞いを表現する図



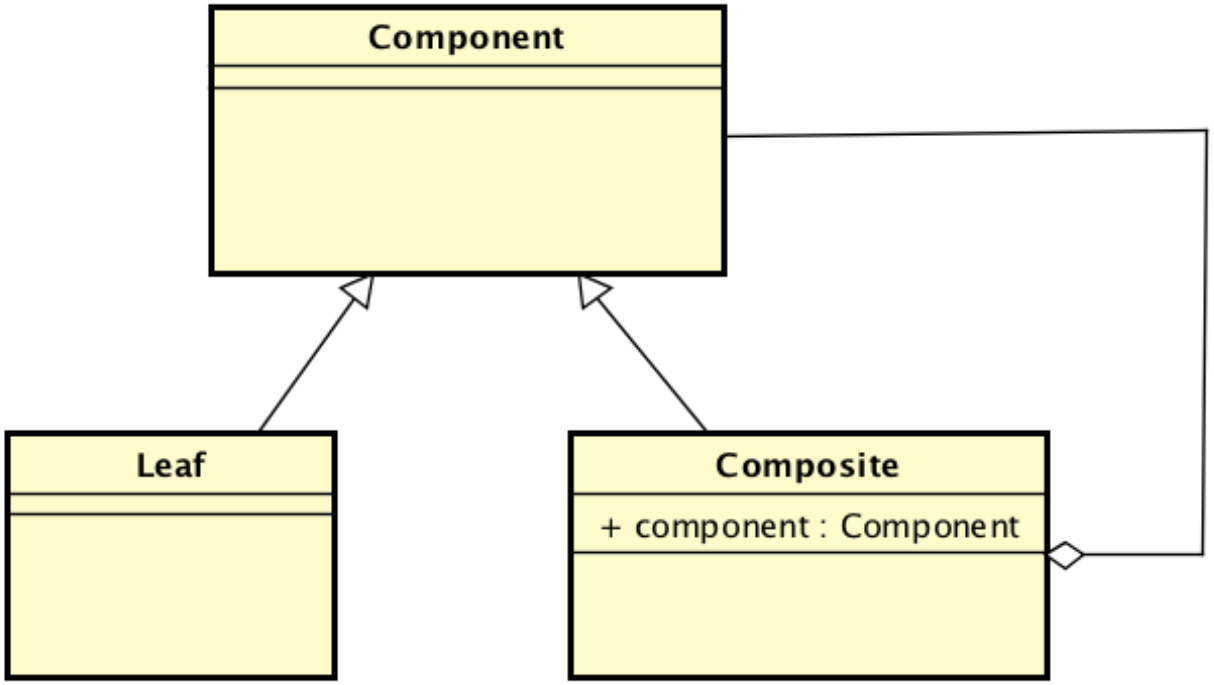
本日の内容

- 講義：
  - C++によるオブジェクト指向プログラミング (クラス宣言、継承)
  - UMLを用いたソフトウェアモデリング (クラス図、ユースケース図)
  - デザインパターン(Composite Pattern)
- 演習：
  - C++によるオブジェクト指向家計簿アプリの作成
  - Astahを用いたUML記述演習

デザインパターン

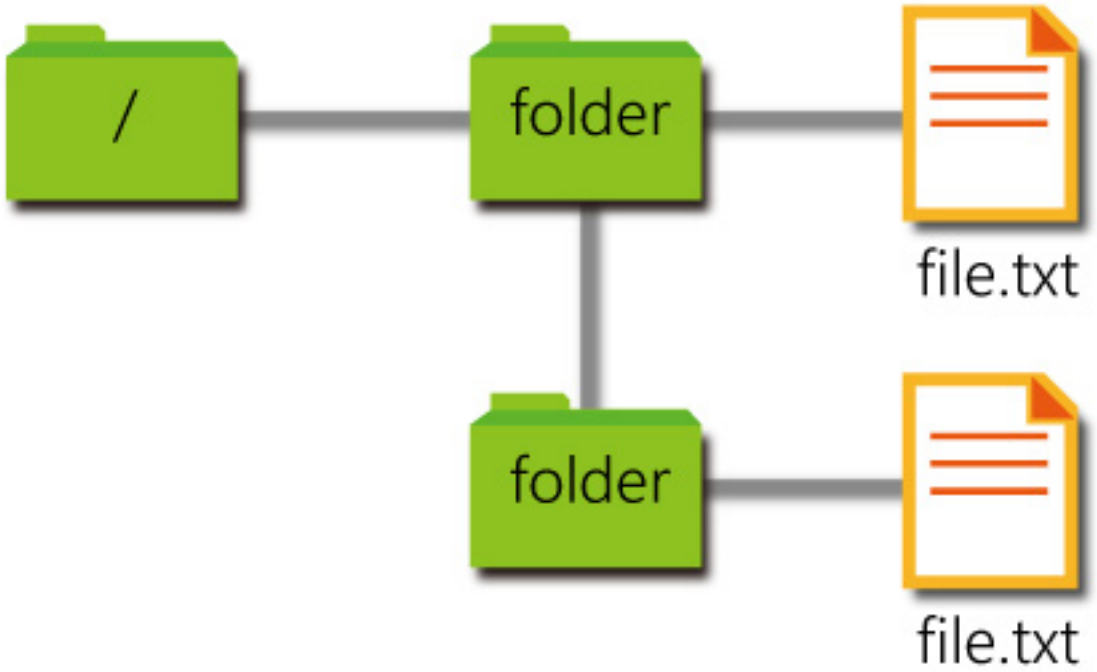
- 「どのようにクラスを設計するか」は、難しい問題
  - 何をメンバ関数/メンバ変数にする？
  - 何を親クラス/子クラスにする？
  - どのクラスをコンポジションの関係にする？
- デザインパターン
  - よく使われる設計パターンをまとめたもの
  - 本講義ではよく使われるパターンの1つ「Composite Pattern」に絞って解説

Composite Pattern



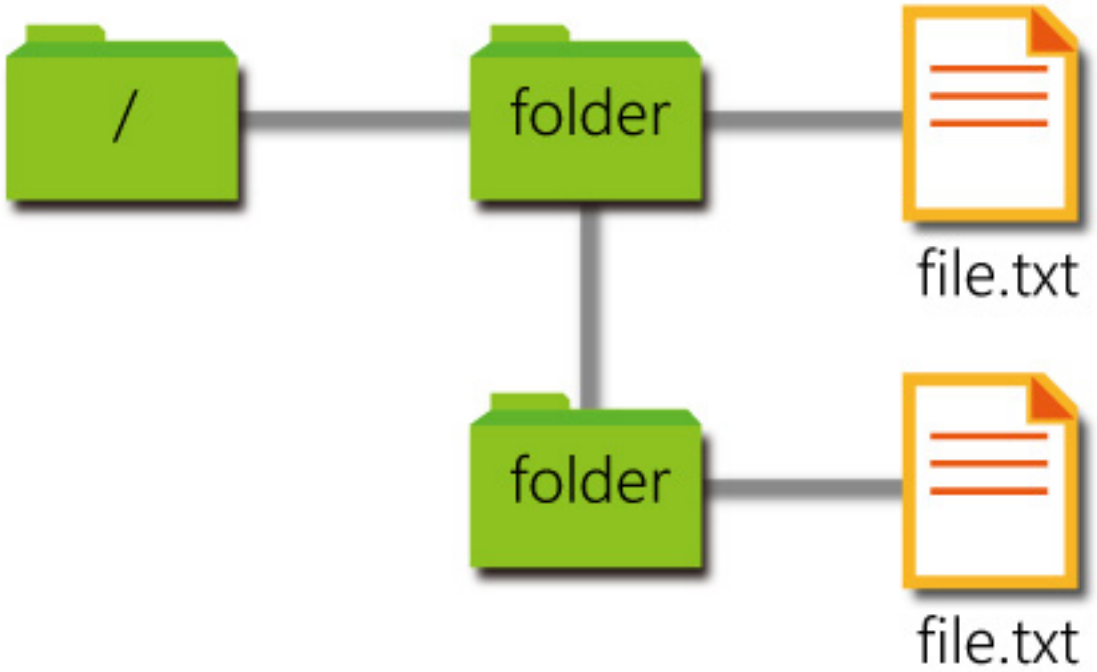
- Component/Leaf/Compositeという3つのクラスから構成されるデザイン
- Leaf/CompositeはComponentを継承、
- CompositeはComponentをメンバ変数として持つ

例) ファイルシステム(1/2)



Directory/Fileどちらも名前(name)を持つ。
Directory：Directory内にはDirectoryかFileを持つ
File　　：何も持たない

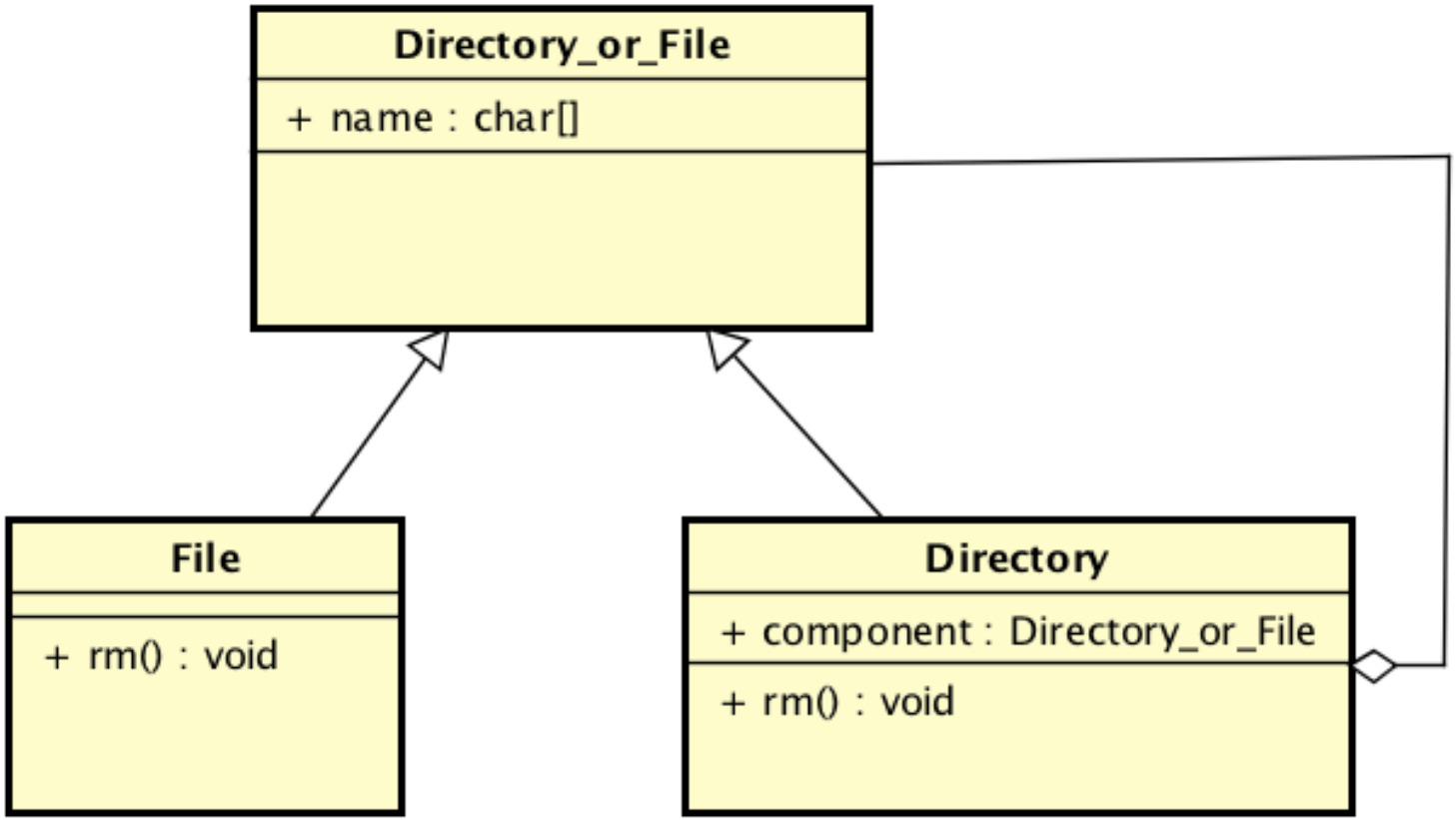
例) ファイルシステム(2/2)



rmコマンドで削除する場合：
File:　　そのファイルだけを削除
Directory: 指定したディレクトリを削除
　　ディレクトリ以下のファイルをすべて削除

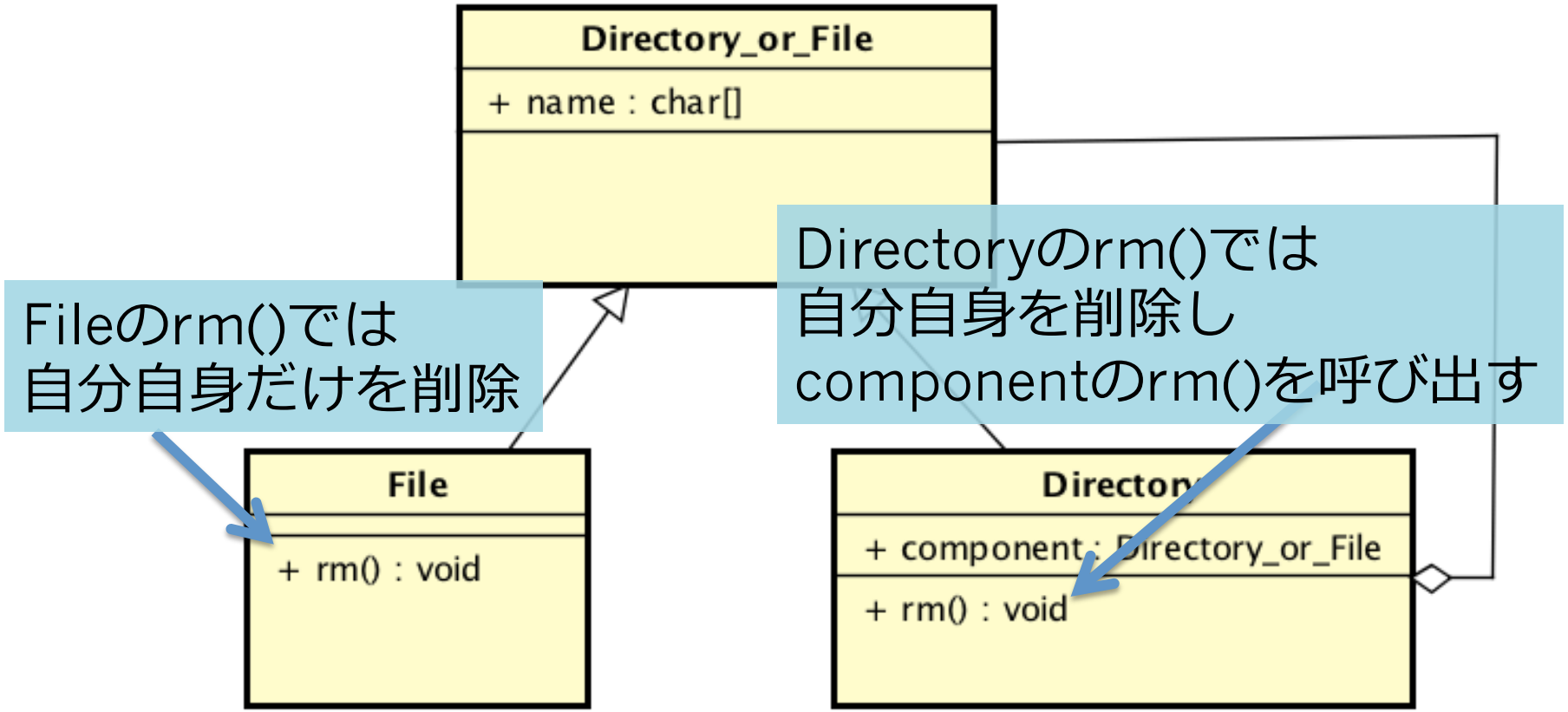
ファイルシステムとComposite Pattern

- ファイルシステムへのComposite Patternの適用



ファイルシステムとComposite Pattern

- ファイルシステムへのComposite Patternの適用



本日の演習

- 演習1–6：クラスを使った家計簿アプリを作成
- 演習7：Astahでクラス図を作成

- 演習8(optional)：
  Astahによるユースケース図作成
- 演習9(optional)：
  継承を使った家計簿アプリの機能拡張

家計簿アプリ演習

- 本演習ではmain()関数に以下を記述すると、出力例のように表示されるプログラムを作成する

main()関数の記述

```
int main() {
    Kakeibo a;
    a.pay(100);
    a.pay(200);
    a.pay(500);
    a.show_payments();
    a.average();
    a.sum();
    return 0;
}
```

出力例

```
100円の支出を記録しました。
200円の支出を記録しました。
500円の支出を記録しました。
これまでの支出を表示します。
0：100円
1：200円
2：500円
支出平均：266
支出合計：800
```

演習0

- c-programming-stxxdyyディレクトリに移動し、kakeibo.cppというファイルを作成

- 前ページのmain()関数及び#include <stdio.h>を記述する

演習2

- Kakeiboクラスを以下のように作成
  - メンバ変数：
    int payments[1000]; // 支払い一覧を格納
  - コンストラクタ(中身は空で良い)
    Kakeibo() {}
  - メンバ関数(中身は空で良い)：
    void pay(int amount) {}
    void show_payments() {}
    void average() {}
    void sum() {}

演習3：コンストラクタ

- コンストラクタKakeibo()に以下の処理を記述せよ
  - メンバ変数payments[1000]の各要素を-1で初期化する
    (for文を使うと良い)

演習3：コンストラクタ

- コンストラクタKakeibo()に以下の処理を記述せよ
  - メンバ変数payments[1000]の各要素を-1で初期化する
    (for文を使うと良い)

    ※なお、paymentsは支払金額を記録する配列

演習3：pay()関数の実装

- メンバ関数pay (int amount)に以下の処理を記述せよ

```
void pay(int amount) {
  int i = 0;
  while( payments[i] != -1) {
    i++;
  }
  payments[i] = amount;
  printf("%d円の支出を記録しました。\n", amount);
}
```

※この処理では配列の先頭から-1を探索し、最初に見つかった-1の要素に支出金額amountを代入する

演習4：list_payments()の実装

- メンバ関数list_payments()に以下の処理を記述せよ
  - メンバ変数payments[1000]に記録された支出金額を以下のように表示する

100円の支出を記録しました。
200円の支出を記録しました。
500円の支出を記録しました。

演習5：average()の実装

- メンバ関数average()に以下の処理を記述せよ
  - メンバ変数payments[1000]に記録された支出金額の平均値を計算し以下のように表示する

支出平均：266円

演習6：average()の実装

- メンバ関数sum()に以下の処理を記述せよ
  - メンバ変数payments[1000]に記録された支出金額の合計値を計算し以下のように表示する

支出合計：800円

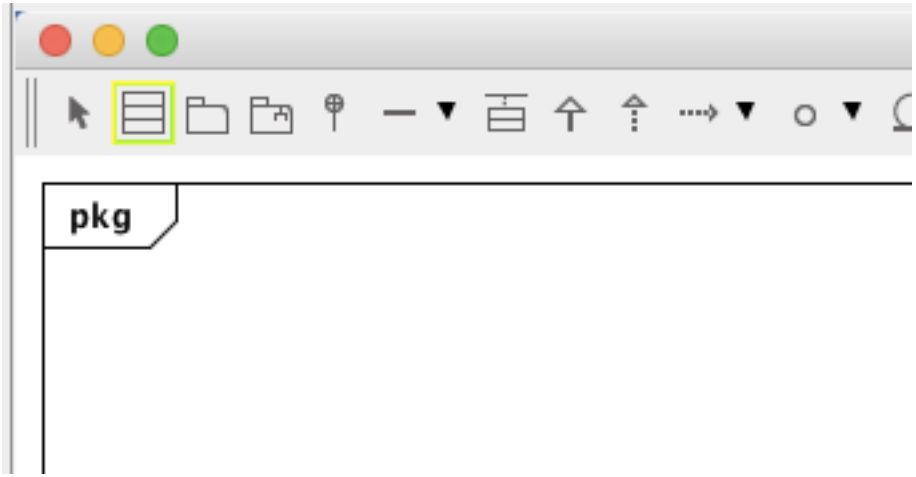
演習7：Astahによるクラス図作成

- 端末で以下を入力
  $ astah

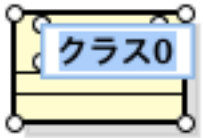
- [図]→[クラス図]を選択

演習7：Astahによるクラス図作成

- 左上のクラス図マークをクリック



- pkgの中でクリックすると、クラス図のひな形が出来る。



演習7：Astahによるクラス図作成

- クラス名をKakeiboに設定
- 右クリックし「属性の追加」を選択するとメンバ変数、「操作の追加」を選択するとメンバ関数が設定できる

- Kakeiboクラスのクラス図を作成し提出せよ

演習8(optional)：ユースケース図

- Astahを再起動し、ユースケース図を選択
- ユースケース図を作成

演習8(optional)：継承を用いた機能拡張

- Kakeiboクラスを継承したExtendKakeiboクラスを作成し、収入金額も入力できるようにしなさい
- 現在の残高を表示する以下の関数を作成しなさい
  void remaining_amount() {}

次回

- 試験、期待しています。
- 計算モデル(オートマトン、正規表現、状態遷移図)に進みます。