

PlantUML

UML図を描こう！

何のツールを使いますか？

- astah*
- Office Visio
- ...
- PlantUML

新たな選択肢を！

Outline

1. PlantUMLのいいこと
2. PlantUMLでの記法

PlantUMLとは

テキストベースでUMLを描けるツール

<http://ja.plantuml.com/>

example

title 『サラダ記念日(俵万智)』 より

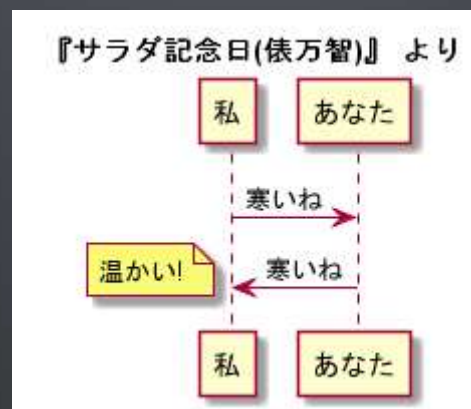
私 -> あなた : 寒いね

あなた -> 私 : 寒いね

note left

温かい!

end note



PlantUMLのいいこと
差分が見やすい

バイナリファイルの場合 (GitHubの場合)

仲間モンスターを追加

プレイヤーは、仲間モンスターのリストを管理する

[Browse files](#)

 master

 **Geroshabu** committed 3 minutes ago

1 parent [9916bcb](#) commit [65843123d34aa85773581bfa8a5609f0e326e679](#)

 Showing **2 changed files** with 0 additions and 0 deletions.

[Unified](#) [Split](#)

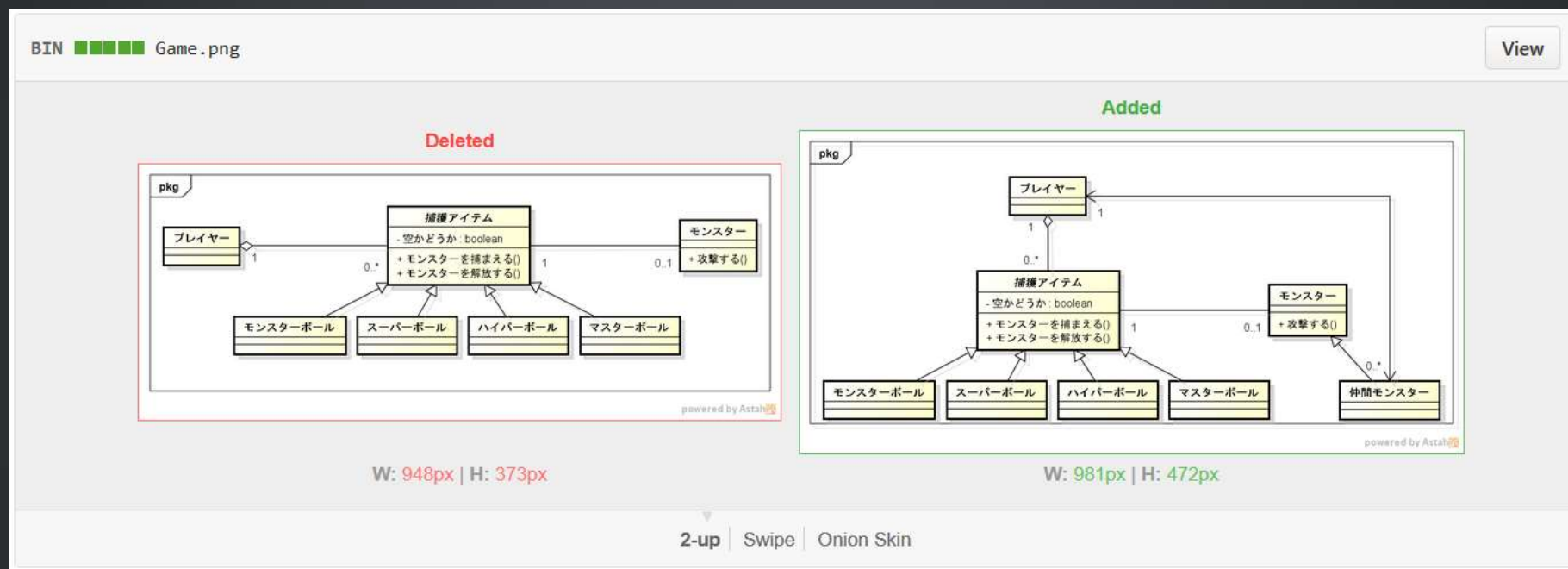
BIN  Game.asta

[View](#)

Binary file not shown. ← バイナリファイルなので差分は見せられないよ！

さすがGitHub

画像なら比較することもできるが...



どこが変わっているか分からない

...というか, なんか全体的に変わってる気がする

テキストファイルの場合

3

■■■■ diff_sample.plantum1

View

✱	@@ -4,5 +4,8 @@ abstract ????????????????????	
4	捕獲アイテム < -- ハイパーボール	4 捕獲アイテム < -- ハイパーボール
5	捕獲アイテム < -- マスターボール	5 捕獲アイテム < -- マスターボール
6		6
		7 +モンスター < -- 仲間モンスター
		8 +
7	プレイヤー "1" o-- "0..*" 捕獲アイテム	9 プレイヤー "1" o-- "0..*" 捕獲アイテム
8	捕獲アイテム "1" -- "0..1" モンスター	10 捕獲アイテム "1" -- "0..1" モンスター
		11 +プレイヤー "1" <--> "0..*" 仲間モンスター

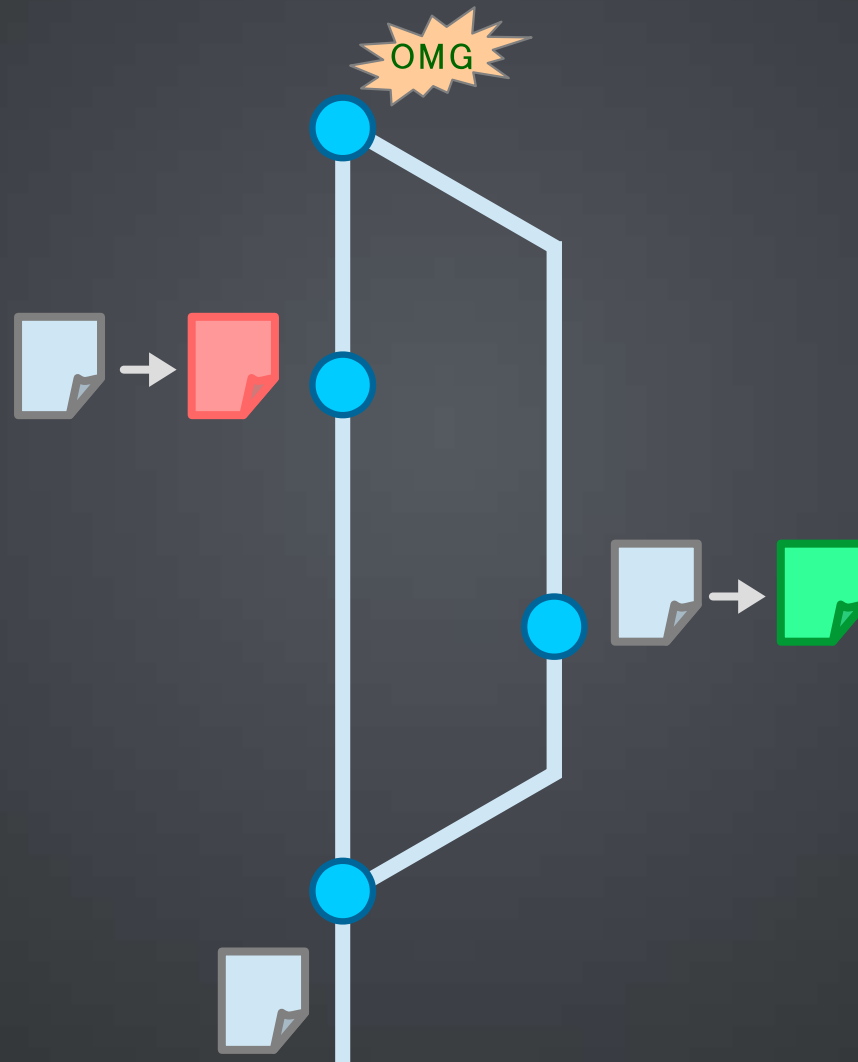
変わっている所が分かること、大事！

そして

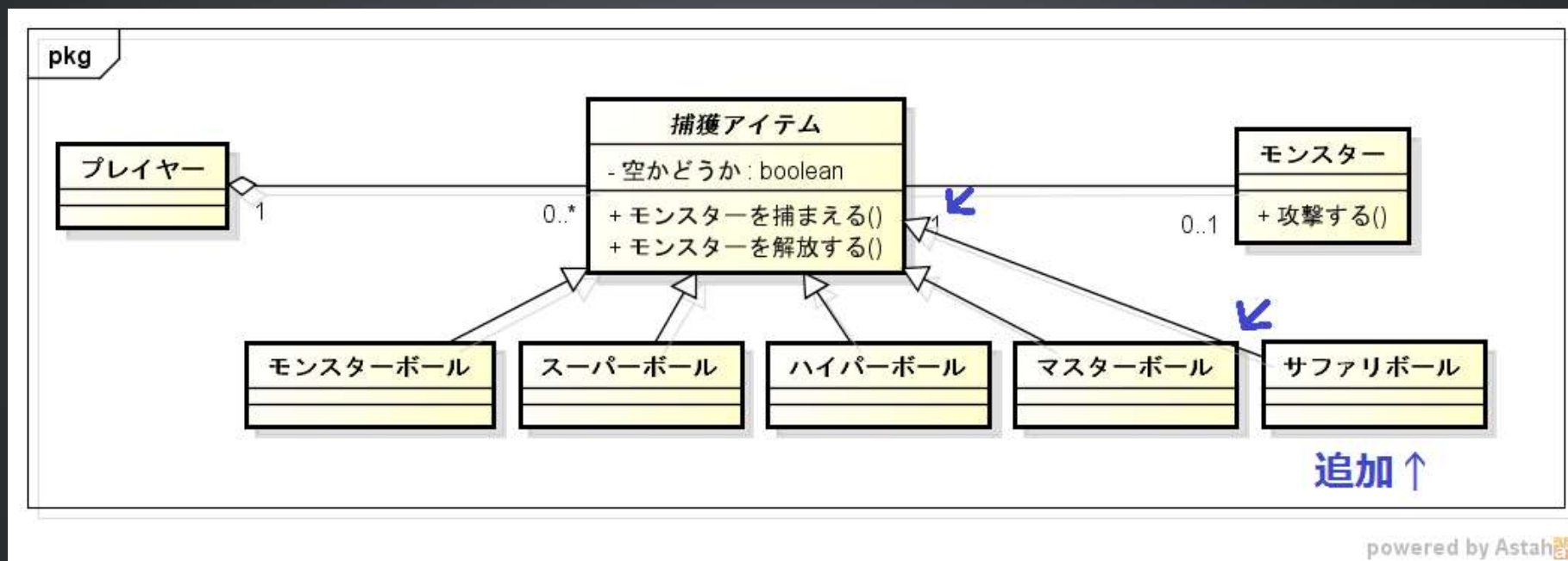
変わっていない所が分かること、大事！

その他のいいこと

ファイルの競合を恐れなくてよい



"きれいに见せる"ことに気を使わなくてよい



行に対してコメントできたり...

```
9   プレイヤー "1" o-- "0..*" 捕獲アイテム  
10  捕獲アイテム "1" -- "0..1" モンスター  
11  +プレイヤー "1" <--> "0..*" 仲間モンスター
```



Geroshabu added a note a minute ago

Owner

...これまずくね？

Add a line note

コード中にも記述できたりする

```
/// @brief 捕獲アイテムを使う
/// @startuml{UseCaptureItem.png}
///     私 --> 捕獲アイテム : アイテムを手を持つ
///     私 --> モンスター : アイテムをぶつける
/// @enduml
public void UseCaptureItem()
{
    ...
}
```

Doxygen連携で, UML図付きのAPIリファレンスも

Outline

1. PlantUMLのいいこと
2. PlantUMLでの記法

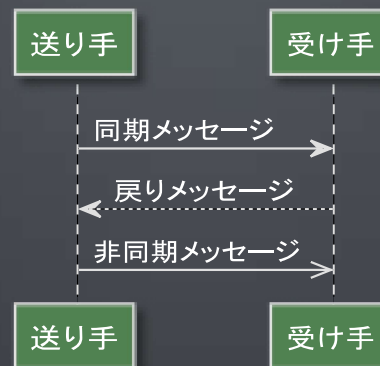
PlantUMLでの記法

- シーケンス図・クラス図・ユースケース図...etc
- なんの図かは自動判定

シーケンス図1

メッセージ

送り手 -> 受け手 : 同期メッセージ
受け手 --> 送り手 : 戻りメッセージ
送り手 ->> 受け手 : 非同期メッセージ



シーケンス図2

作成と破棄

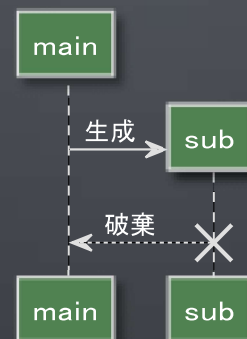
```
participant main
```

```
create sub
```

```
main -> sub : 生成
```

```
sub --> main : 破棄
```

```
destroy sub
```



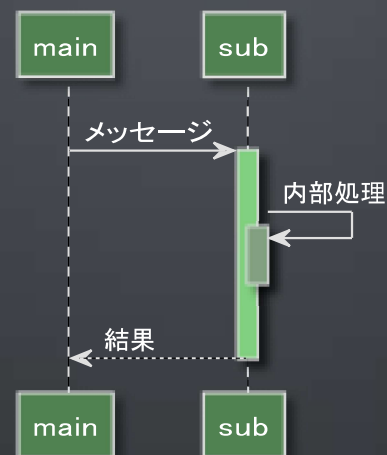
シーケンス図3

実行状態

```
main -> sub : メッセージ  
activate sub
```

```
sub -> sub : 内部処理  
activate sub  
deactivate sub
```

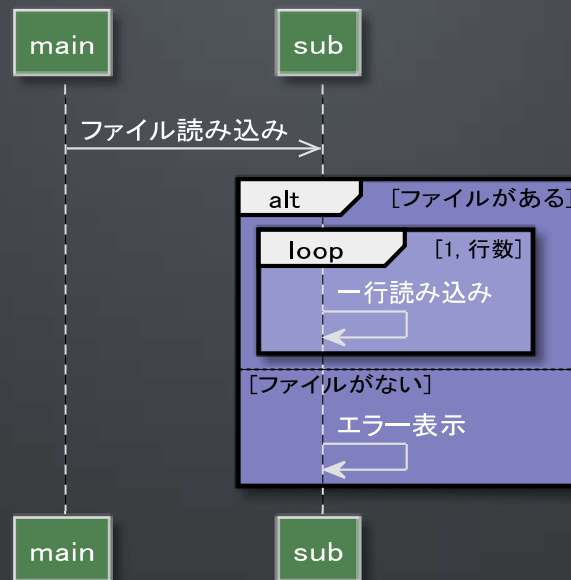
```
sub --> main : 結果  
deactivate sub
```



シーケンス図4

複合フラグメント

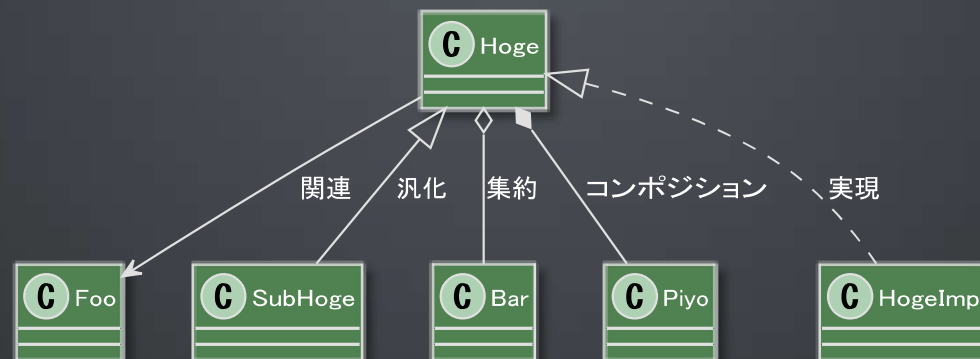
```
alt ファイルがある
  loop 1, 行数
    sub -> sub : 一行読み込み
  end
else ファイルがない
  sub -> sub : エラー表示
end
```



クラス図1

クラスの関係

Hoge --> Foo : 関連
Hoge <|-- SubHoge : 汎化
Hoge o-- Bar : 集約
Hoge *-- Piyo : コンポジション
Hoge <|.. HogeImpl : 実現

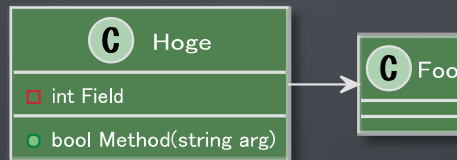


クラス図2

クラスの定義

```
class Hoge {  
  - int Field  
  + bool Method(string arg)  
}
```

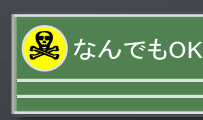
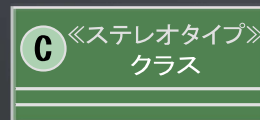
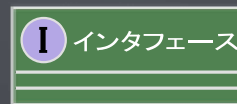
Hoge -> Foo



クラス図3

クラスの種類

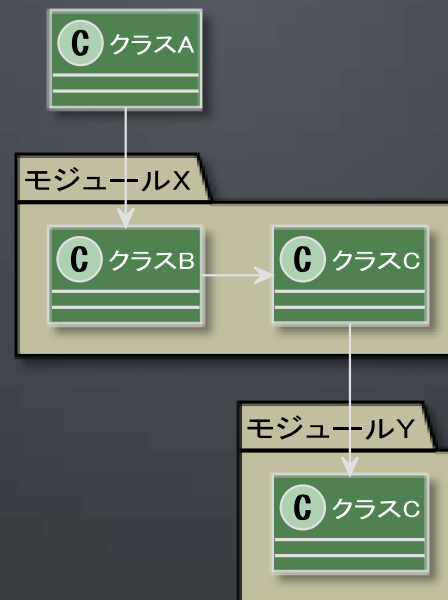
abstract 抽象クラス
interface インタフェース
enum 列挙型
class クラス <<ステレオタイプ>>
class なんでもOK << (💀, yellow) >>



クラス図4

パッケージ

```
クラスA -> モジュールX.クラスB  
namespace モジュールX {  
  クラスB -> クラスC  
  クラスC -> モジュールY.クラスC  
}
```

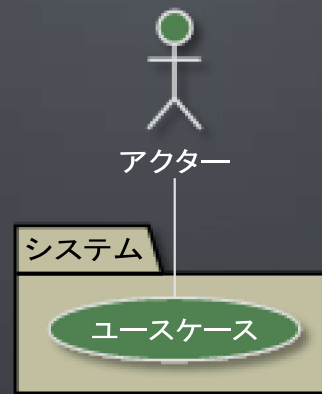


ユースケース図

アクターとユースケース

:アクター:

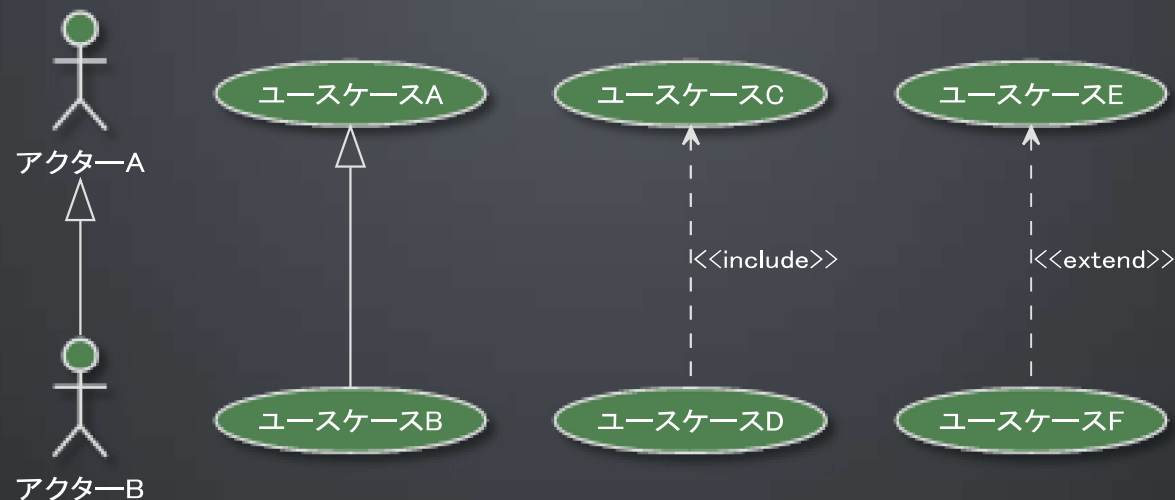
```
package "システム" {  
    アクター -- (ユースケース)  
}
```



ユースケース図

汎化、包含、拡張

:アクターA: <|-- :アクターB:
(ユースケースA) <|-- (ユースケースB)
(ユースケースC) <.. (ユースケースD) : <<include>>
(ユースケースE) <.. (ユースケースF) : <<extend>>



アクティビティ図

アクション、開始、終了

(*) --> アクション
--> (*)

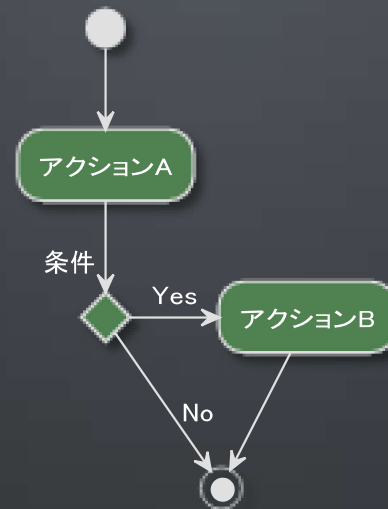


アクティビティ図

分岐

(*) --> アクションA

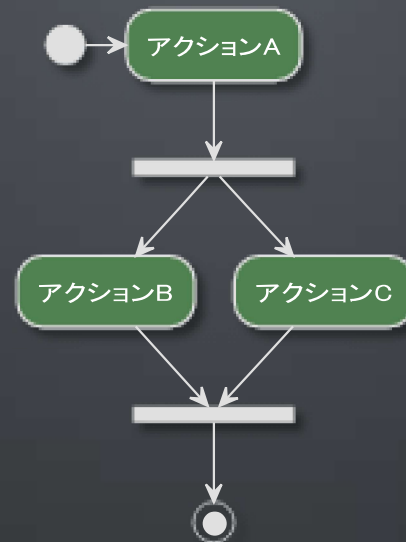
```
if "条件" then
  -> [Yes] アクションB
  --> (*)
else
  --> [No] (*)
endif
```



アクティビティ図

並列処理

```
(*) -> アクションA
--> ===Fork===
--> アクションB
--> ===Join===
===Fork=== --> アクションC
--> ===Join===
--> (*)
```

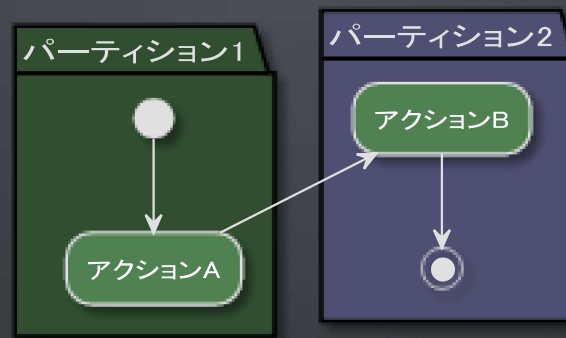


アクティビティ図

パーティション

```
partition パーティション1 #305030 {  
    (*) --> アクションA  
}
```

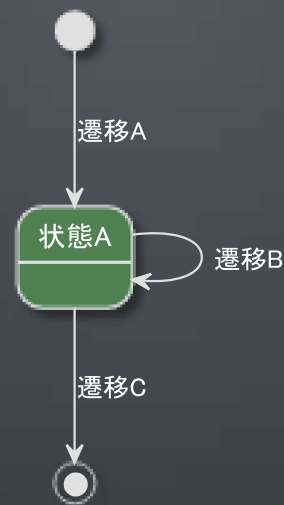
```
partition パーティション2 #505070 {  
    -up-> アクションB  
    --> (*)  
}
```



ステートマシン図

遷移

[*] --> 状態A : 遷移A
状態A --> 状態A : 遷移B
状態A --> [*] : 遷移C

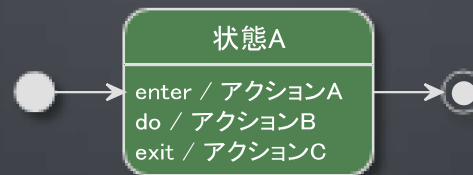


ステートマシン図

状態の定義

```
state 状態A : enter / アクションA  
state 状態A : do / アクションB  
state 状態A : exit / アクションC
```

```
[*] -> 状態A  
状態A -> [*]
```



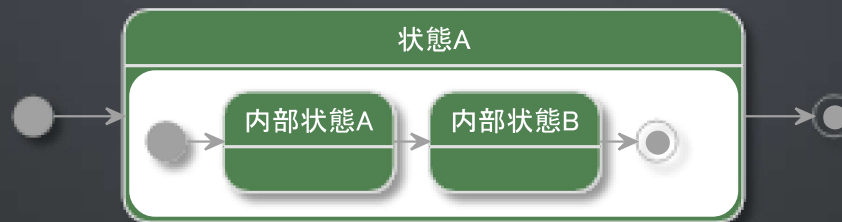
ステートマシン図

複合状態

[*] --> 状態A

```
state 状態A {  
  [*] -> 内部状態A  
  内部状態A -> 内部状態B  
  内部状態B -> [*]  
}
```

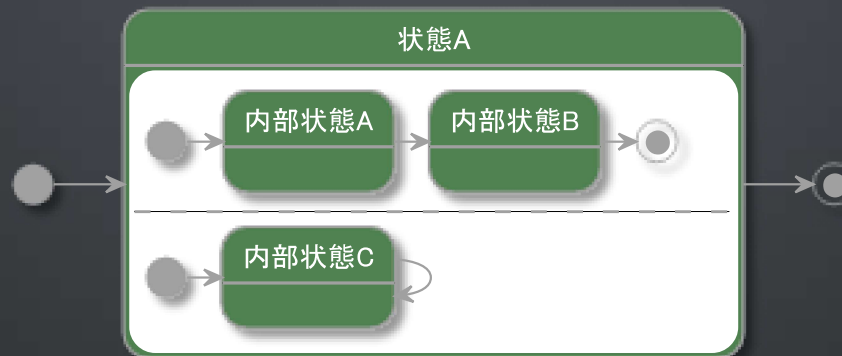
状態A --> [*]



ステートマシン図

独立

```
[*] -> 状態A
state 状態A {
  [*] -> 内部状態A
  内部状態A -> 内部状態B
  内部状態B -> [*]
  --
  [*] -> 内部状態C
  内部状態C -> 内部状態C
}
状態A -> [*]
```



thanks_for_listening.plantuml

1 私 -> 皆さん : ごせいち

2

ご成長

ご清聴ありがとう

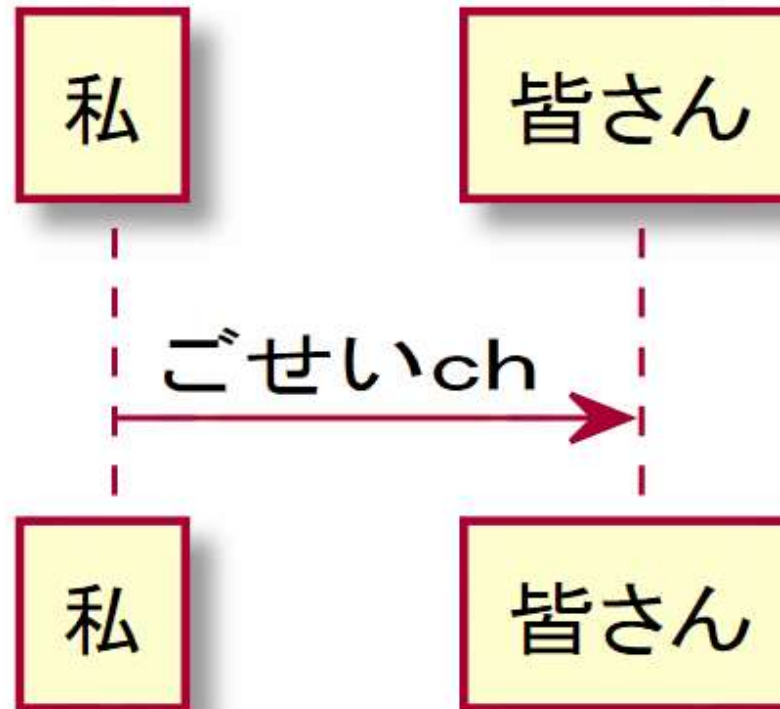
ご清聴ありがとうございました

ご請求

ご清栄のことと

Tab キーで予測候補を選択

thanks_for_listening.plantuml View



(1, 5)

CRLF UTF-8 PlantUML



(PlantUML with Atom)