

# State Pattern

# 0. 청중

청중 = 디자인 패턴에 대해 모르는 사람..

말투도 그에 맞췄기에.. 불쾌해하지 않으셨으면 좋겠습니다 ㅠ.ㅠ



# 1. 목차

1. 상태 패턴이 필요한 이유 with example
  1. Tetris 생각해보기
  2. Tetris + 뿌요뿌요 생각해보기
2. 상태 패턴 정의해보기
3. 상태 패턴 다이어그램 이해해보기
4. 내가 생각한 장단점 & TMI,,

그럼 시작합니다



# 1. State Pattern이 필요한 이유

```
fun main() {
```

```
// 여기다가  
// Tetris 만들자 . . . !
```





망곰이의 과제! - main 함수 안에다가 테트리스 만들기

# 1. State Pattern이 필요한 이유 - 테트리스 만들기

과제 요구사항



# 1. State Pattern이 필요한 이유 - 테트리스 만들기

|  |  |  | Play Tetris<br>Button Clicked? | Game Over? |
|--|--|--|--------------------------------|------------|
|  | HOME   |  | X                              | X          |
|  | <br>Tetris! |  | O                              | X          |
|  | <br>Result  |  | .                              | O          |

# 1. State Pattern이 필요한 이유 - 테트리스 만들기

```
/**
 * 망곰이의 과제 - 테트리스 만들어봐요
 */
fun main() {
    var isPlayPressed: Boolean = false
    var isGameOver: Boolean = false

    // H.W. Logic!

    if (!isPlayPressed && !isGameOver) {

        // something about home

    } else if (isPlayPressed && !isGameOver) {

        // 테트리스 Play! 🎮

    } else if (isGameOver) {

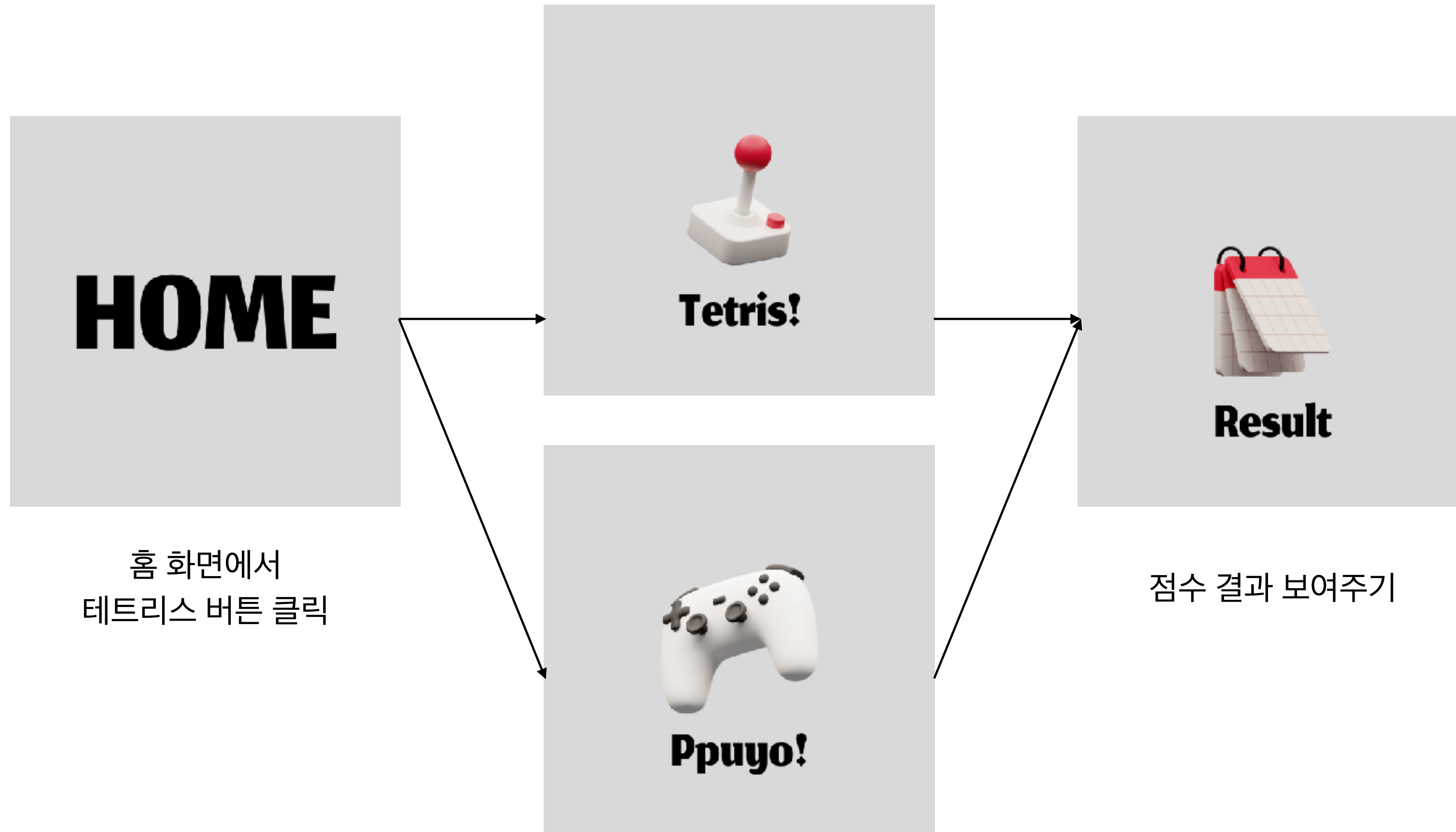
        // 테트리스 점수! 🎮 100

    }
}
```



# 1. State Pattern이 필요한 이유 - 뽀요뽀요 추가하기

교수님: 거기다가 그대로 뽀요뽀요를 만들어오도록 하세요





# 1. State Pattern이 필요한 이유 - 뽀요뽀요 추가하기

|  |  |  | Play Tetris<br>Button Clicked? | Play Ppuyo<br>Button Clicked? | Game Over? |
|--|--|--|--------------------------------|-------------------------------|------------|
|  | HOME   |  | X                              | X                             | X          |
|  | <br>Tetris! |  | O                              | X                             | X          |
|  | <br>Ppuyo!  |  | X.                             | O                             | X          |
|  | <br>Result  |  | .                              | .                             | O          |

```

/**
 * 망곰이의 과제 - 뿌요뿌요를 추가해봐요
 */
fun main() {
    var isPlayPressed: Boolean = false
    var isGameOver: Boolean = false

    // H.W. Logic!
    if (!isPlayPressed && !isGameOver) {

        // something about home

    } else if (isPlayPressed && !isGameOver) {

        if (isPlayPressed.type == "Puyo") {
            // 뿌요 Play! 🎮
        }

        if (isPlayPressed.type == "Tetris") {
            // 테트리스 Play! 🎮
        }

    } else if (isPlayPressed && isGameOver) {

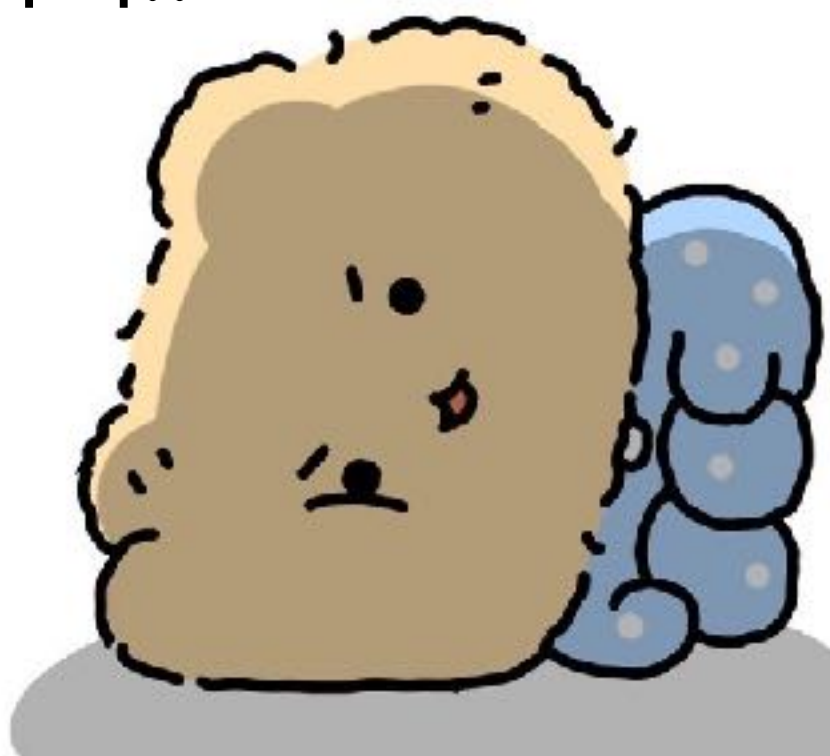
        if (isPlayPressed.type == "Puyo") {
            // 뿌요 점수! 🎮 100
        }

        if (isPlayPressed.type == "Tetris") {
            // 테트리스 점수! 🎮 100
        }

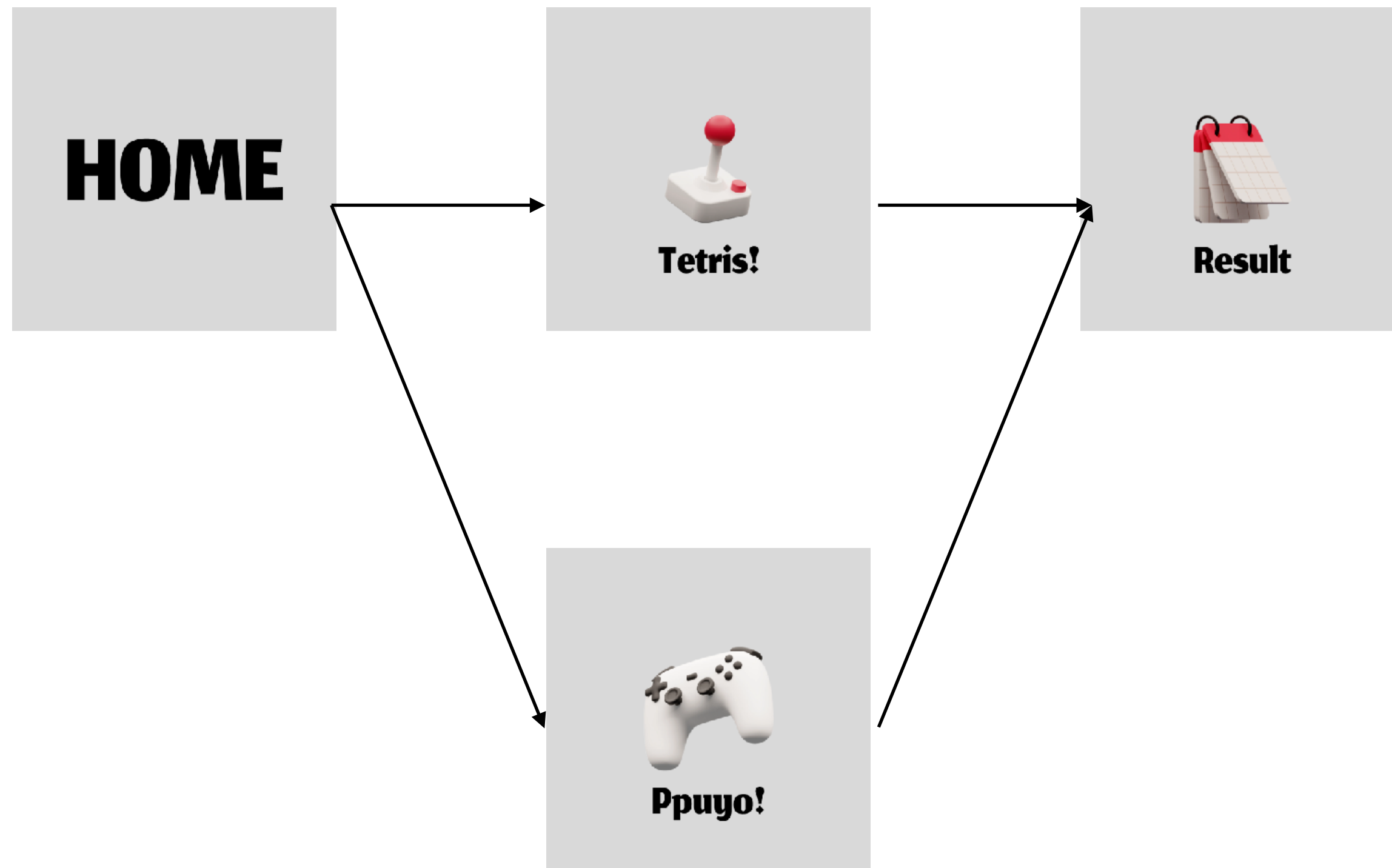
    }
}

```

스파이더 만들라하면  
어떡하지..



# 1. State Pattern이 필요한 이유 - 뽕요뽕요 추가하기



게임기의 단계는 명확하고,  
사용자가 어느 단계에 있느냐에 따라 해줄 일이 명확함.

1. UI 만들어주기
2. 사용자의 이벤트 감지해주기
  1. 현재 화면에서 작동해주기
  2. 다음 스텝으로 넘어가기

# 1. State Pattern이 필요한 이유 - 뽕요뽕요 추가하기 우아하게요!



```
/**
 * 각 게임의 단계
 */
abstract class GameStep {

    // GameStep's 해야할 일
    init {
        configureUI()
        addEventHandlers()
    }

    // UI 만들어주기
    abstract fun configureUI()

    // 사용자의 이벤트 감지해주기
    abstract fun addEventHandlers()
}
```

**HOME**



**Tetris!**



**Puyo!**



**Result**



# 1. State Pattern이 필요한 이유 - 뽕뽕 추가하기 우아하게요!

```
/**
 * 각 게임의 단계
 */
abstract class GameStep {

    // GameStep's 해야할 일
    init {
        configureUI()
        addEventHandlers()
    }

    // UI 만들어주기
    abstract fun configureUI()

    // 사용자의 이벤트 감지해주기
    abstract fun addEventHandlers()
}
```

**HOME**

1. tetris, ppuyo 버튼 두 개 있는 UI 만들기
2. 버튼 이벤트 보고 Tetris, Ppuyo Game step으로 넘어가주기

# 1. State Pattern이 필요한 이유 - 뽕요뽕요 추가하기 우아하게요!

```
/**
 * 각 게임의 단계
 */
abstract class GameStep {

    // GameStep's 해야할 일
    init {
        configureUI()
        addEventHandlers()
    }

    // UI 만들어주기
    abstract fun configureUI()

    // 사용자의 이벤트 감지해주기
    abstract fun addEventHandlers()
}
```



1. 게임 UI 만들기
2. 유저 입력으로 게임 진행할 수 있게 하고,  
게임 오버 시 Result Game Step으로 이동

# 1. State Pattern이 필요한 이유 - 뽕요뽕요 추가하기 우아하게요!

```
/**
 * 각 게임의 단계
 */
abstract class GameStep {

    // GameStep's 해야할 일
    init {
        configureUI()
        addEventHandlers()
    }


    // UI 만들어주기
    abstract fun configureUI()

    // 사용자의 이벤트 감지해주기
    abstract fun addEventHandlers()
}
```



1. 점수 보여주는 UI 만들기
2. 뒤로가기 눌리면 Home으로 이동하기





```
/**
 * 망곰이의 리팩토링 된 과제
 */
fun main() {
    // 1. init GameSteps
    val result = Result()
    val tetris = Tetris()
    val ppuyo = Ppuyo()
    val home = Home()

    // 2. init detail
    result.setHome(home)
    ppuyo.setResult(result)
    tetris.setResult(result)
    home.setGames(listOf(ppuyo, tetris))

    // 3. start game logic!
    if (userEnter) {
        home.start()
    }
}
```



## 2. State Pattern이 뭐냐면요..

상황 별로 해야될 행동이 정해져 있을 때, 상황한테 인격 부여해서 알아하라고 시키는 패턴

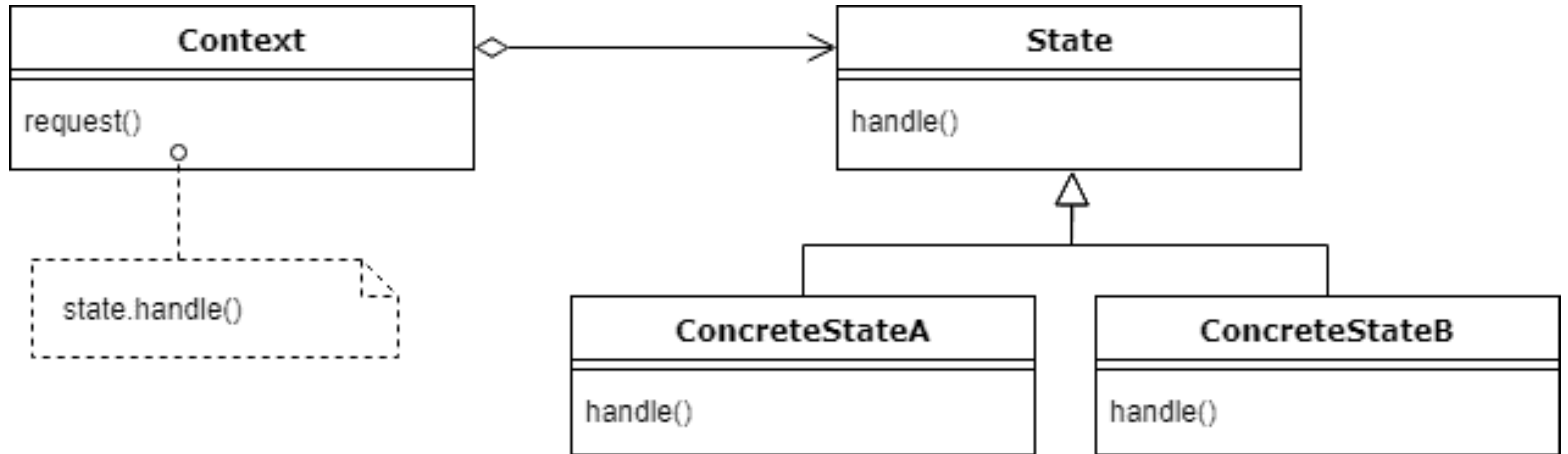
## 2. State Pattern이 뭐냐면요..

상황 별로 해야될 행동이 정해져 있을 때, 상황에 인격 부여해서 알아하라고 시키는 패턴

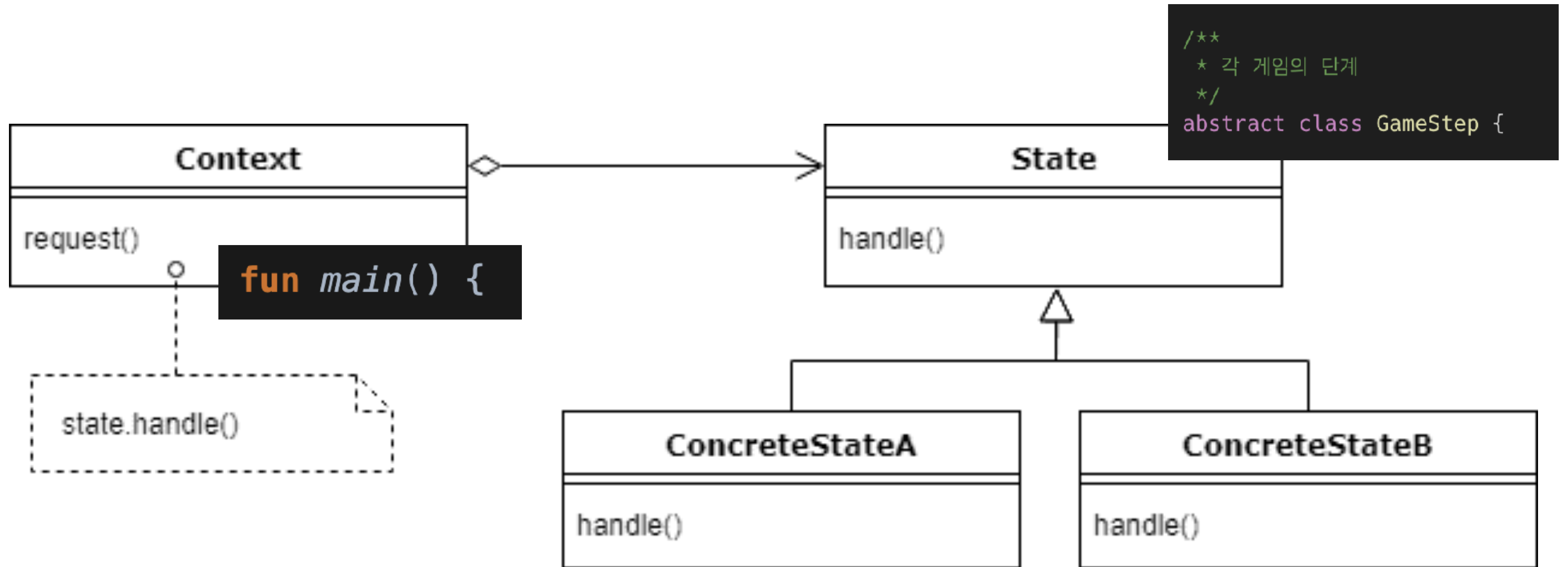
==

전체를 상태 별로 나눌 수 있을 때, 상태를 객체로 만드는 패턴

### 3. State Pattern 다이어그램 이해해보기



# 3. State Pattern 다이어그램 이해해보기



**HOME**



**Tetris!**



**Puyo!**



**Result**

# 4. State Pattern - TMI

장점 : OCP

지옥의 조건문에서 벗어남

➡ OCP를 지킬 수 있음.

➡ 코드의 가독성이 좋아짐.

# 4. State Pattern - TMI

단점

💬 오버엔지니어링은 아닐까?

💬 허접한 개발자(ME)에게 디버깅하기 쉬운 로직일까?

💬 관리해야 될 객체가 늘어난다