

### 第3章 クラス図とオブジェクト図

オブジェクト指向分析の前半では、各機能の実現に必要なオブジェクトを発見し、システムのもつ静的な構造をクラス図で定義する。

#### 3-1 オブジェクトの抽出

オブジェクトを抽出する方法として、名詞・名詞句の抽出がある。  
オブジェクトは、名詞・名詞句としてシナリオに登場する。シナリオごとに、登場する名詞・名詞句をオブジェクトの候補として抽出する。

オブジェクトは概ね、次のような種類に分類する。

- アクター
- オブジェクト
- オブジェクトの属性
- 処理機能
- 開発対象以外のもの

表 シナリオから名詞・名詞句を抽出

ユースケース名	車を予約する
シナリオ1: 予約が成功する場合	
1. 予約受付係は予約処理を選択する	
2. 予約受付係は希望日「20XX/11/20」を入力する	
3. システムは希望日「20XX/11/20」に空いている車の一覧を表示する	
4. 予約受付係は一覧から「6号車:ワゴン 4WD」を選択する	
5. システムは予約者を問い合わせる	
6. 予約受付係は予約者の情報を入力する。	
氏名「及川 源太」 電話番号「019-622-1500」 郵便番号「020-0021」	
住所「岩手県盛岡市中央通3丁目2-17」	
7. 予約受付係は予約を実行する	
8. システムは予約を受け付ける	
9. システムは予約番号「2001」を表示する	
10. 予約受付係は終了を実行する	

表 オブジェクト候補の選別

抽出した名詞、名詞句	選別結果
予約受付係	アクター
予約処理	提供する機能
希望日「20XX/11/20」	予約オブジェクトの属性
システム	開発対象のオブジェクト
空いている車の一覧	オブジェクト
「6号車:ワゴン 4WD」	オブジェクト
予約者	オブジェクト
予約者の情報	予約者オブジェクトと同じ意味
氏名「及川 源太」	予約者オブジェクトの属性
電話番号「019-622-1500」	予約者オブジェクトの属性
郵便番号「020-0021」	予約者オブジェクトの属性
住所「岩手県盛岡市中央通3丁目2-17」	予約者オブジェクトの属性
予約(項番 7)	処理
予約(項番 8)	オブジェクト
予約番号「2001」	予約者オブジェクトの属性、または、予約オブジェクトの属性
終了	処理

#### 【参考】ロバストネス分析

ユースケース記述やシナリオのように文章で記述された要求定義から分析レベル※1のオブジェクトをみつけ、適切な単位にまとめる分析手法である。

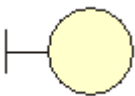


※1分析レベル・・・ここでは具体的なアーキテクチャが考案される前の分析レベルとなる

#### 【ロバネストレス分析によって得られる効果】

- ①ユースケースの妥当性を検証することができる
- ②ユースケースが洗練される
- ③概念モデル(エンティティモデル)が洗練される
- ④設計モデルのインプットを作成できる

ロバストネス分析ではロバストネス図という図を記述する。ロバストネス図とは、ユースケースに登場するクラスを定義するための図であり、このクラスの静的な側面をクラス図で、動的な側面を相互作用図(シーケンス図かコラボレーション図)で記述する。

ロバネストレス分析では、システムに登場するオブジェクトを3種類に分類することで、オブジェクト間の関連や制御フローをコンパクトに表現できる。

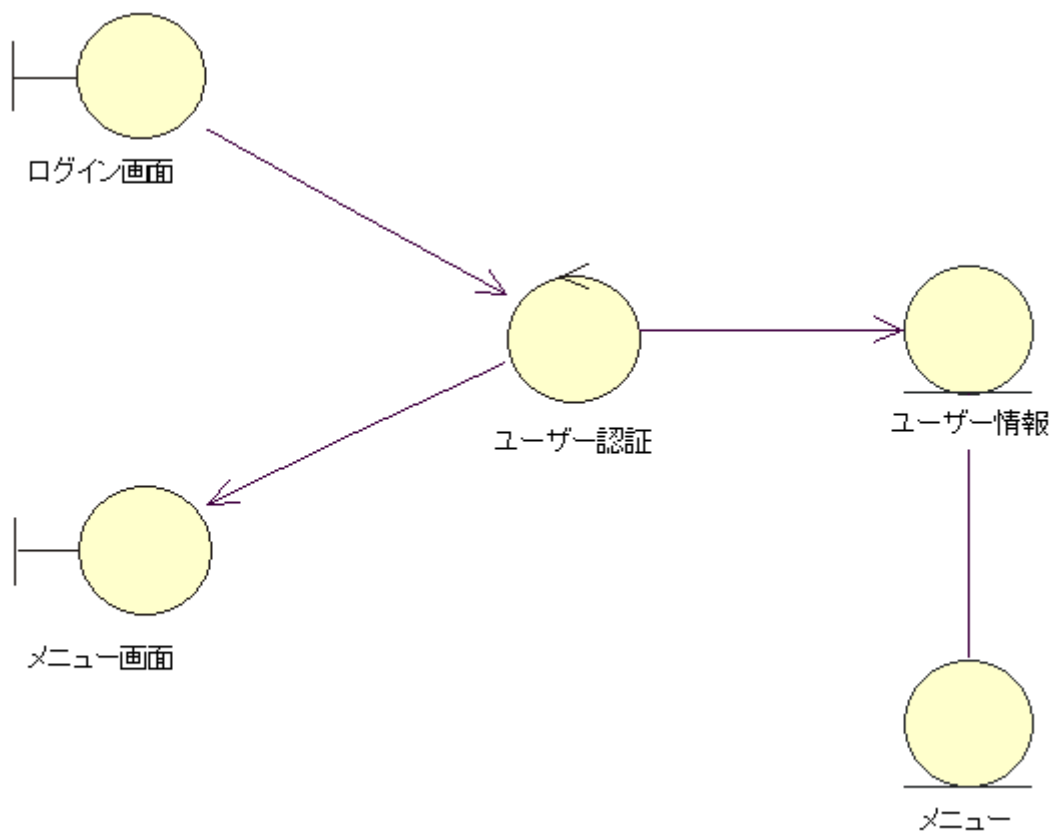
バウンダリ	データを保持するオブジェクト	
エンティティ	画面や外部インターフェース部分のオブジェクト	
コントロール	制御を行うオブジェクト	

例題 次のユースケースシナリオを分析し、ロバネストレス図しなさい。

【ログインユースケース】

- 1. 営業員はユーザー ID とパスワードを入力する
- 2. システムはユーザー ID とパスワードが正しいことを確認し、メニュー画面にユーザーが選択可能なメニューを表示する

【ロバネストレス図】



### 3-2 クラス図の作成

同じ性質を持つオブジェクトをグループ化し、そのグループごとにクラスを定義する。定義したクラスからクラス図を作成する。

表 クラスの定義

オブジェクト	クラス
希望日「20XX/11/20」の空いている車の一覧	空車リスト
当日の空いている車の一覧	
6号車:ワゴン 4WD	車
予約済みの車「7号車:ハッチバック 2WD」	
予約者「及川 源太」	顧客
借用者「大谷 翔平」	
予約番号「2001」の予約	予約
貸出	貸出

クラス図とは、システムの静的な構を表すためのダイアログラムであり、クラスやインタフェース、およびそれらの間の関係 (relationship) から構成される。

ユースケース記述などを参考にして抽出したオブジェクトを、現実存在する「もの」(オブジェクト)の構造(属性)、振る舞い(操作)の共通性に着目して抽象化する。

#### ▼クラスの表記

クラスは3つの区画に分割された長方形で表記する。

一番上の区画には、クラス名、中間の区画には属性名、一番下の区画には操作名を表記する。

属性および操作は1行に1つ配置する。

オプションとしてステレオタイプをクラス名の上に置くことができる。



#### ▼クラスの表記の種類

クラスは属性および相殺の表示領域どちらか、または、両方表示しなくても構わない。ただし、表示しない場合も、属性、操作がないとは限らない。

分析の初期段階では、属性や操作が明確に定義できないことがある。

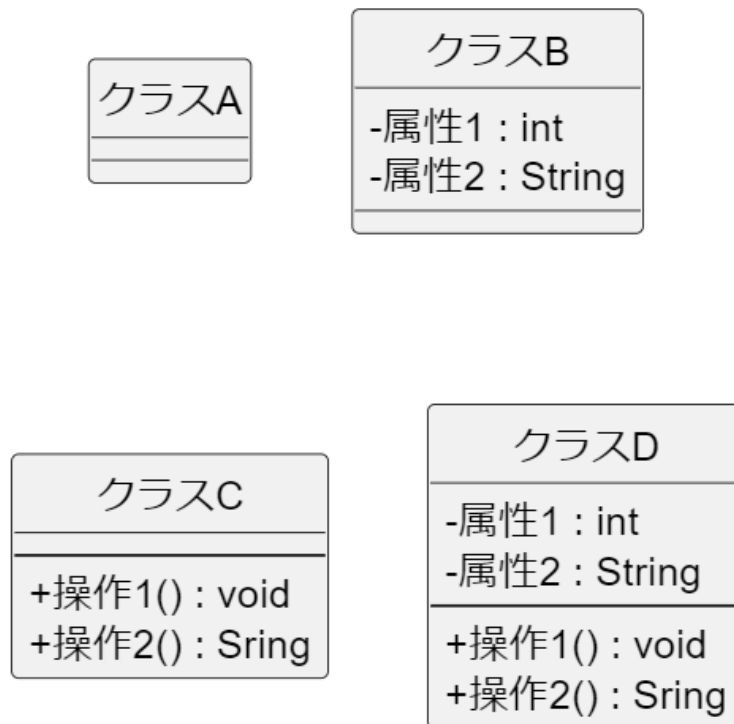


図 クラス表記の種類

```
@startuml
    skinparam classAttributeIconSize 0
    class A <<stereotype>> {}
    class B {
        - 属性1:int
        - 属性2:String
    }
    class C {
        ---
        + 操作1():void
        + 操作2():String
    }
    class D {
        - 属性1:int
        - 属性2:String
        ---
        + 操作1():void
        + 操作2():String
    }
    hide A circle
    hide B circle
    hide C circle
    hide D circle
```

```
@enduml
```

## 【属性】

可視性 属性名 : 型表現 = 初期値

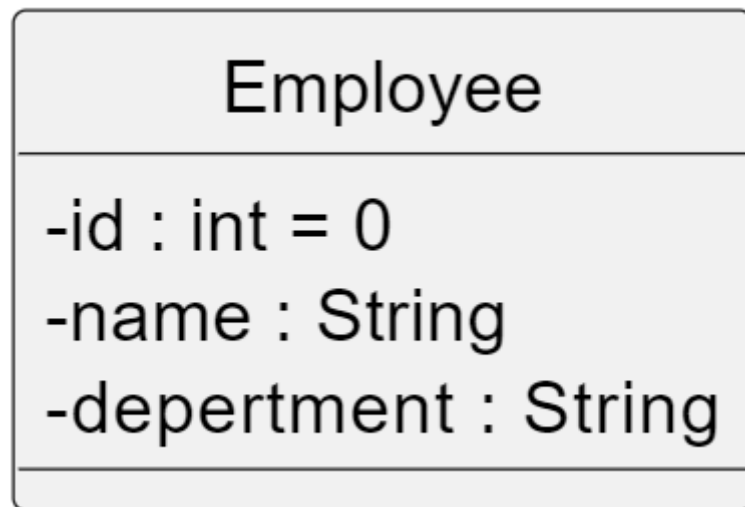


図 属性の記述方法

## 【操作】

可視性 操作名(パラメータリスト):戻り値の型

※パラメータリストが複数ある場合は、「パラメータ名:型」をカンマ区切りで並べる

例) + setAddress(pref:String, city:String):void

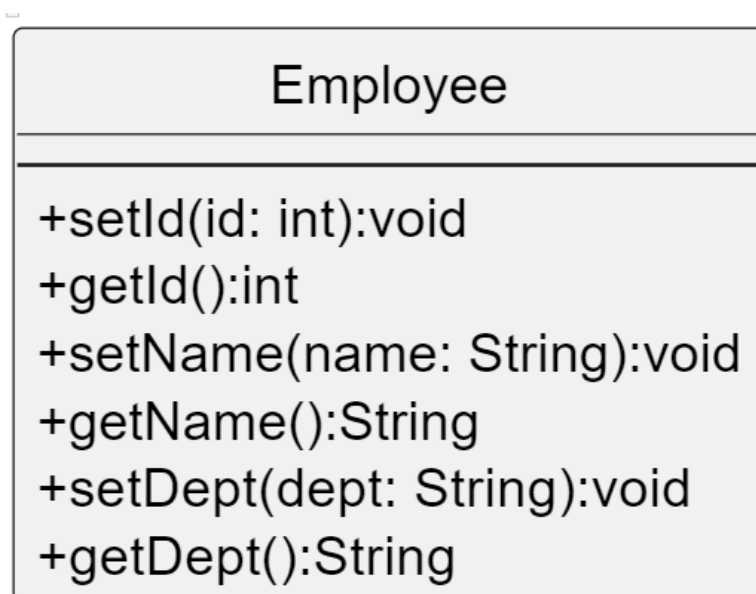










図 操作の記述方法

## ▼可視性

可視性は、属性や操作にアクセスできる権限を表現します。

表記	フィールドのアイコン	メソッドのアイコン	キーワード	意味
+			public	すべてのクラスから参照可能
—			private	当該クラス内のみ参照可能
#			protected	当該クラスとそのサブクラスからのみ参照可能
~			package private	同一パッケージ内の任意のクラスから参照可能
/			派生属性	他の属性から計算できる属性

文字フィールドのアイコンメソッドのアイコン可視性

Employee
-id : int = 0 -name : String #deperment : String / 勤続年数 : int
+setId(id: int):void +getId():int +setName(name: String):void +getName():String +setDept(dept: String):void +getDept():String

Employee
□ id : int = 0 □ name : String ◇ deperment : String / 勤続年数 : int
● setId(id: int):void ● getId():int ● setName(name: String):void ● getName():String ● setDept(dept: String):void ● getDept():String

図 可視性

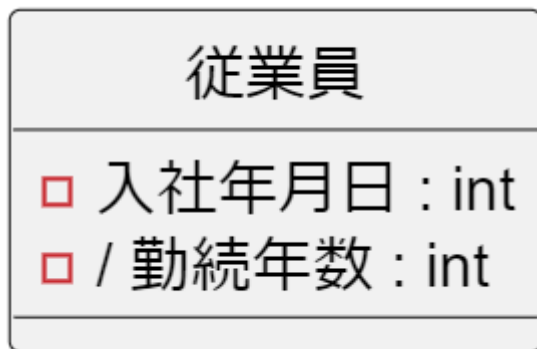
```
@startuml
skinparam classAttributeIconSize 0
class Employee{
- id:int=0
- name:String
#deperment:String
/連続年数:int
```



```

---
+setid(id:int):void
+getid():int
+setname(name:String):void
+getName(name:String):void
+setdept(dept:String):void
+getdept():String
}
hide Employee circle
@enduml

```



例)

「従業員」クラスの「勤続年数」は「入社年月日」と現在の日付から計算が可能なので、派生属性となります。

図 派生属性

#### ▼ 関連(association)

関連は、クラス間に「has-a(所有する)」の関係があることを表現する。

関連は2つのクラス間の実線として記述し、両端に矢印をつけることができる。

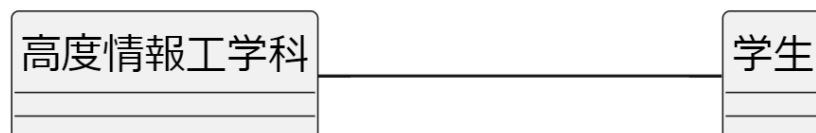


図 関連

```

@startuml
    left to right direction
    高度情報工学科 -- 学生
@enduml

```

#### ▼ 関連の方向(誘導可能性)

関連の端に矢印をつけた場合は、片方のクラスのみがもう一方のクラスを認識していることを示す。

矢印のない場合は、双方向の関連もしくは関連の方向が未決定であることを示す。

矢印をつけて関連の方向性を明確にすることを「誘導可能性」という。

※下図のメッセージ送信とは、Java言語仕様のメソッド呼び出しに相当する。

#### ●双方向の関連

関連している2つのオブジェクトが、お互いにメッセージ送信をしてもかまわないことを示す。



図 双方向の関連

#### ●片方向の関連

関連している2つのオブジェクトが、片方向のメッセージを行うが、逆方向のメッセージ送信は行わないことを示す。

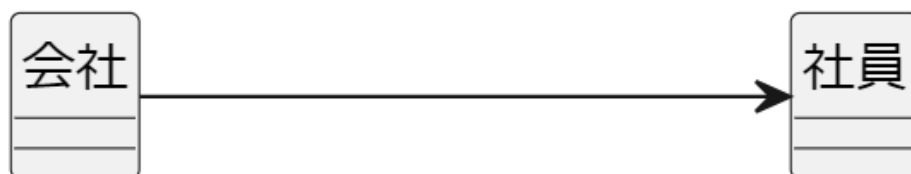


図 片方向の関連

```
@startuml
    left to right direction
    会社 --> 社員
@enduml
```

#### ▼ 関連名

関連名は、関連の意味を明確にするために関連(集約、複合)の実線の中央に記述される名前である。  
関連名に方向性がある場合には、関連名の左右に黒塗りの三角を表示し、関連の主体と客体を表す。

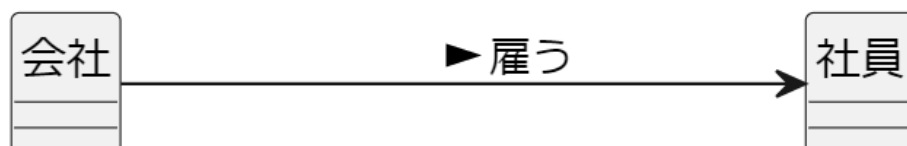


図 関連名

```
@startuml
    left to right direction
    会社 --> 社員:雇う >
@enduml
```

## ▼ ロール(役割)名

ロール名は、関連先のクラスの役割を明確にするために関連(集約、複合)の端(関連端)に表示する名前である。

設計フェーズでは、通常、関連先のオブジェクトを保持するための属性名となる。

ロール名はその役割を持つクラス側の端に記述する。

※なお、PlantUMLではロール名の実装がされてない。



図 ロール(役割)名

## ▼ 多重度

多重度とは、関連するクラス間の数的な対応関係を表す。

関連元クラスの1つのオブジェクトに対応する関連先クラスのオブジェクトの数を、関連先の関連端に表示する。



図 多重度

表記	接続される可能性のある要素の数
0..1	0 か 1
1	1 のみ
0..* または *	0以上
1..*	1以上
3..10	3 ～ 10
1..5,10..20	1～5 と 10 ～ 20

表 多重度

```
@startuml
    left to right direction
    会社 "1"--"*" 社員
@enduml
```

### ▼ 集約 (aggregation)

集約は関連の一種であり、クラス間に「**has-a**(持っている)/**part-of** (一部である)」の関係があることを表現する。集約は「全体 — 部分」の構造や「所有する側 — 所有される側」の関係を明確に表現する場合に使用される。集約は、全体側のクラスに白抜きひし形をつけた実線で記述される。

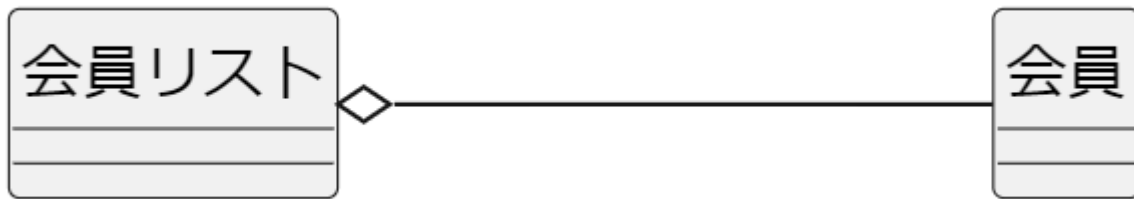


図 集約

```
@startuml
    left to right direction
    会員リスト o-- 会員
    note right of 会員リスト
        会員クラスがなくて成り立つ
    end note
@enduml
```

### ▼ 複合 (composition)

複合は、集約と同様に関連の一種であり、クラス間に「**is-a-part-of** (一部である)」の関係があることを表現する。複合は集約と比較してより強い所有関係を表現しており、「全体 — 部分」あるいは「所有する側 — 所有される側」のオブジェクトの生存期間が一致する。

全体側のオブジェクトが生成されると同時に部分側のオブジェクトが生成され、全体側のオブジェクトが消滅すると部分側のオブジェクトも消滅する場合に使用される。

複合は、全体側のクラスに黒塗りのひし形をつけた実線で記述される。

※複合における全体が和のオブジェクトの多重度は、その性質上必ず「1」になる



図 複合

```
@startuml
    left to right direction
    パソコン "1" *-- ハードディスク
    note right of パソコン
        ハードディスクがないと成り立ちません。
    end note
@enduml
```

### ▼ 依存関係

依存関係は、一時的に相手オブジェクトを使用する関係を表現する。関連に比べて、2つのオブジェクトが関係する期間が短い関係である。

依存は、依存する側から依存される側に向かって破線矢印を記述する。

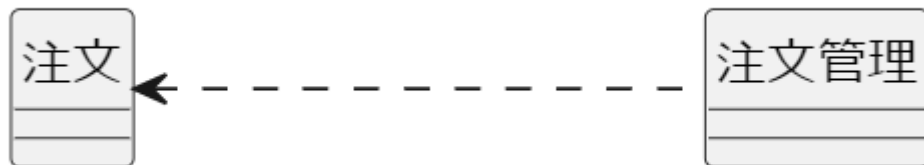


図 依存関係

```
@startuml
    left to right direction
    注文 <-- 注文管理
    note top of 注文管理
        注文管理が注文のメソッドもしくは属性を呼び出している
    end note
@enduml
```

### 【Point】 依存関係を用いるケース

- 分析の段階で、クラス図が関連だらけになってしまい分かりにくくなりそうな場合に、一時的な関係と判断したものを依存関係で表現する。
- 実装を考慮してクラス図を洗練していく段階で、依存するオブジェクトが引数や戻り値として扱われるような一時的な関係であると判断した場合に関連を依存関係に変更する。

### ▼ 継承

継承(特化)は、既存クラスの操作を引き継ぐことにより、新しいクラスを効率よく定義する仕組みである。

新しいクラス(サブクラス)は、既存クラス(スーパークラス)にはなかった新しい操作の定義(差分コーディング)に集中できる。

継承は、継承元のクラスに白抜きの三角形をつけた実線で記述される。

差分をコーディングする

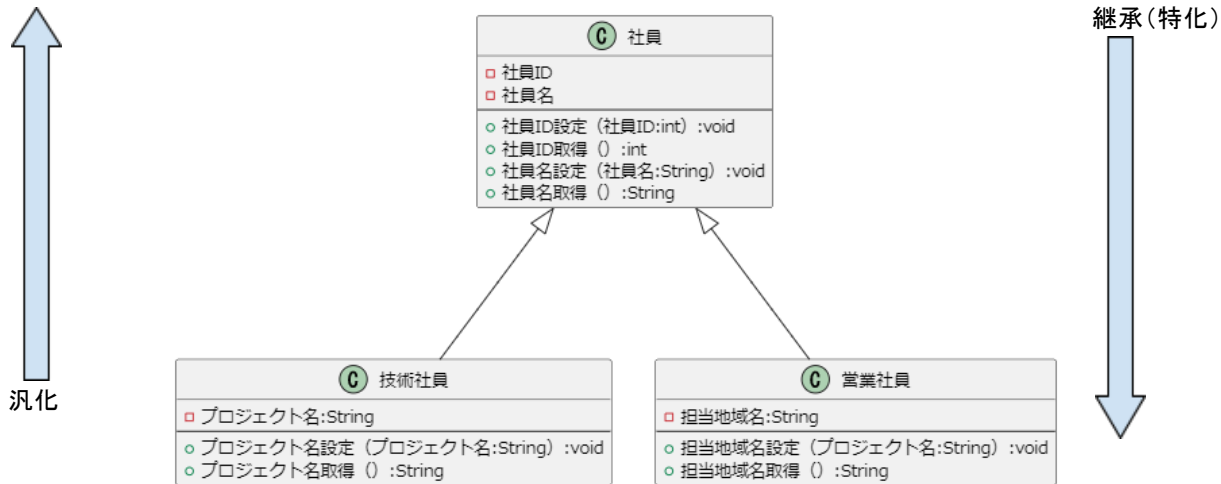


図 継承(特化)・汎化

```
@startuml
class 社員 {
- 社員ID
- 社員名
---
+ 社員ID設定(社員ID:int):void
+ 社員ID取得():int
+ 社員名設定(社員名:String):void
+ 社員名取得():String
}

class 技術社員 {
- プロジェクト名:String
---
+ プロジェクト名設定(プロジェクト名:String):void
+ プロジェクト名取得():String
}

class 営業社員 {
- 担当地域名:String
---
+ 担当地域名設定(プロジェクト名:String):void
+ 担当地域名取得():String
}

社員 <|--down- 技術社員
社員 <|--down- 営業社員
@enduml
```

#### ▼ 汎化

開発対象が複数ある場合に、共通点を抜き出し抽象的(汎用的)にスーパークラスを定義する考え方である。

## ▼抽象クラス

抽象クラスは、通常の操作のように処理内容を具体的に実装した操作(具象操作)と、操作の定義(戻り値、操作名、数)だけを決めて実装のない抽象的な操作(抽象メソッド)を定義することができる。

サブクラスの共通している操作は具体的に実装し、処理内容がサブクラスによって異なる操作は操作の定義(戻り値、操作名、引数)のみを決めておきたいという場合に抽象クラスを利用する。

具象操作の具体的な内容は、サブクラスで再定義(オーバーライド)する。

クラス図では、抽象クラス、抽象操作(抽象メソッド)を斜体()で表記する。

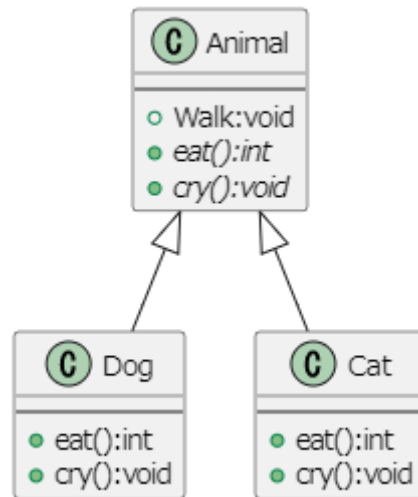


図 抽象クラス

```

@startuml

class Animal {
  ---
  + Walk:void
  + {abstract} eat():int
  + {abstract} cry():void
}

class Dog {
  ---
  + eat():int
  + cry():void
}

class Cat {
  ---
  + eat():int
  + cry():void
}

Animal <|-- Dog
Animal <|-- Cat
@enduml

```

## ▼インターフェース

インターフェースは、実装のない抽象的な操作(抽象メソッド)を定義することができる。

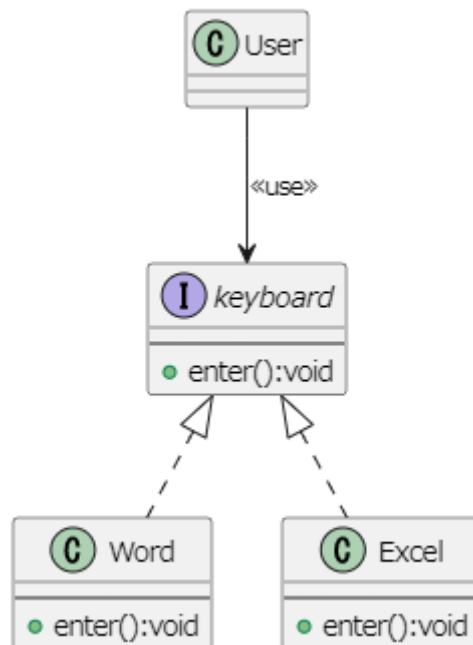
抽象操作の具体的な処理内容は、サブクラスで再定義(オーバーライド)する。

処理内容の違うサブクラスの機能を、1つの操作方法で利用できるようにしたい場合にインターフェースを利用する。

インターフェースを利用してポリモーフィズムを実現することができる。

インターフェースは、実装元のクラスに白抜き三角形をつけた破線で記述される。





```

@startuml

class Word {
  ---
+ enter():void
}

class Excel {
  ---
+ enter():void
}

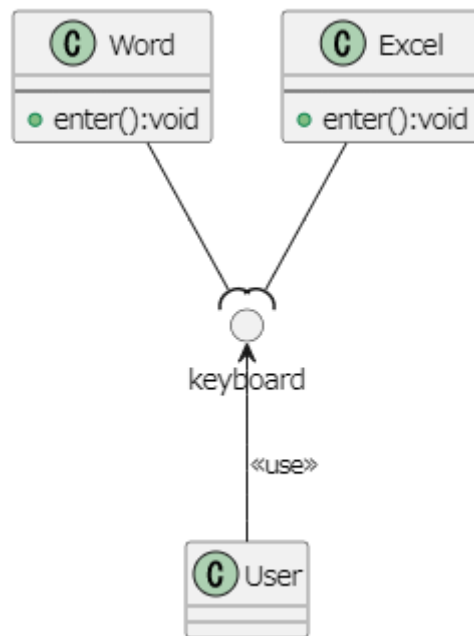
interface keyboard {
  ' <<interface>>
  ---
+ enter():void
}
keyboard <|.. Word
keyboard <|.. Excel

class User {

}
keyboard <-up- User:<< use >>

@enduml
  
```

インターフェースは、下図のように白丸（ロリポップ表記）で記述することもできる。※ロリポップ表記はコンポーネント図で 사용되는。



```

@startuml

class Word {
  ---
  + enter():void
}

class Excel {
  ---
  + enter():void
}

circle "keyboard" as I1
Word --( I1
Excel --(
class User {

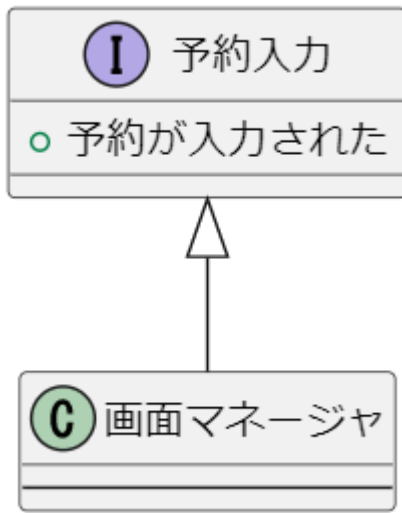
}
I1 <-- User:<< use >>

@enduml

```

【例題3－1】次の条件を満たすようにクラス図を作成しなさい。

- 予約入力インターフェースは「予約入力」というインターフェース名で、「予約が入力された」というメソッドをもつ。
- 画面マネージャーは、予約入力インターフェースを実装している



【提出ファイル】例題3-1\_予約入力インターフェース.pu

### 3-3 オブジェクト図 <https://plantuml.com/ja/object-diagram>

オブジェクト図は、クラスからインスタンスされたオブジェクト間の関係を示したものである。

クラス図に反映するオブジェクトの数的関係を確認したいときに利用する。

オブジェクトは2つの区画に分割された長方形のアイコンで表記する。

上の区画にはオブジェクト名、下の区画には属性名を記述する。

オブジェクト名は、「オブジェクト名: クラス名」の形式で表記する。(通常は下線を引く)

オブジェクト名は

また、属性名には初期値を記述することもできる。

オブジェクト間の関係は、リンク(実線)で表記する。

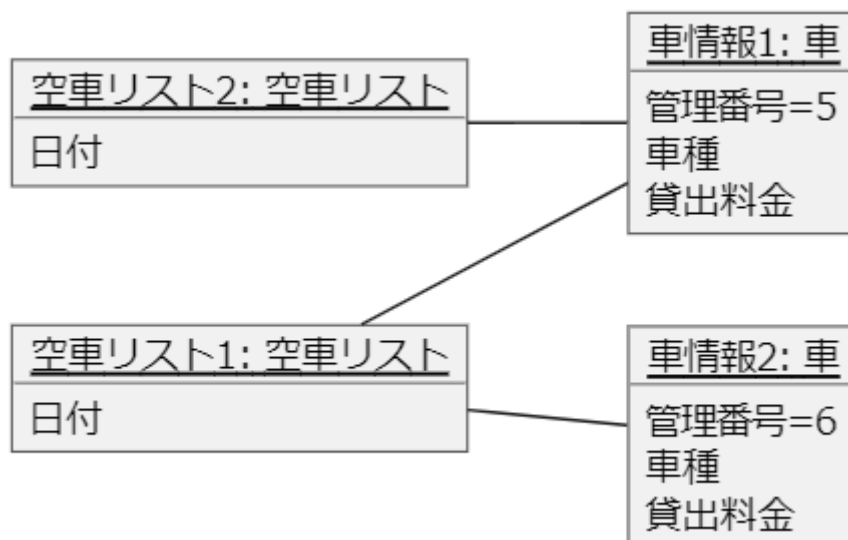


図 オブジェクト図

left to right direction

```
object "<u>空車リスト2: 空車リスト</u>" as k2 {  
  日付  
}  
  
object "<u>空車リスト1: 空車リスト</u>" as k1 {  
  日付  
}  
  
object "<u>車情報1: 車</u>" as car1 {  
  管理番号=5  
  車種  
  貸出料金  
}  
  
object "<u>車情報2: 車</u>" as car2 {  
  管理番号=6  
  車種  
  貸出料金  
}  
  
k1 -- car1  
k1 -- car2  
k2 -- car1  
  
@endum1
```

上記のオブジェクト図を利用して分析したオブジェクト間の関係を、クラス図にフィードバックすると下図になる。

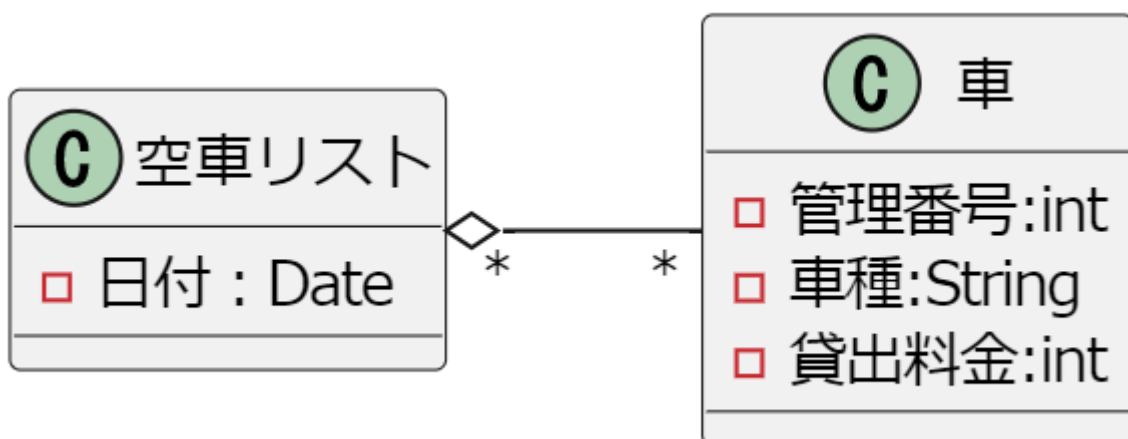


図 オブジェクト図の分析結果をフィードバックしたクラス図

以降、クラス図に反映するオブジェクトの数的関係をオブジェクト図で確認し、各クラス間にその関係を明記して、クラス図を洗練させていくことになる。

下記の表「クラスの定義」からオブジェクト図を作成し、分析を行ってクラス

表 クラスの定義

オブジェクト	クラス
希望日「20XX/11/20」の空いている車の一覧	空車リスト
当日の空いている車の一覧	
6号車:ワゴン 4WD	車
予約済みの車「7号車:ハッチバック 2WD」	
予約者「及川 源太」	顧客
借用者「大谷 翔平」	
予約番号「2001」の予約	予約
貸出	貸出

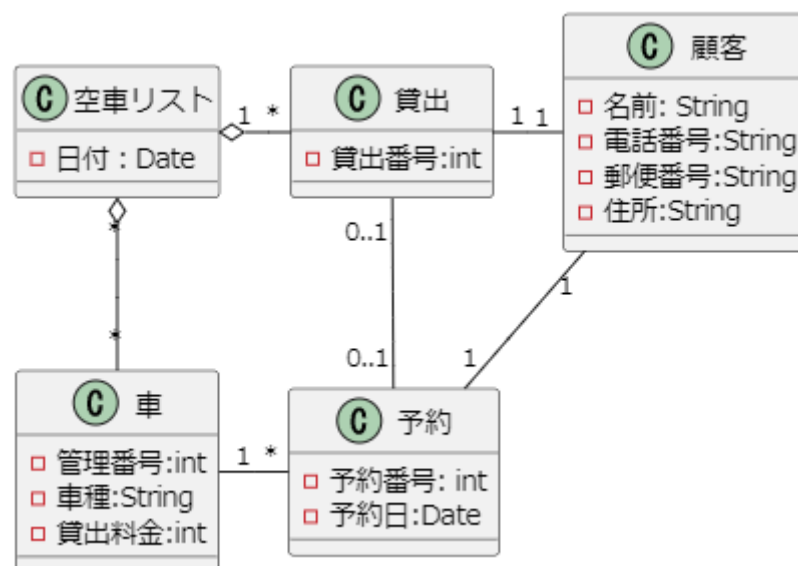


図 貸出管理システムのクラス図

```

@startuml
' left to right direction

class "空車リスト" as empty_car {
    - 日付 : Date
}

class "車" as car {
    - 管理番号:int
    - 車種:String
  
```

```

- 貸出料金:int
}

class "貸出" as rental {
- 貸出番号:int
}

class "顧客" as customer {
- 名前:String
- 電話番号:String
- 郵便番号:String
- 住所:String
}

class "予約" as reserve {
- 予約番号: int
- 予約日:Date
}

empty_car "1" o-ri- "*" rental
empty_car "*" o-down- "*" car

rental "0..1" -down- "0..1" reserve
car "1" -ri- "*" reserve

rental "1" -ri- "1" customer
reserve "1" -up- "1" customer

@enduml

```

## 演習3－1 次の要件に対して適切なクラス図を作成しなさい。

---

従業員には、派遣、契約社員、正社員の3つの雇用形態があります。

※3つの雇用形態を抽象化して汎化関係を表現してください。

【提出ファイル】演習3-1\_CL\_雇用形態.pu

## 演習3－2 次の要件に対して適切なクラス図を作成しなさい。

---

医師は患者を診察します。医師は診察のつどに、患者の診察記録を作成します。

患者には担当医が1人決まっています、担当医以外が診察することはありません。  
医師は1つの医局（外科、内科、小児科など）に所属しています。

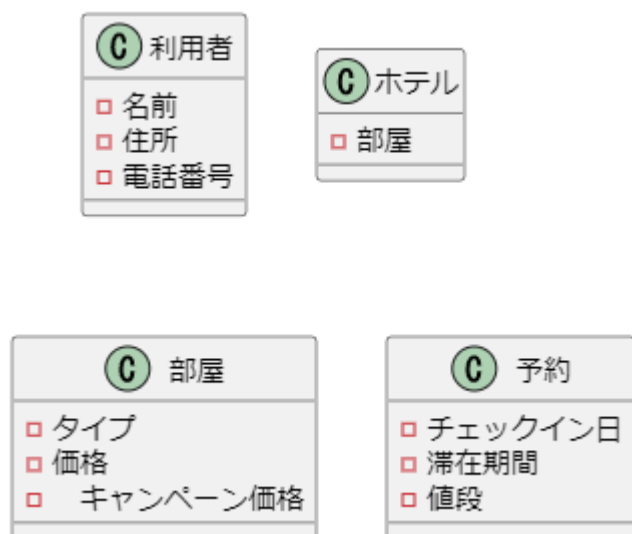
※多重度、関連名、関連端名なども反映してください。

【提出ファイル】演習3-2\_CL\_医師と患者と医局と診療記録.pu

## 演習3－3 次の要件に対して適切なクラス図を作成しなさい。

---

- ホテルには部屋が複数個（いくつかは不明）あります。
- ホテルには利用者が会員として登録されています。
- 利用者は部屋を予約します。



※集約、コンポジション、関連を適切に表現しましょう。

【提出ファイル】演習3-3\_CL\_ホテルと利用者と部屋と予約.pu

## 演習3－4

---

車にはタイヤが4本装着されている。車クラスとタイヤクラスを①オブジェクト図と②クラス図で表現しなさい。  
※関連の多重度をよく考えて作成してください。

①オブジェクト図          インスタンスされたオブジェクトすべてを記載する

【提出ファイル】演習3-4\_オブジェクト図.pu

②クラス図                  【提出ファイル】演習3-4\_クラス図.pu

## 演習3－5

次の文章の条件を満たすオブジェクト図を完成させなさい。なお、属性に具体的な値があるクラスは属性値を表記し、ないクラスは省略形で表記しなさい。

※演習3－3と連携した問題です。

AさんとBさんがホテルUMLに宿泊をしました。客室のタイプはシングル、宿泊費は一泊11,000円、キャンペーン価格9,000円の部屋です。チェックイン日は、5月3日で2泊の予定です。4月29日から5月6日まではキャンペーン期間です。システムはAさんの宿泊予定の客室を301号室、Bさんの宿泊予定の客室を303号室としました。

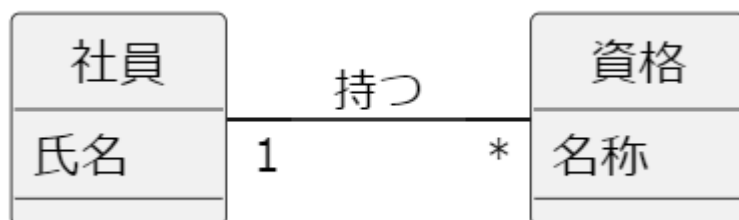
1. 予約のオブジェクトは、省略する

2. 部屋のオブジェクト名は、301号室、302号室、303号室とする

【提出ファイル】演習3-5\_OBJ\_ホテル宿泊予約.pu

## 演習3－6

下記の図は、社員の資格取得状況を記録するためのシステムのクラス図の一部です。次のシナリオに基づいて、オブジェクト図を記述しなさい。



【シナリオ】

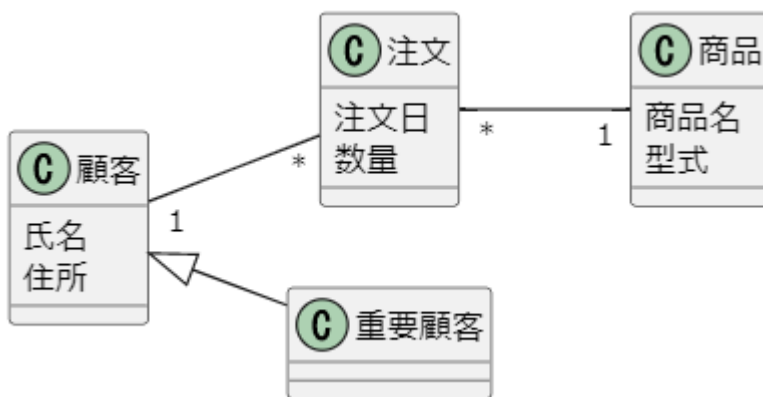


高橋洋平さんは、情報処理技術者試験の情報処理安全確保支援士の試験に合格し、**2020年10月21日**にそのことを申請しました。樋口正之さんもその試験に合格しています。さらに、高橋洋平さんは、**LinuC試験Level1**も合格し、**2021年3月28日**に会社に申し出ました。

【提出ファイル】演習3-6\_OBJ\_資格取得のオブジェクト図.pu

## 演習3－7

下記のクラス図は注文のモデルです。次のシナリオに基づいて、オブジェクト図を作成しなさい。



### 【シナリオ】

11月21日、重要顧客の高橋洋平さんが売り場に来られて、テレビ(型式**V12345**)を2台と冷蔵庫(型式**R8765**)1台注文していききました。12月23日高田利恵さんも同じ冷蔵庫を1台注文していききました。12月30日に大坪直之さんが来店し、売り場のプロジェクタ(型式**P1122**)を熱心に見ていましたが、残念ながら注文せずにお帰りになりました。

【提出ファイル】演習3-7\_OBJ\_注文のオブジェクト図.pu