設計の計測とプログラミング

FU14 ソフトウェア工学概論

第11回

吉岡 廉太郎

前回の内容

- デザインパターン：設計の定石
  - 再利用可能な部分的解法
  - 設計の原則に則った「設計のパターン集」
- その他の設計要素
  - データ管理の設計、例外処理の設計、UI設計
- 設計書
  - プログラミングに必要な情報を記述する

今日の内容

- 設計の計測
  - ドキュメントを使って計測
  - 規模と品質の測定→管理と予測
- プログラミング工程
  - コーディング規約
  - プログラミングの指針
  - ドキュメント

設計の計測

- 計測の意義は、理解を深め、予測に役立てること

- 目的
  - モジュールの性質を計測し、設計の理解を深める
  - 開発計画（期間、リソース）の検討
  - テスト工程の見積もり

- 評価対象
  - 規模
  - 品質

オブジェクト指向における規模

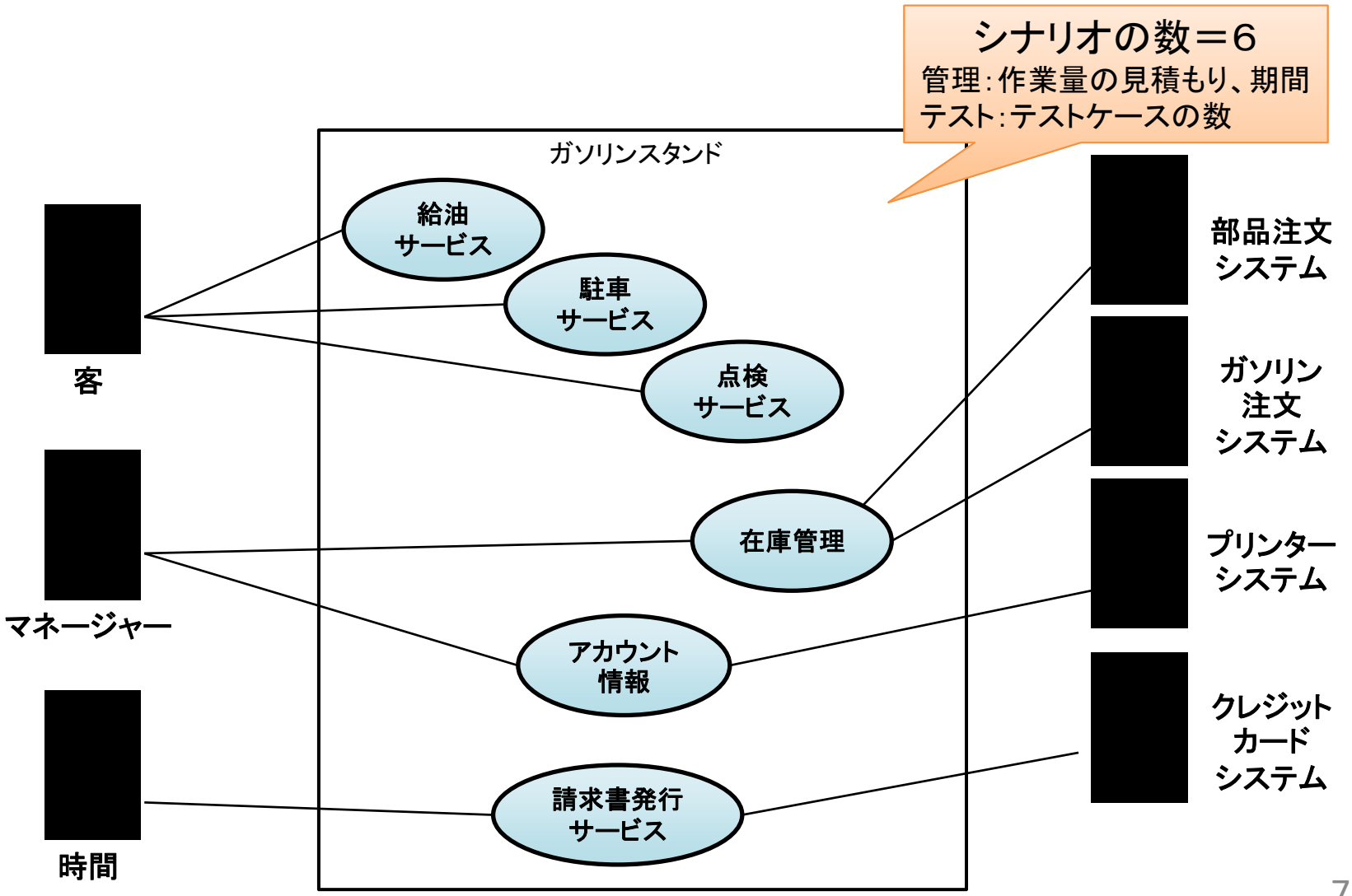
- オブジェクトとメソッドで規模を予測する
- 規模を表す9要素
  - ①シナリオの数
  - ②主要クラスの数
  - ③補助クラスの数
  - ④主要クラスごとの補助クラスの数
  - ⑤サブシステムの数
  - ⑥クラスの大きさ（継承元も含めた、属性と操作の合計数）
  - ⑦サブクラスが上書きする操作の数（NOO）
  - ⑧サブクラスが追加する操作の数
  - ⑨Specialization Index（SI）
    - SI = (NOO × 深さ）／（クラスのメソッド数）

Lorenz and Kidd

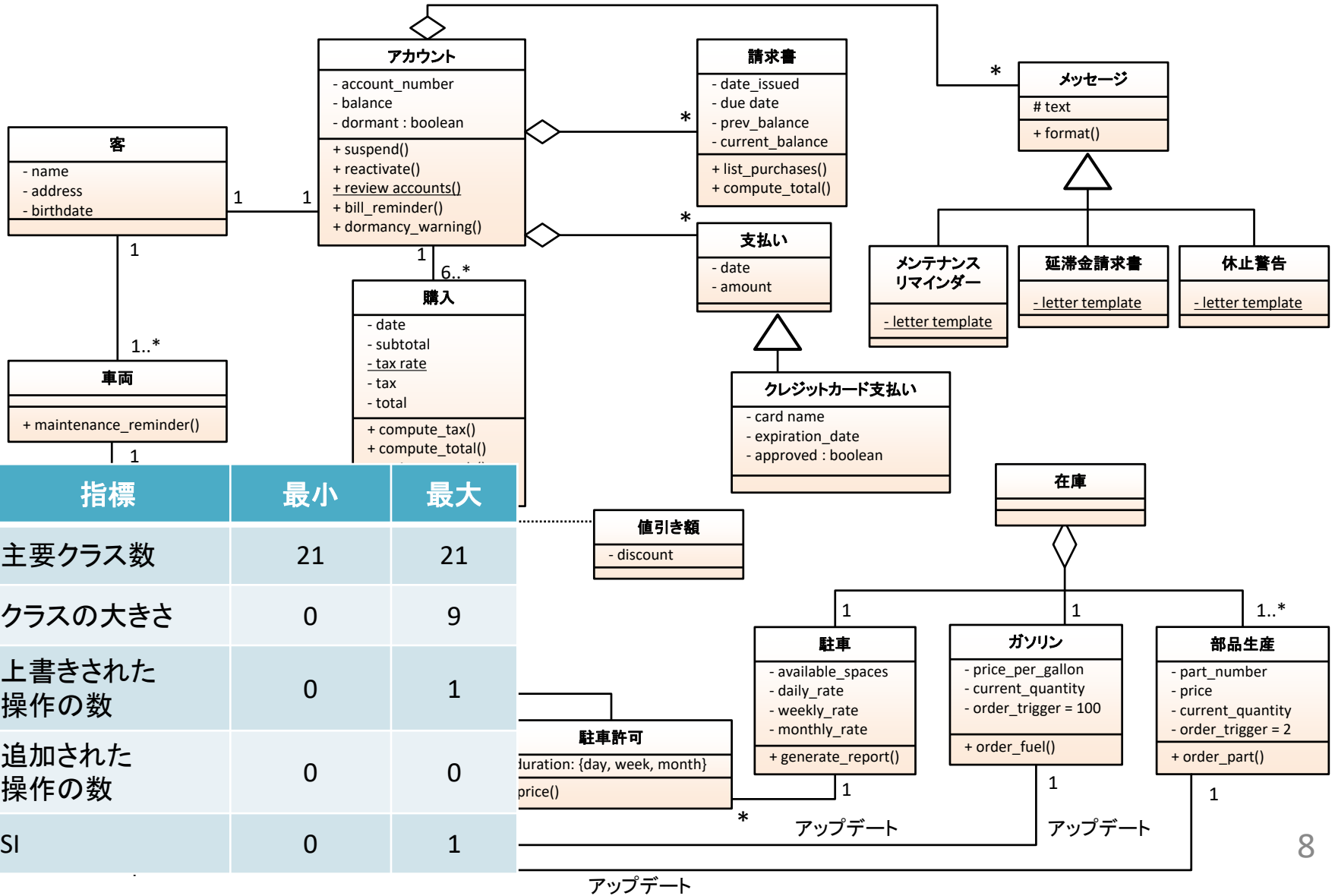
工程と測定メトリックスの収集

メトリックス	要求記述	アーキテクチャ設計	モジュール設計	コーディング	テスト
シナリオの数	X				
主要クラスの数	X	X			
補助クラスの数			X		
主要クラスごとの補助クラスの数			X		
サブシステムの数			X	X	
クラスの大きさ		X	X	X	
サブクラスが上書きする操作の数		X	X	X	X
サブクラスが追加する操作の数		X	X	X	
Specialization Index		X	X	X	X

ガソリンスタンドのユースケース



ガソリンスタンドのクラス階層



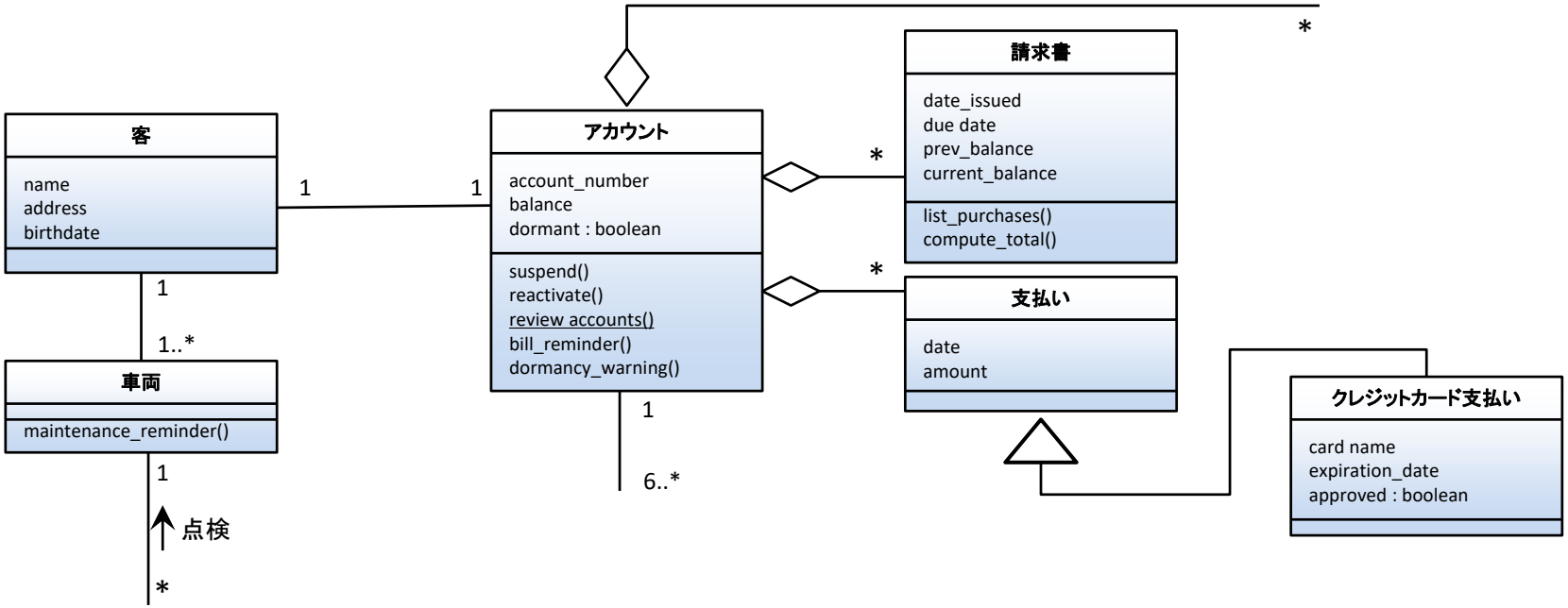
指標	最小	最大
主要クラス数	21	21
クラスの大きさ	0	9
上書きされた操作の数	0	1
追加された操作の数	0	0
SI	0	1

品質の計測

Chidamber,Kemerer

- オブジェクト指向設計を対象とした品質の指標
  - クラス毎のメソッドの重み= $\sum_{i=1}^{n} c_i$ { n:メソッドの数 c:各メソッドの複雑度
  - 継承の深さ
  - 子クラスの数
  - オブジェクト間の結合性
  - レスポンスの数
  - メソッドの強度

ガソリンスタンド：品質



指標	請求書	支払い	クレジットカード支払い	アカウント	客	車両
クラス毎のメソッドの重み	2	0	0	5	0	1
子クラスの数	0	1	0	0	0	0
継承の深さ	0	0	1	0	0	0
オブジェクト間の結合性	1	2	1	5	2	2

強度の計算

強度の弱さを数値化する

- クラスCはM_1からM_nまでのn個のメソッドを持つ
- メソッドM_jで使用されるインスタンス変数のセットを$\{ I_j \}$とする
- I_rとI_sが共通のメンバーを共有しない場合、ペアのコレクション(I_r, I_s)をPとする
  - $P = \{(I_r, I_s) \mid I_r \cap I_s = \emptyset\}$
- I_rとI_sが少なくとも1つのメンバーを共有している場合、ペアのコレクション(I_r, I_s)をQとする
  - $Q = \{(I_r, I_s) \mid I_r \cap I_s \neq \emptyset\}$
- Cのメソッドの強度は
  - $|P| > |Q|$ なら $|P| - |Q|$
  - それ以外は 0

✓同じ変数を使うメソッドは強度が高い

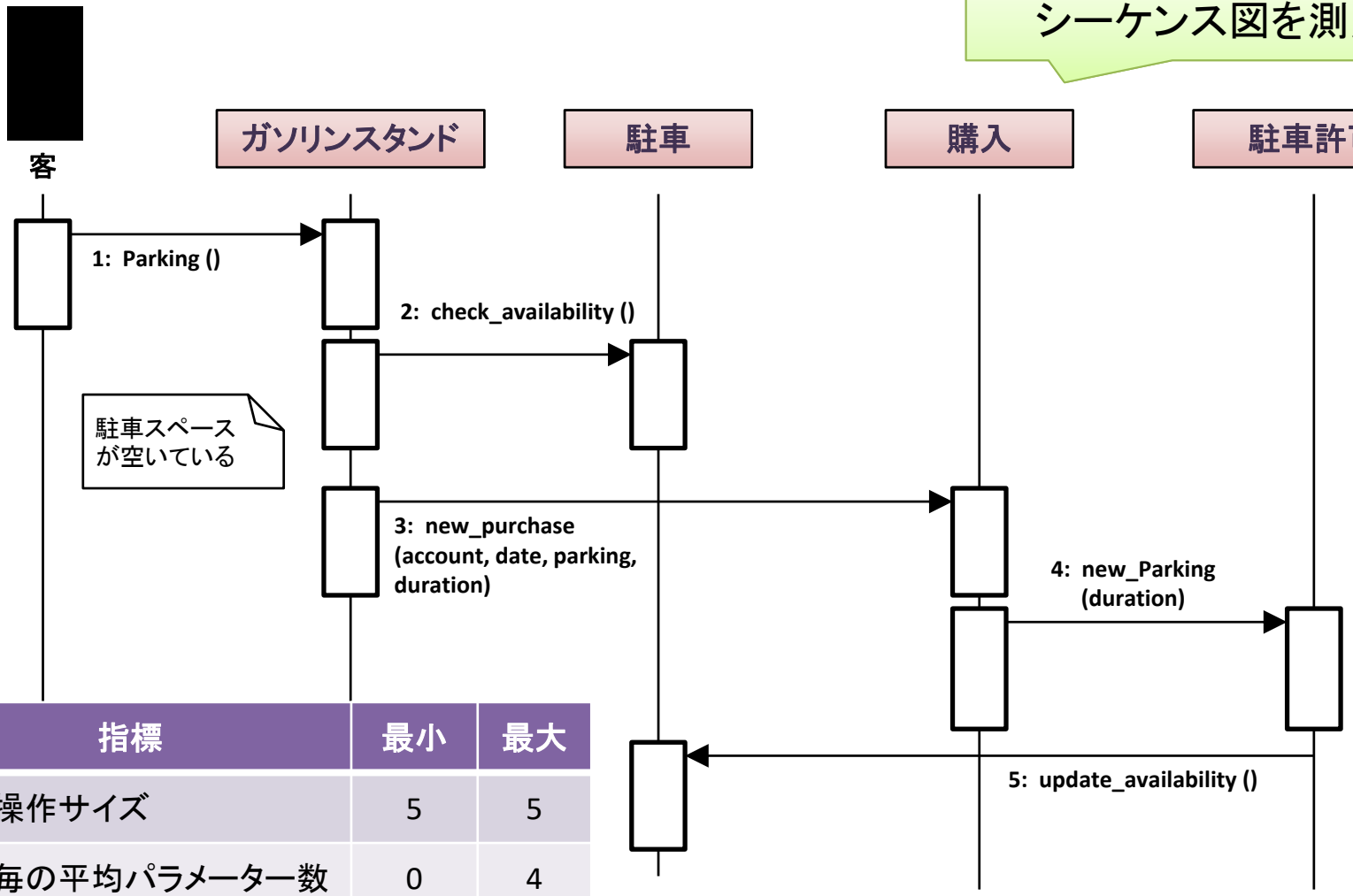
✓共有する変数が多いほど強度が高い

その他の指標

- 運用後のクラスの大きさの変化を予測する
  - メッセージ通信結合
    - クラス内でのメソッド呼び出しの数
  - データ抽象化結合
    - 利用している抽象データ型のうち、他クラスで宣言されているものの数
- 操作の規模を測る
  - 平均操作サイズ
  - 操作毎の平均パラメータ数

操作の規模

シーケンス図を測定



指標	最小	最大
平均操作サイズ	5	5
操作毎の平均パラメーター数	0	4

ドキュメントと測定できる指標の関係

メトリックス	ユースケース	クラス図	インタラクション図	クラス記述	ステートマシン図	パッケージ図
シナリオの数	x					
主要クラスの数		x				
主要クラスごとの補助クラスの数		x				
サブシステムの数						x
クラスの大きさ		x		x		
サブクラスが上書きする操作の数		x				
Specialization Index		x				
オブジェクト間の連結度		x				
レスポンスの数				x		
メソッド間の結合の弱さ				x		
平均操作サイズ			x			
操作毎の平均パラメーター数			x			
操作の複雑さ				x		
publicとprotectedの割合				x		

注意：完全なリストではありません

→必要な指標に応じてドキュメントを作成する

プログラミングの工程

- 設計を実装する
  - 顧客、ユーザの要求は設計として表現されている
- 課題
  - 設計では表面化しない細かい問題が存在する
  - 見直しやすい理解しやすいプログラムを記述する
  - 設計やプログラミング言語の特徴を引き出しつつも、再利用しやすいコードを記述する

コーディング規約

- プログラムやプログラミング作業の標準化
  - コードのフォーマットの定義
  - 変数や関数の命名規則
  - コメントなど、説明を記述する形式や位置についての規則
  - 考えを整理し、設計に則ったコードの作成手順

- 目的と効果
  - 結合性の低減、強度の向上、良く定義されたインタフェースを実現する
  - 統合テスト、メンテナンス、将来の拡張を容易にする
  - 全体像を分かりやすくする
  - 自動化ツールで一括処理できるようにする

規約で定める内容の例

- 例）Google Code Style
  - ソースファイルのファイル形式
  - ソースファイルの構造
  - コードのフォーマット
  - 命名規則
  - プログラミング指針

https://google.github.io/styleguide/javaguide.html

- 特徴
  - 具体（機械）的に実施できる内容に限定
  - 数々のツール向けの設定ファイルを配布し適用を後押し

4.8.4 Switch statementsからの引用

```
switch (input) {
  case 1:
  case 2:
    prepareOneOrTwo();
    // fall through
  case 3:
    handleOneTwoOrThree();
    break;
  default:
    handleLargeNumber(input);
}
```

4.8.4.1 Indentation

...

After a switch label, there is a line break, and the indentation level is increased +2, ...

4.8.4.2 Fall-through: commented

Within a switch block, each statement group either terminates abruptly ..., or is marked with a comment to indicate that execution will or might continue into the next statement group. ... (typically // fall through) ...

プログラミングの指針

- 設計の限界
  - プログラミングには創造性が不可欠
  - プログラミング言語特有の事情を考慮する
  - 一つの機能の実現方法は無数にある
  - 設計には「遊び」が存在する

- 高品質のコードを作成するためのガイドライン
  - 言語に依存しない作法

①制御構造

指針

- 制御が読み取りやすいように書く
- ブロックに分けて組み上げる
- 細か過ぎず、大雑把過ぎない程度に調整する
- 引数やコメントを活用して、連結をわかりやすくする
- コンポーネント間の依存性が見えるようにする

制御構造の例

- 悪：制御が命令の間を飛び回る

```
        benefit = minimum;
        if (age < 75) goto A;
        benefit = maximum;
        goto C;
        if (age < 65) goto B;
        if (age < 55) goto C;
A:          if (age < 65) goto B;
        benefit = benefit * 1.5 + bonus;
        goto C;
B:          if (age < 55) goto C;
        benefit = benefit * 1.5;
C:          next statement
```

給付金の額を計算するプログラム

- 良：整理後…

```
if (age < 55) benefit = minimum;
elseif (age < 65) benefit = minimum + bonus;
elseif (age < 75) benefit = minimum * 1.5 + bonus;
elseif benefit = maximum;
```

②アルゴリズム

指針

- 効率(処理速度)を高める
- 処理速度以外のコスト効率を高める

- 効率の追求に伴うコストも削減する
  - より速いコードを書くのにかかるコスト
  - そのコードのテストにかかるコスト
  - そのコードを理解するのに必要なコスト
  - そのコードを変更するのに必要なコスト

③データ構造

指針

- データ構造でプログラムを整理する
- データ構造とプログラム構造を一致させる
  - 適切なプログラミング言語の選択

- 効果
  - 単純化
  - 読みやすく理解しやすくなる

例：所得税の計算

1. 最初の¥10,000の収入にかかる税率は10%
2. ¥10,000から次の¥10,000の収入にかかる税率は12%
3. ¥20,000から次の¥10,000の収入にかかる税率は15%
4. ¥30,000から次の¥10,000の収入にかかる税率は18%
5. ¥40,000より多くの収入にかかる税率は20%

```
tax = 0.
if (taxable_income == 0) goto EXIT;
if (taxable_income > 10000) tax = tax + 1000;
else{
    tax = tax + .10 * taxable_income;
    goto EXIT;
}
if (taxable_income > 20000) tax = tax + 1200;
else{
    tax = tax + .12 * (taxable_income - 10000);
    goto EXIT;
}
if (taxable_income > 30000) tax = tax + 1500;
else{
    tax = tax + .15 * (taxable_income - 20000);
    goto EXIT;
}
if (taxable_income > 40000) {
    tax = tax + .18 * (taxable_income - 30000);
    goto EXIT;
}
else
    tax = tax + 1800. + .20 * (taxable_income - 40000);
EXIT:  ;
```

例：所得税の計算

- 税率のテーブルを用意する

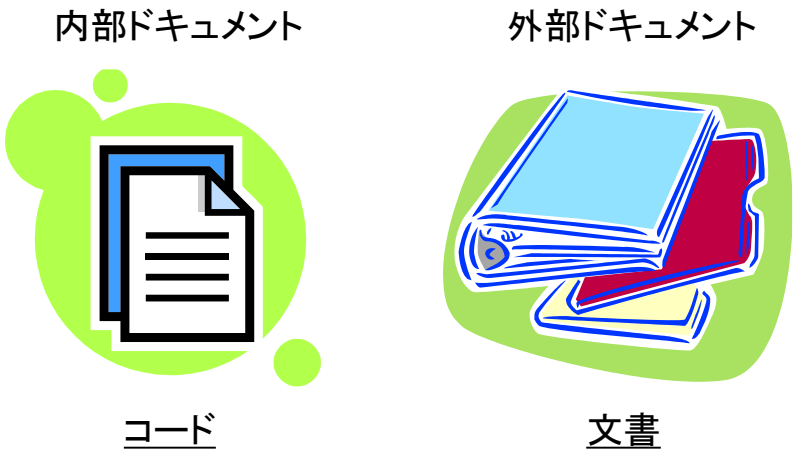
収入範囲(bracket)	基本額 (base)	税率 (percent)
0	0	10
10,000	1000	12
20,000	2200	15
30,000	3700	18
40,000	5500	20

- 単純化されたコード

```
for (int i=2 ; level=1 ; i<=5 ; i++)
    if (taxable_income > bracket[i])
        level = level + 1;
tax = base[level] + percent[level] * (taxable_income - bracket[level]);
```

プログラミング工程のドキュメント

- ドキュメントの種類
  - 内部ドキュメント
  - 外部ドキュメント



- 内部ドキュメント
  - ヘッダーコメントブロック
  - 変数名や関数名で表現されたその意味や役割
  - その他プログラム中のコメント
  - わかりやすいフォーマット
  - データに対するドキュメント

良いコメントと悪いコメント

- ほとんど意味の無いコメント

```
// Increment  i3
i3 = i3 + 1;
```

- 意味を伝えるコメント

```
// Set counter to read next case
i3 = i3 + 1;
```

- 命名規則を定めて、よりわかりやすく

```
case_counter = case_counter +1;
```

```
weekwage = (hrrate * hours) + (. 5) * (hrrate) * (hours - 40 .) ;
```

外部ドキュメント

- コンポーネントが解決する課題の説明
  - コンポーネントがいつ呼び出され、どのような役割を担っているか
- 課題を解決するためのアルゴリズムの説明
  - 数式、境界条件、その他特別な条件
  - アルゴリズムの出典
- コンポ。レベルでのデータの流れを説明

本日のまとめ

- 設計の計測
  - ドキュメントを用いた計測で、規模と品質を測る
- コーディング規約
  - 開発者の作業品質を向上
  - 他工程や将来の拡張を見据えた効率化
- プログラミングの指針
  - わかりやすく、品質の高いコードを開発するためのガイドライン
- プログラムのドキュメント
  - コードを理解しやすくする内部ドキュメント
  - プログラムのことを記述する外部ドキュメント

次回講義の事前学習：
6.1.4, 6.2.2, 6.2.4, 6.2.5, 7.2.5, 7.2.6, 7.4.2