第4章　シーケンス図

オブジェクト指向分析の公判では、前半で抽出したオブジェクトの相互作用を定義し、オブジェクト間のメッセージのやり取りをシーケンス図で定義する。

４－１　シーケンス図

シーケンス図は、オブジェクト間のメッセージのやり取りを時間軸に沿って表すための動的なダイアグラムである。

オブジェクト間の相互作用（Interaction）が、一定の時間軸に行われていることを明確に表現できる。

ユースケース記述で表現されたイベントフローの各ステップをメッセージ送信に置き換える。

オブジェクトから延びる縦軸で時間の経過を、縦軸間を結ぶ横軸の矢印でオブジェクト間のメッセージを表す。

例えば、次のようなユースケース記述の場合、シーケンス図では次のページのように表記される。

表　４－１

ユースケース図	ログインする
概要	このユースケースは登録会員によって開始され、登録会員の認証を行う。
アクター	登録会員
事前条件	会員情報が登録済みである。
事後条件	認証に成功すると、登録会員は「注文する」ユースケース、および、「会員情報を変更する」ユースケースが実行できる。

イベントフロー
【基本フロー】
1. 登録会員がログインを選択する
   「ログインする」ユースケースが開始される
2. システムはログイン情報（会員番号、パスワード）を要求する
3. 登録会員はログイン情報（会員番号、パスワード）を入力する
4. システムはログイン情報（会員番号、パスワード）を検証する （E-1）
5. システムはログイン成功を通知する

【代替フロー】
E－１：会員番号またはパスワードが違っていた
1. システムは会員番号またはパスワードが違っている旨を表示する
2. 登録会員はメッセージを確認し、「３．」に戻るかユースケースを終了する

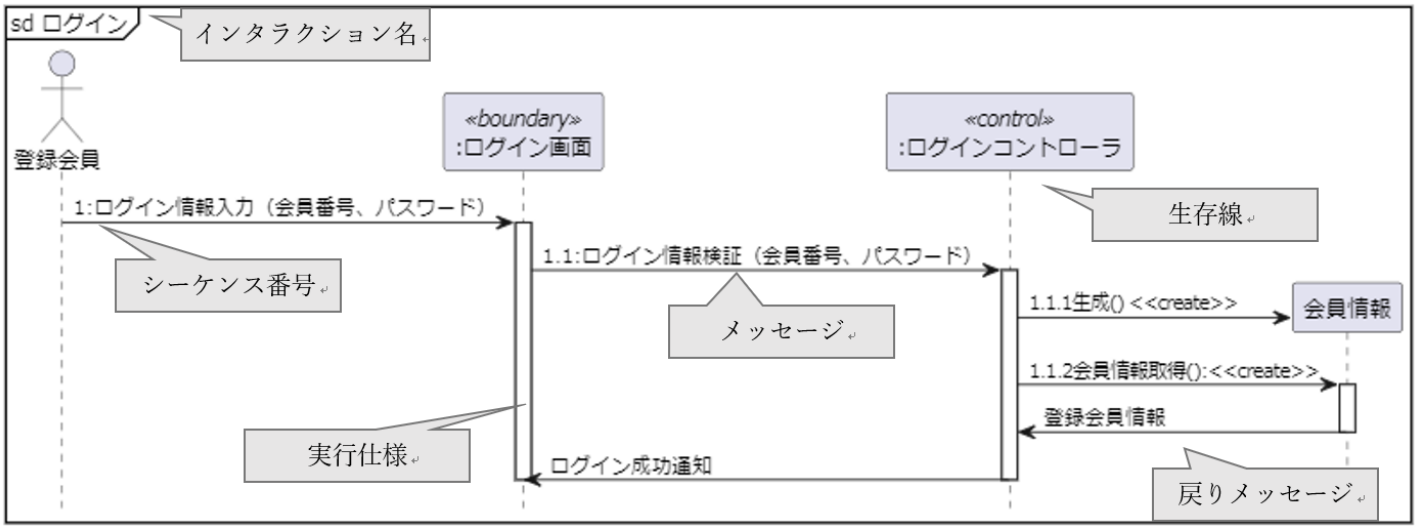
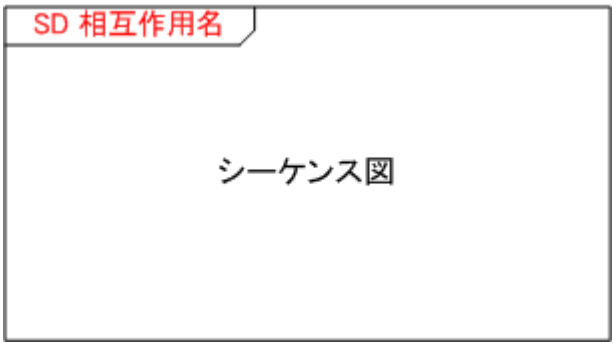


図４－１　シーケンス図

```
@startuml
hide footbox
Actor 登録会員  as member
participant ":ログイン画面" as boundary <<boundary>>
participant ":ログインコントローラ" as control <<control>>
participant "会員情報" as entity

mainframe sd ログイン

member -> boundary : 1:ログイン情報入力(会員番号、パスワード)
activate  boundary
    boundary -> control : 1.1:ログイン情報検証(会員番号、パスワード)


    activate control
    create entity
    control -> entity :1.1.1生成() <<create>>
        control -> entity : 1.1.2会員情報取得():<<create>>

        activate entity
            control <- entity : 登録会員情報
        deactivate entity

    boundary <- control : ログイン成功通知

    deactivate control

deactivate boundary


' control -> entity : <<create>>\n1.1.1生成():
' control -> entity : 1.1.2会員情報取得():<<create>>
' control <- entity : 登録会員情報


@enduml
```

※機能ごとに相互作用（Interaction）と呼ばれる下記のようなフレーム内に処理内容を記述する。インタラクション名のSDは　sqequence diagramの略である。



・交番は必ずしもユースケース記述と一致しない

※要求定義からオブジェクト指向分析までのフェースで抽出された、バウンダリ、コントロール、エンティティ（第3章　ロバネスト分析を参照）のオブジェクトとして登場させ、基本的な流れとしては、バウンダリ→コントロール→エンティティの順で進むようにするとよい。

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

▼生存線

生存線（ライフライン）は、シーケンス図に参加する要素を表す。

クラスからインスタンスされオブジェクトを長方形のボックスるやアクターを表すスティックマンを用いて、左からメッセージが送られる順に並べる。

下方に垂直に伸びる破線でオブジェクトが存在している期間を示す。

「:」の右側にオブジェクトのもととなるクラス名、左側に生成されたオブジェクトの名前を記述する。なお、オブジェクト名は、分析段階では明確ではなく省略する場合もある。

オブジェクトの消滅を表現する場合は、生存線の先に「×」をつける

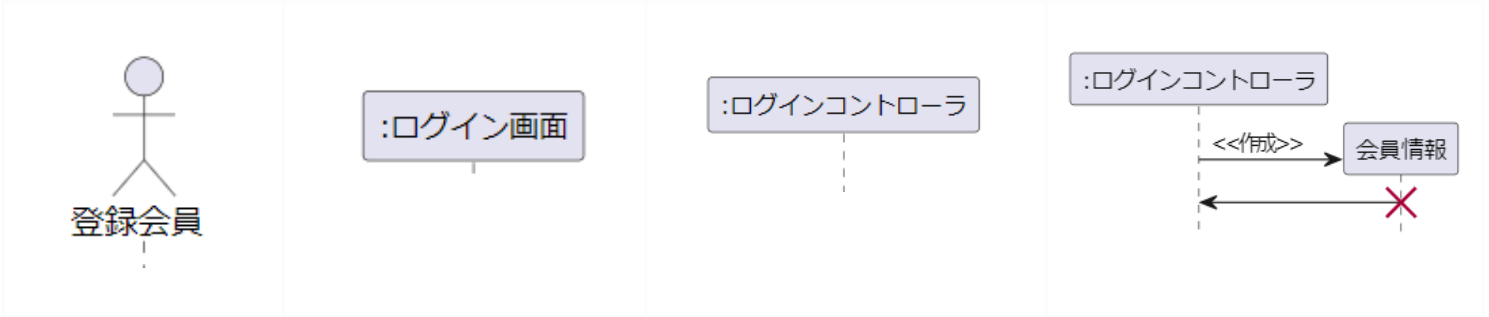


図４－２　生存線、オブジェクトの生成と消滅

```
@startuml
 hide footbox
 participant ":ログインコントローラ" as control
 participant "会員情報" as entity
 create entity
 control -> entity:<<作成>>
 control <- entity
destroy entity
@enduml
```

▼実行仕様（**Execution Specification**）

○実行仕様

生存線（ライフライン）上で、操作が実行されている時間（区間）を表現する。

メッセージの送信はメソッドの呼び出しと同等である。

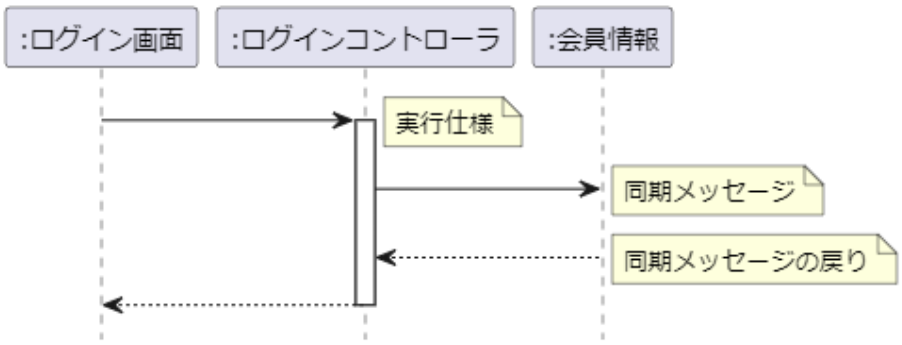


図４－３　実行仕様

```
@startuml
 hide footbox
participant ":ログイン画面" as boundary
participant ":ログインコントローラ" as control
```

```
participant ":会員情報" as entity

boundary -> control
  note right : 実行仕様

  activate control
   control -> entity
     note right: 同期メッセージ
   control <-- entity
     note right: 同期メッセージの戻り
  boundary <--- control
deactivate control


@enduml
```

〇同期メッセージ

メッセージの送信側が依頼した処理が終了するまで、次の操作へは移行しないメッセージを示す。

黒塗り三角形つきの実線で記述する。パラメータ（引数）を記述することも可能である。

〇同期メッセージの戻り

同期メッセージにより依頼した処理が終了するタイミングを示したもので、矢印つきの破線で示す。

戻り値を記述することもある。

〇非同期メッセージ

非同期のメッセージは、メッセージの送信側が依頼した処理の終了を待たずに、次の操作へ移行するメッセージを示す。

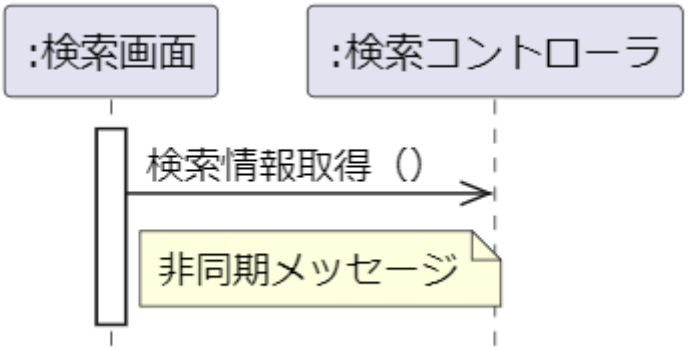


図４－４　非同期のメッセージ

```
@startuml
 hide footbox
skinparam style strictuml /' 厳密なUMLに準拠する'/
```

```
participant ":検索画面" as seach
participant ":検索コントローラ" as control

activate seach
  seach ->> control: 検索情報取得()
deactivate control

note right of seach : 非同期メッセージ


@enduml
```

○再帰呼出し

自分自身を呼ぶ同期メッセージ(オブジェクト自身が持つメソッドの呼び出し)は、再帰呼出しで表現する。

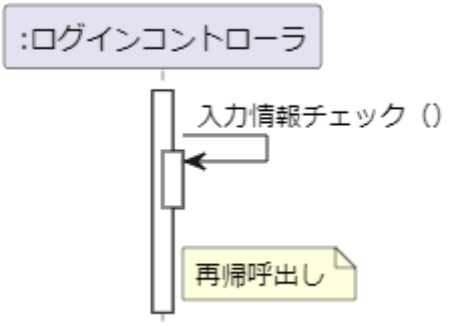


図 4-5 再帰呼出し

```
@startuml
 hide footbox
skinparam style strictuml /' 厳密なUMLに準拠する'/
participant ":ログインコントローラ" as control

activate control
    control -> control: 入力情報チェック()
    activate control
    ' deactivate control
note right of control : 再帰呼出し

deactivate control

@enduml
```

▼複合フラグメントとグループ化

シーケンス図では、制御構造を表現するために「複合フラグメント」を使用する。

以下の表は PlantUML で使える複合フラグメント。

キーワード	意味
ref(外部参照)	別のシーケンス図を参照することを示す
alt(オルタナティブ)/else	条件分岐
opt(オプション)	条件を満たした場合のみ実行する処理
par(パラレル)	並列処理
loop	ループ処理
break	中断処理
critical	マルチスレッド環境などでの排他制御

○**ref**(外部参照)を使用した例

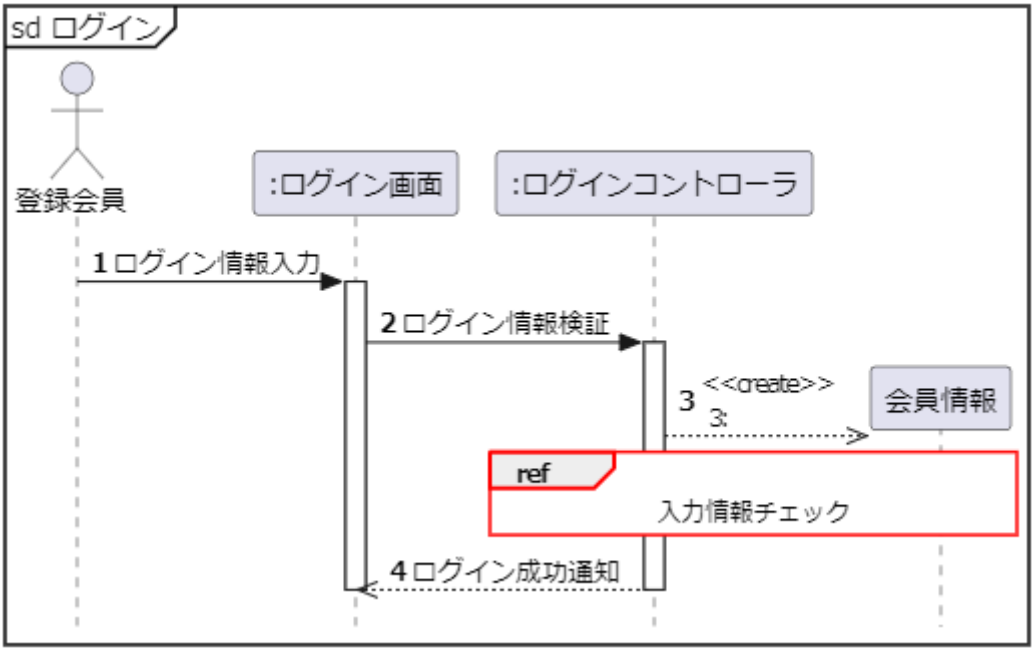


図4-6 「ref」を使用した例

```
@startuml
hide footbox
Actor 登録会員  as member
participant ":ログイン画面" as login
participant ":ログインコントローラ" as control
participant "会員情報" as entity

 'メッセージに自動で番号を降る
 autonumber
' 厳密なUMLに準拠する
skinparam style strictuml

' refの枠線
skinparam sequenceReferenceBorderColor red
' refの背景色
skinparam sequenceReferenceBackgroundColor white

' skinparam sequenceReferenceHeaderBackgroundColor lightblue

mainframe sd ログイン

member -> login : ログイン情報入力
activate  login
    login -> control : ログイン情報検証

    activate control
    create entity
    control -->> entity :<<create>>\n 3:

    ref over control, entity
        入力情報チェック
    end ref

    login <<-- control : ログイン成功通知

    deactivate control

deactivate login


@enduml
```

○**alt(**条件分岐)を使用した例

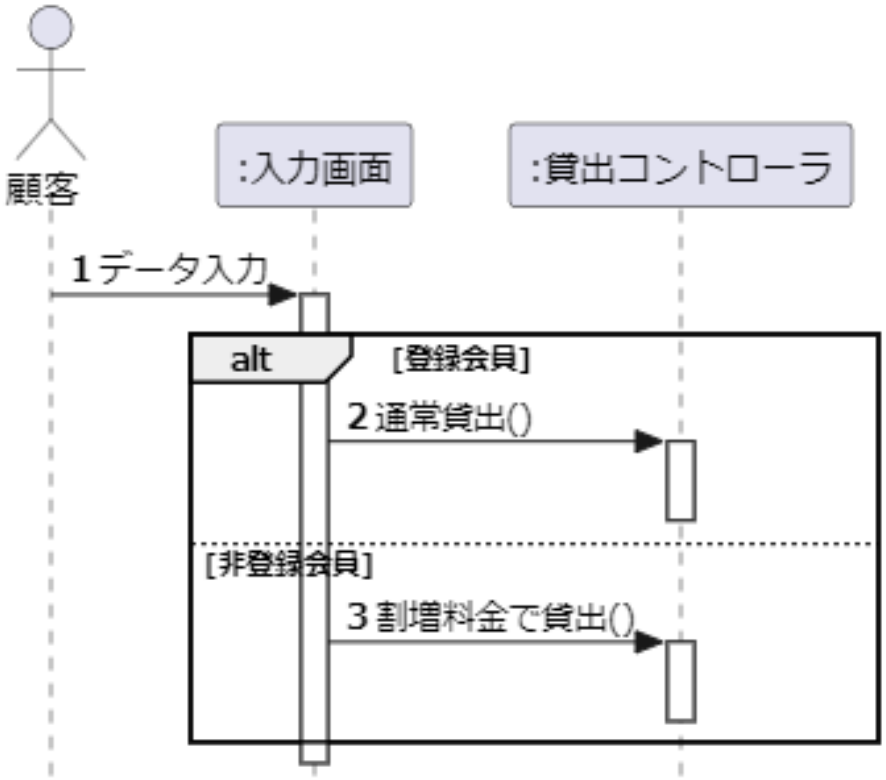


図4－7 「alt」を使用した例

[登録会員][非登録会員]をガード条件という。

ガード条件の記述方法には、規定はない。

```
@startuml
hide footbox
autonumber /'メッセージに自動で番号を降る '/
skinparam style strictuml /' 厳密なUMLに準拠する '/

Actor 顧客 as user
participant ":入力画面" as input
participant ":貸出コントローラ" as control

user -> input : データ入力
activate  input

  alt 登録会員
    input -> control:通常貸出()
    activate  control
    deactivate control
  else 非登録会員
    input -> control:割増料金で貸出()
    activate  control
    deactivate control
  end alt
```

```
deactivate control

@enduml
```

〇**opt**(条件を満たした場合のみ実行する処理)を使用した例

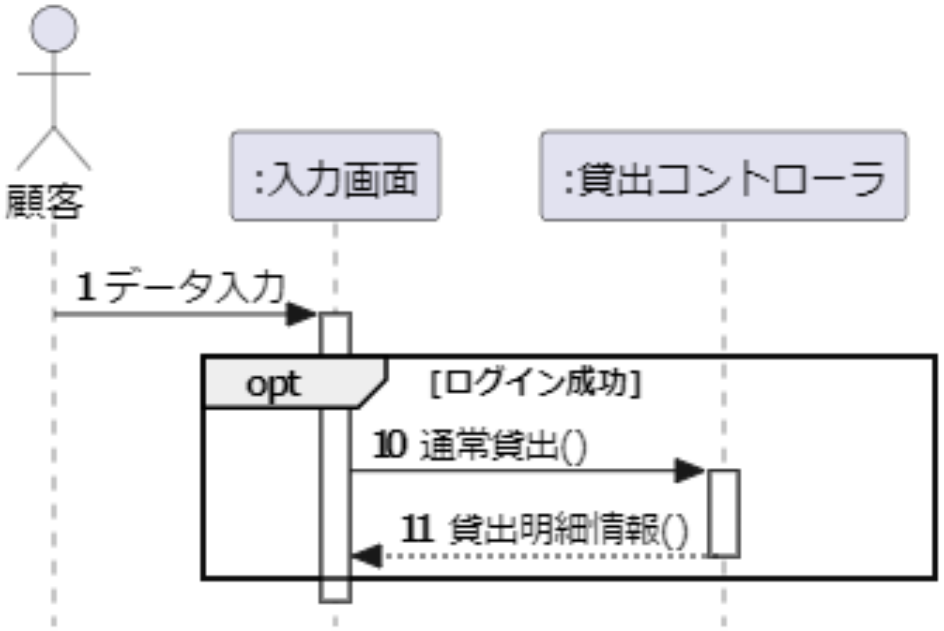


図4－8 「opt」を使用した例

```
@startuml
hide footbox

skinparam style strictuml /' 厳密なUMLに準拠する '/
Actor 顧客 as user
participant ":入力画面" as input
participant ":貸出コントローラ" as control

autonumber
user -> input : データ入力

activate  input
autonumber 10 /'メッセージに自動で番号を降る '/
  opt ログイン成功
    input -> control:通常貸出()
    activate  control
    input <-- control:貸出明細情報()
    deactivate control
  end alt

deactivate control
```

```
@enduml
```

○loop（ループ処理）を使用した例

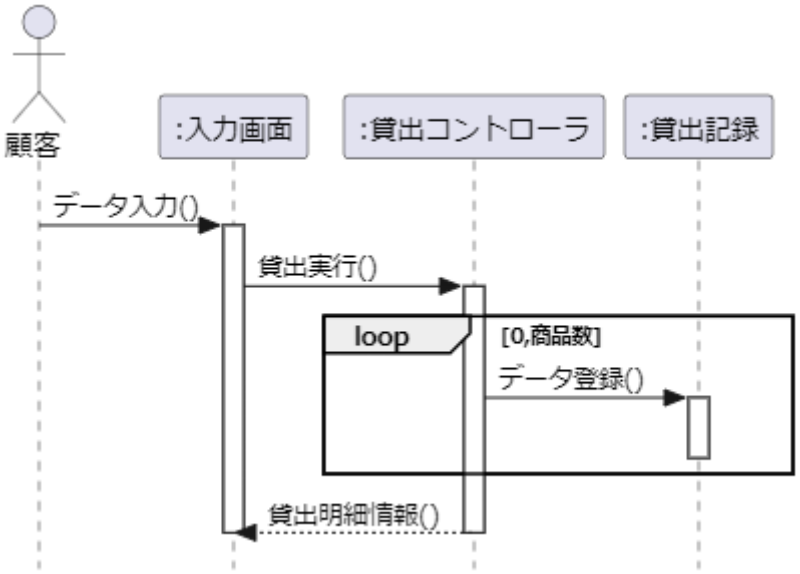


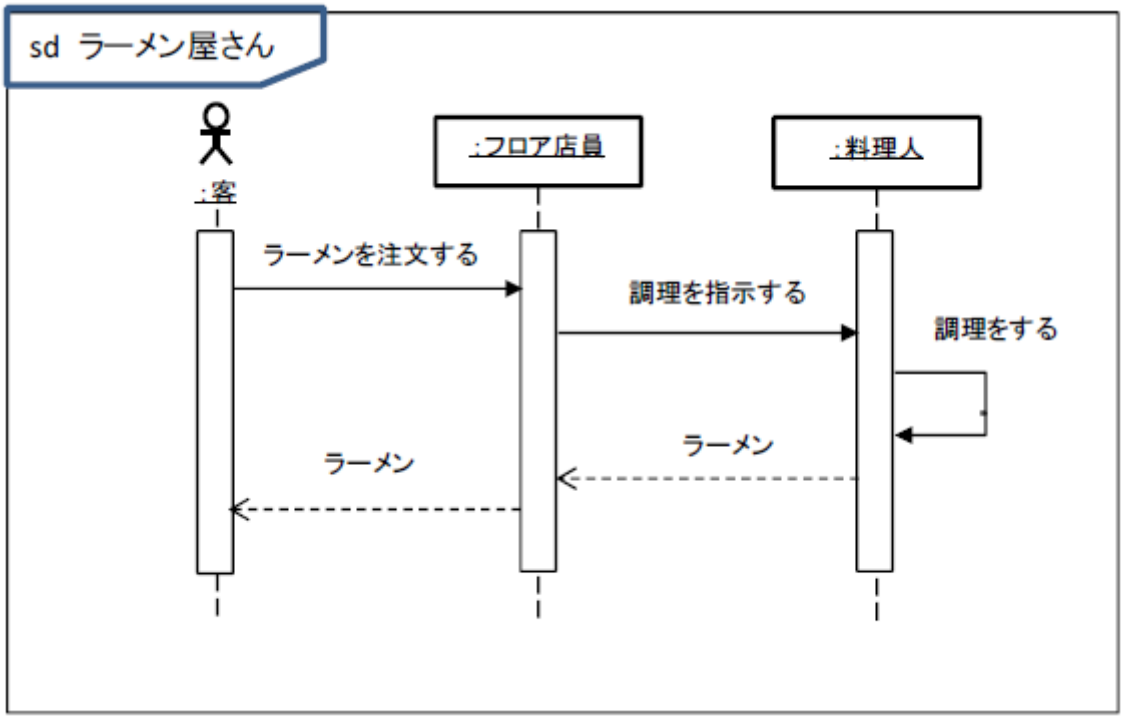
図４－９　loopを使用した例

```
@startuml
hide footbox

skinparam style strictuml /' 厳密なUMLに準拠する '/
Actor 顧客 as user
participant ":入力画面" as input
participant ":貸出コントローラ" as control
participant ":貸出記録" as record

user -> input : データ入力()
activate  input
  input -> control : 貸出実行()
  activate  control

  loop 0,商品数
    control -> record:データ登録()
    activate  record
    deactivate record
  end loop
input <-- control : 貸出明細情報()
deactivate  control

deactivate input

@enduml
```

[0, 商品数] 開始:0　終了:商品数　を示す

演習４－１

ラーメン屋さんで、客が注文してから配膳されるまでのシーケンス図を作成しなさい。



【提出ファイル】演習**4-1_SD_**ラーメン屋さんでの注文**.pu**

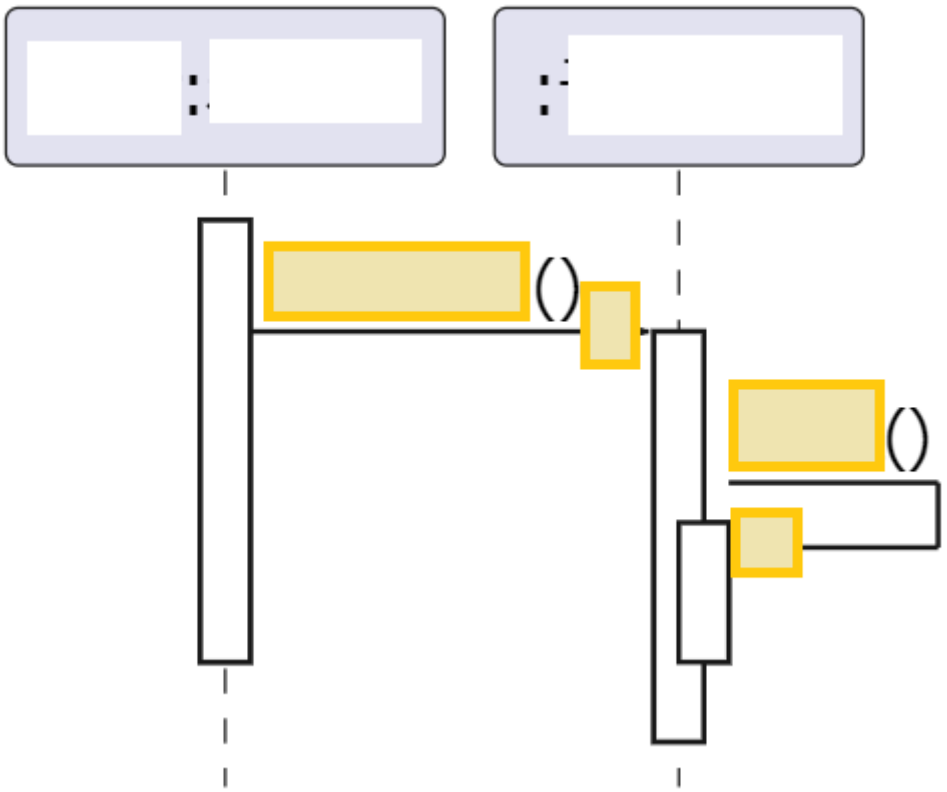
演習４－２

次のシーケンスを満たすように、シーケンス図を作成しなさい。同期メッセージに関してはリターンも表記すること。

1.利用者クラスのオブジェクトuserは、ログインを行う（予約画面にログインメッセージを送る）
また、ログインメッセージは非同期とする

2. 次に予約画面クラスのオブジェクトは、自分自身の確認操作(認証）を呼び出す（同期）

【提出ファイル】演習**4-2_SD_**利用者と予約画面**.pu**



演習４－３

次のシーケンスを満たすように、シーケンス図を作成しなさい。

1. データ更新コントローラクラスのオブジェクトcontrollerは、ホテルクラスの予約操作を呼び出す
2. ホテルクラスのオブジェクトhotelは、予約オブジェクトを作成する
3. ホテルクラスのオブジェクトhotelは、自身の予約登録操作を呼び出す

【提出ファイル】演習**4-3_SD_**データ更新コントローラとホテルと予約**.pu**

演習４－４

次の要件に対して適切なシーケンス図を作成しなさい。

要件）「商品を注文する」シーケンスです

受注係は注文登録画面で商品情報を確認し、注文情報の登録を行います。

・アクター：受注係

・バウンダリ：注文登録画面

〇商品情報確認（）

〇商品情報表示（商品情報）：再帰呼出し

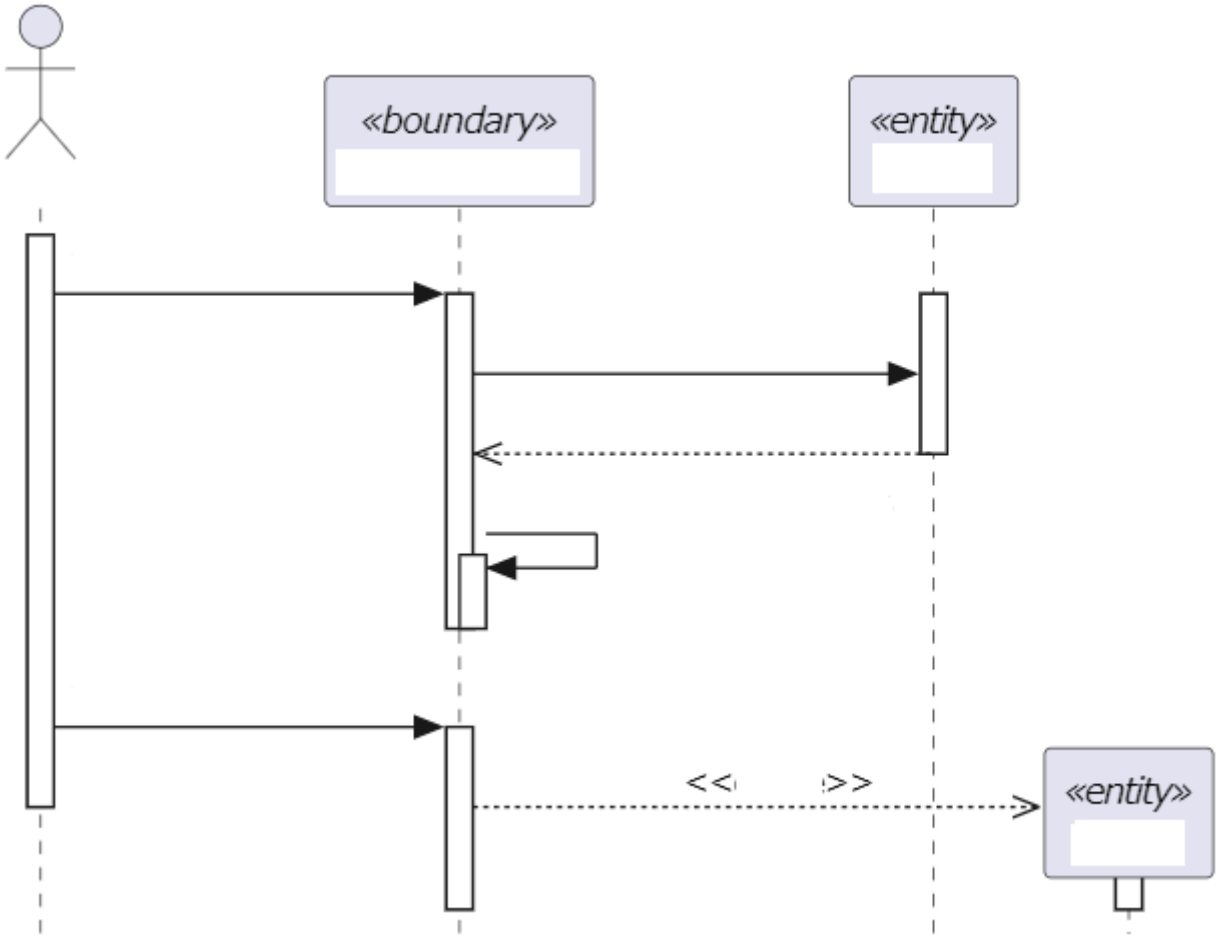
〇商品登録（注文情報）：注文オブジェクトを生成する

・エンティティ：商品

〇商品情報取得（）：商品情報を返す

・エンティティ：注文

ヒント）



【提出ファイル】演習**4-4_SD_**商品注文**.pu**

演習４－５

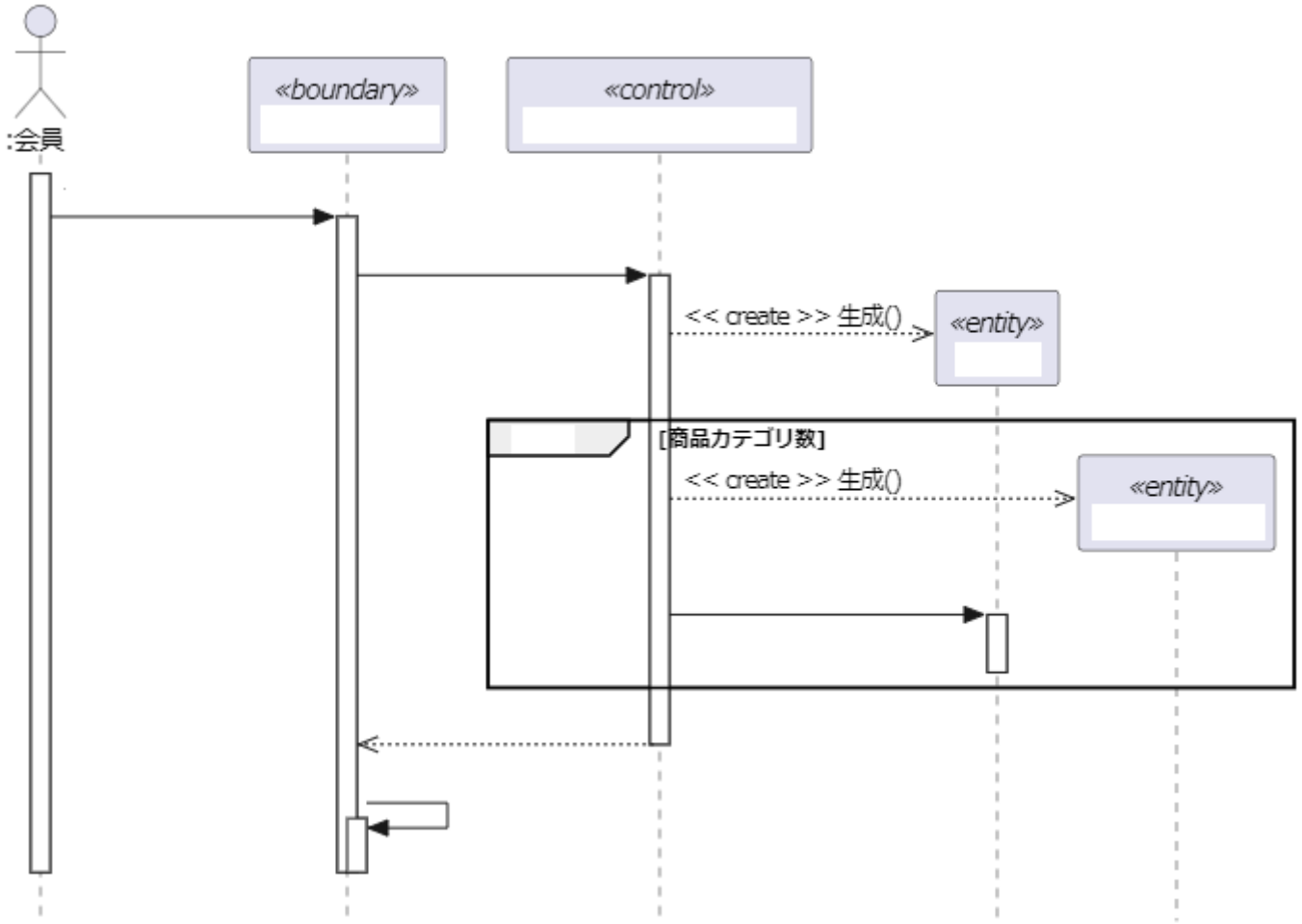
次の要件に対して適切なシーケンス図を作成しなさい。

要件)「商品情報を検索する」シーケンス

会員は商品検索画面で商品検索をし、検索結果として商品情報を確認することができます。

商品には検索する商品カテゴリが関連します。

ヒント)



【提出ファイル】演習**4-5_SD_**商品情報検索**.pu**

演習４－６

次のシナリオはWeb ページで画像を検索して閲覧するまでの仕組みです。このシナリオからシーケンス図を描いてください。

(シナリオ)

① 登場するオブジェクトは、ユーザー、ブラウザ、Web サーバー、画像サーバー、です。

② ユーザーはブラウザ画面から、閲覧したい画像のリンクボタンを押しました。

③ ブラウザはその画像を探すように画像のファイル名をWeb サーバーに伝えます。

④ Web サーバーは画像サーバーから該当する画像ファイルを探します。
⑤ 見つかった画像ファイルはWeb サーバーで閲覧可能な状態にします。
⑥ ブラウザはWeb サーバーからの画像ファイルを再生します。

【提出ファイル】演習4-6_SD_Web画像検索.pu