v 1.0-20221120

GAME PROGRAMMING PATTERNS IN JAVA - A SELECTED VIEW -

Alessandro Ricci
a.ricci@unibo.it

OBJECTIVES

- Design Patterns applied to (Basic) Game Programming
  - some patterns come from GoF
  - some other are new

ABOUT GAME DEVELOPMENT

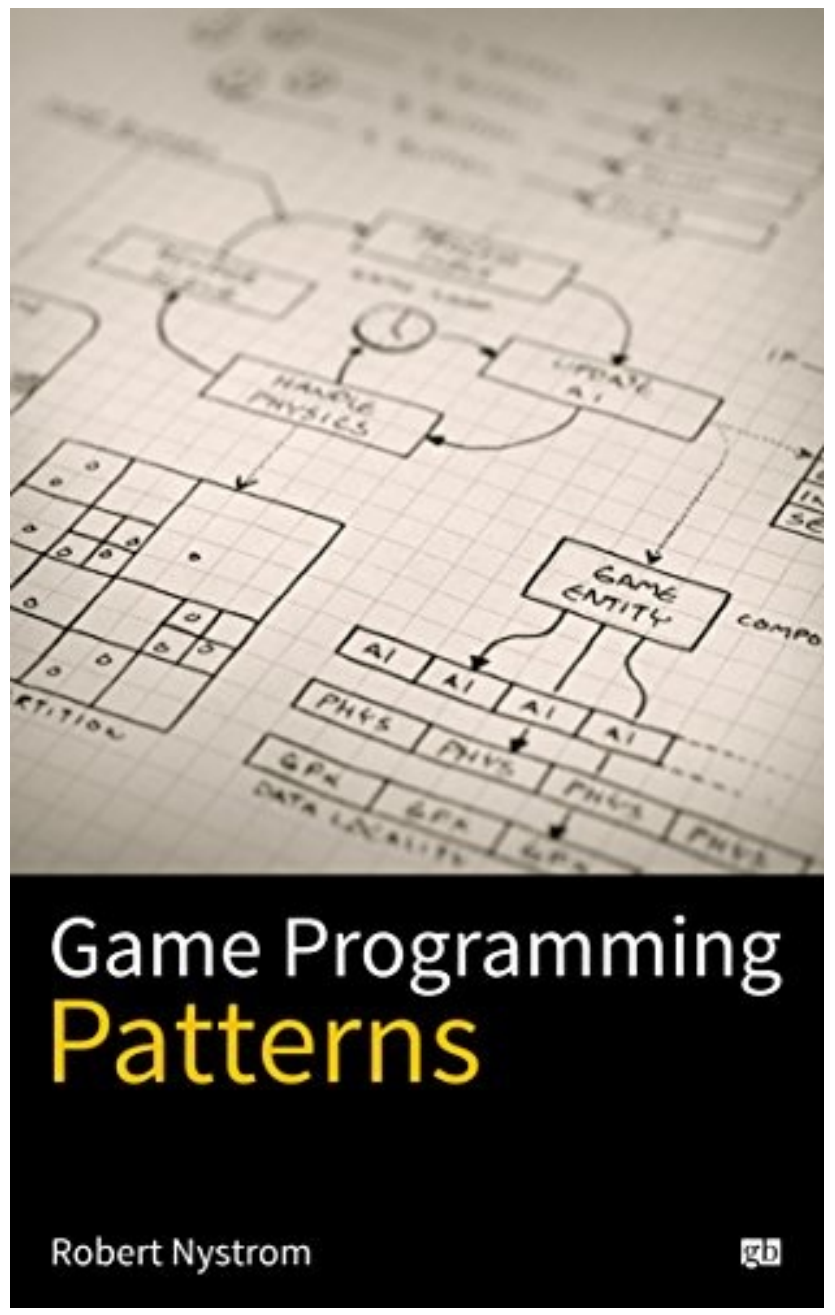
Second Edition Game Engine Architecture Jason Gregory
Game Programming All in One Third Edition
A Platform-Agnostic Approach Game Programming Algorithms and Techniques Sanjay Madhav
Game AI Pro 2 Collected Wisdom of Game AI Professionals Edited by Steve Rabin
Game Programming Gems 8 Edited by Adam Lake
Game Programming Gems
Game Coding Complete Fourth Edition
Foundations of Game Engine Development Volume 1 Mathematics Eric Lengyel
The Ultimate Guide to Video Game Writing and Design Flint Dille & John Zuur Platten
C++ for Game Programmers
Game Engine Design and Implementation Alan Thorn
Game Physics Engine Development Second Edition Ian Millington
Beginning Game Programming Fourth Edition Jonathan S. Harbour
Second Edition The Art of Game Design A Book of Lenses Jesse Schell
Architecting Networked Games Multiplayer Game Programming Joshua Glazer Sanjay Madhav
Real-Time Rendering Tomas Akenine-Möller Eric Haines Naty Hoffman
ISI LT
ROGRAMMING PATTER

GAME PROGRAMMING - GENERAL ASPECTS

- Game architecture: structure and dynamics
- Game world and object modeling - structure, behaviour
- Kinematics and dynamics - collisions, physics
- Input control & interaction with the physical environment
- Graphics, animation, audio - rendering the scene, effects
- AI
- Networking & multi-player
- …

GAME PROGRAMMING *PATTERNS* PERSPECTIVE

- Reference book
  - **[GPP] Robert Nystrom, "Game Programming Patterns", 2014**
  - …with some personal adaptations and extensions

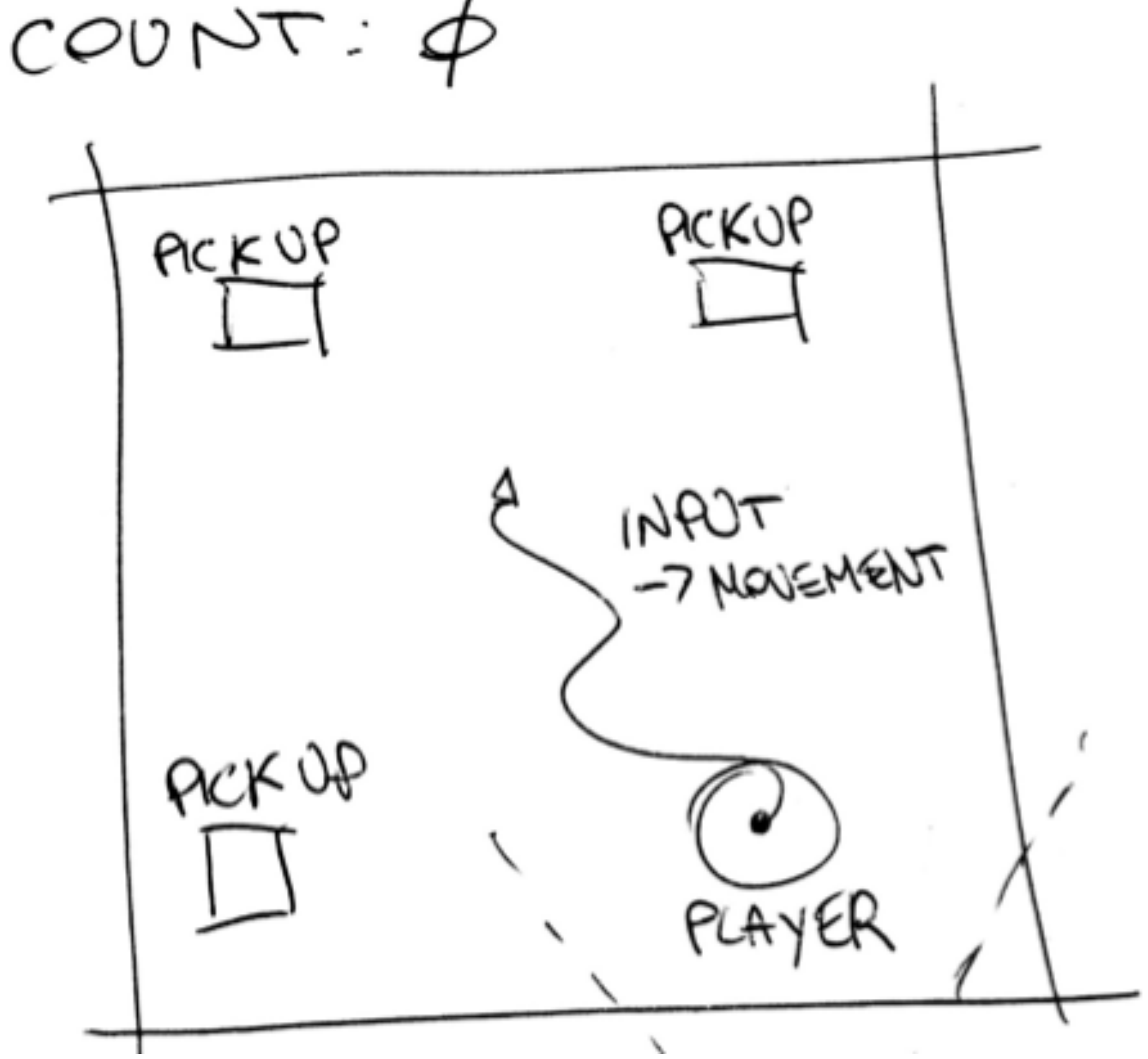


APPROACH OF THIS SEMINAR

- Programming a **simple game** as a case study (“Roll a ball” game)
  - .. in a sequence of **steps** ..
  - .. introducing incrementally a set of **patterns** ..
  - .. adopting a “**game engine**”-like framework perspective

CASE STUDY: “ROLL A BALL”

- A *ball* rolling through a 2D environment, moved by the player
- Objective:
  - collect the *pick-up objects* as fast as possible, avoiding borders



“GAME ENGINE” PERSPECTIVE

- Game engines
  - *Frameworks* factorising functionalities/services for developing and running games
    - isolating the game from the HW on which it is running
    - enforcing a separation of concerns about game aspects
    - *framework* perspective vs. libraries
      - managing the control flow
- Many examples
  - http://en.wikipedia.org/wiki/List_of_game_engines

PATTERNS

- **Game Loop**
- **Command**
- **Observer**
- **Event Queue**
- **Component**
- **State Pattern**

STEP #01
DEFINING THE CONTROL ARCHITECTURE
THE **GAME LOOP**

THE GAME LOOP

- Control loop ruling the execution of a game
  - *"Decouple the progression of game time from user input and processor speed"* [GPP, CH 9]
  - https://java-design-patterns.com/patterns/game-loop/
- Evolution ~ sequence of frames

```
while (true) {
  processInput()
  update()
  render()
}
```

SYNCH WITH REAL TIME

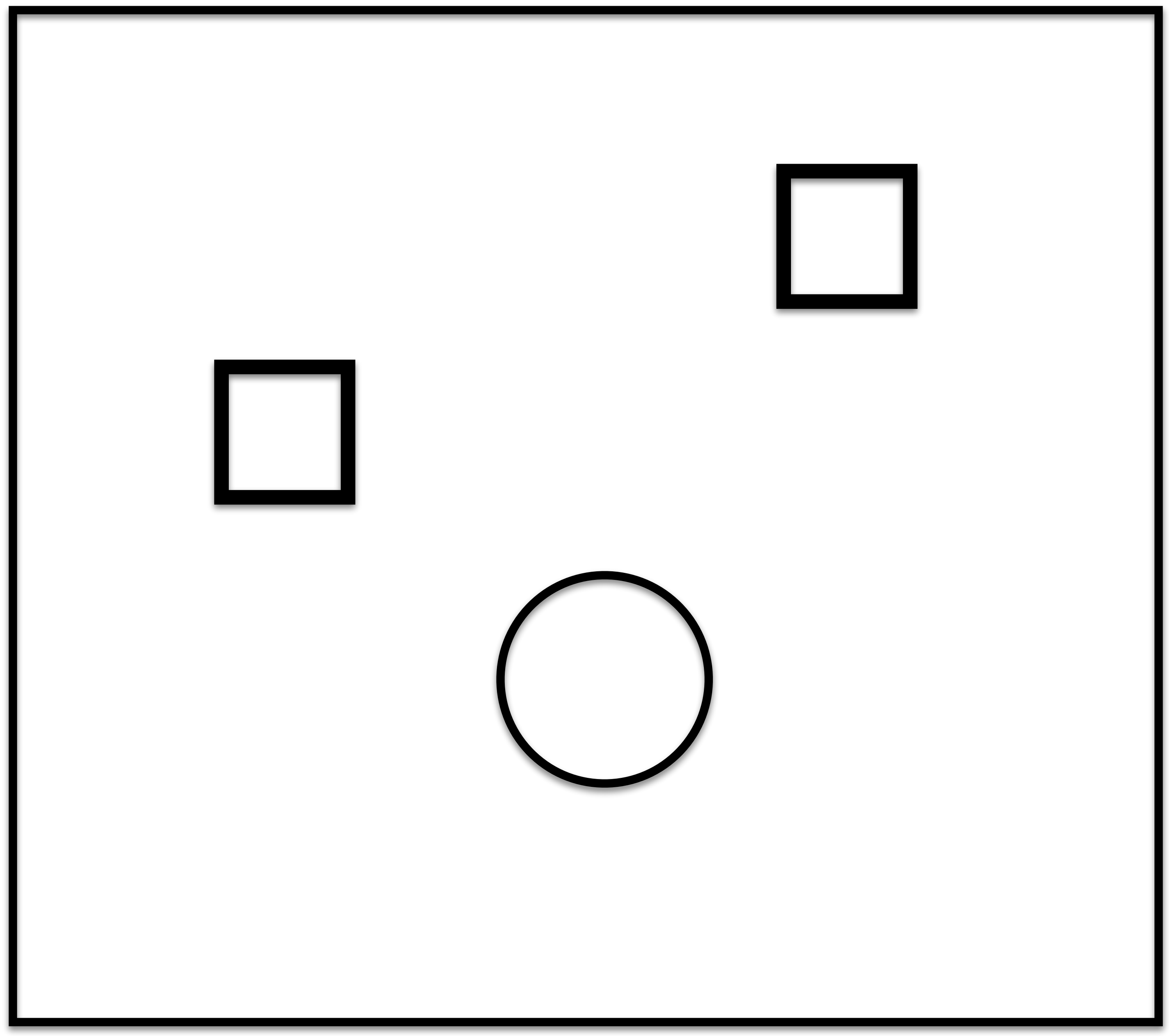
- Choosing a time step to advance the game depending on how much real time elapsed since the previous frame
    - the game runs at the same rate in spite of the specific HW…
    - players with more powerful HW gains in gameplay smoothness
- Go to the code: Step #01

```
double lastTime = getCurrentTime()
while (true) {
  double current = getCurrentTime()
  double elapsed = current - lastTime
  processInput()
  update(elapsed)
  render()
  lastTime = current
}
```

STEP #02:
START MODELLING THE GAME
(~**DOMAIN MODEL PATTERN**)

START MODELLING THE GAME

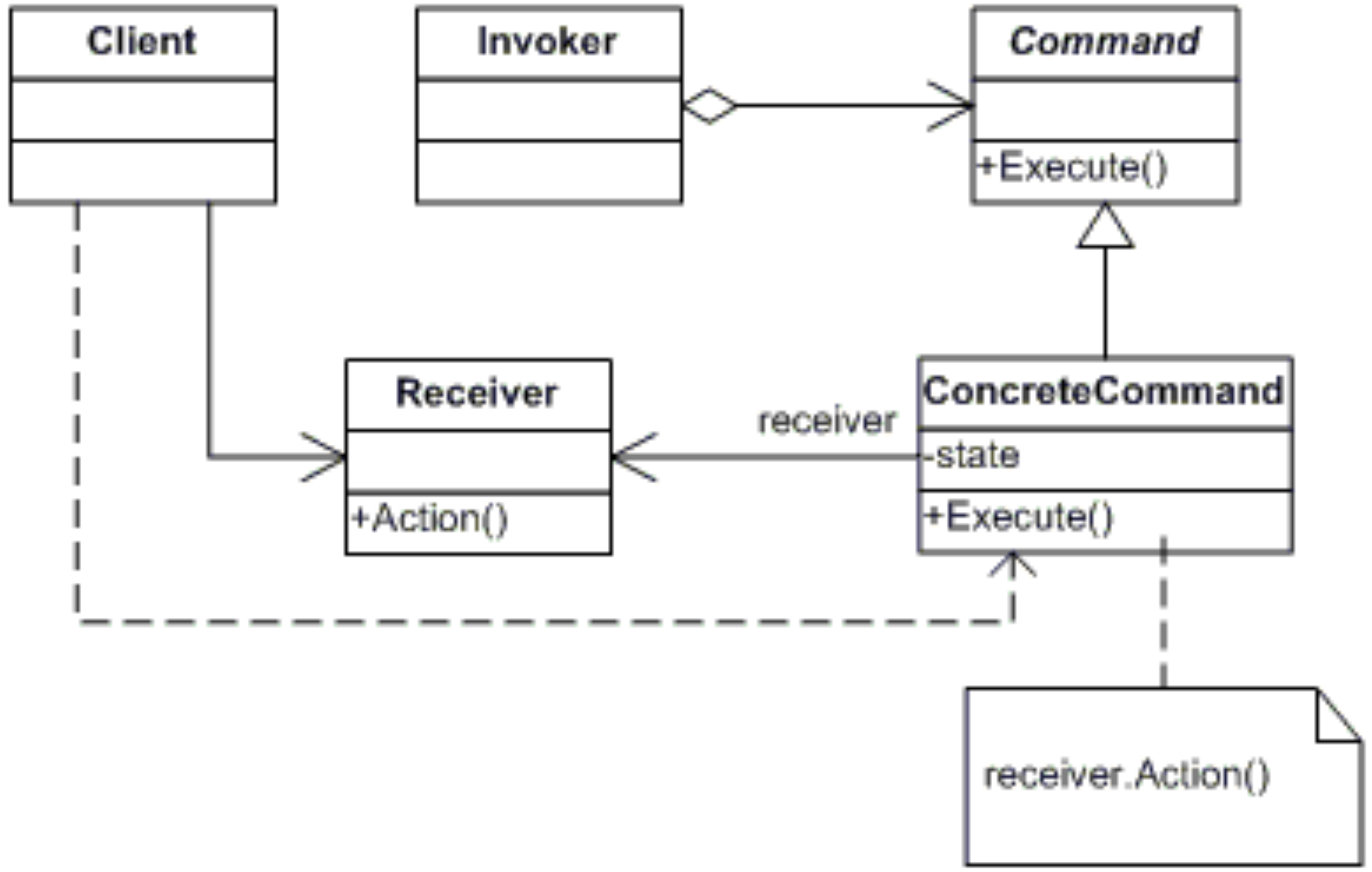
- OOP modeling of Roll-a-Ball game
  - **GameObject**
    - Ball, PickUpObj
  - **World**
- Basic *separation of concerns*
  - model/graphics
- Recalling the domain model pattern
  - https://java-design-patterns.com/patterns/domain-model/
- Go to the code: Step #02



STEP #03:
HANDLING INPUT WITH THE **COMMAND** PATTERN

INPUT PROCESSING

- **Command** pattern
  - *"Encapsulate a request as an object, thereby letting users parametrize clients with different requests, queue or log requests…"* [GPP, CH2]
  - https://java-design-patterns.com/patterns/command/
- Async processing - event listening
  - async keyboard events => commands
- Go to the code: Step #03



STEP #04
HANDLING COLLISIONS

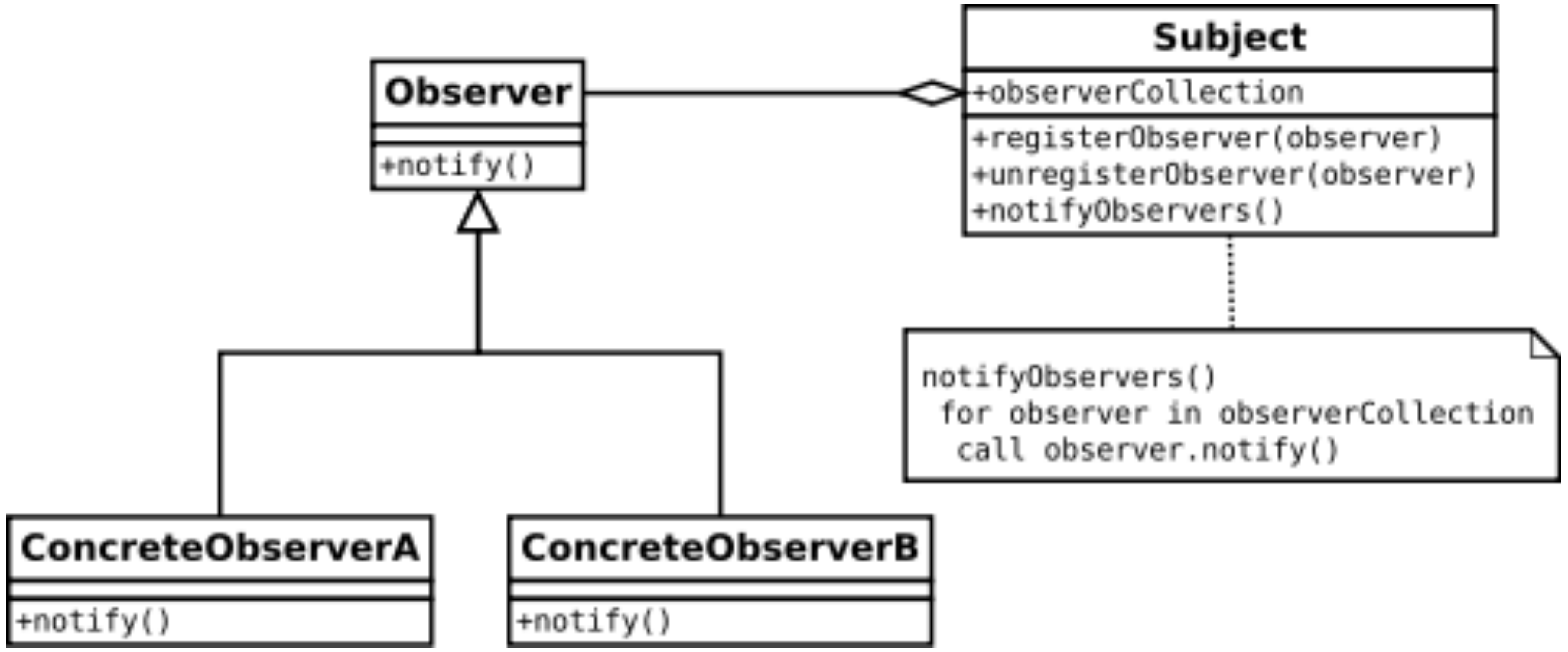
A LITTLE BIT OF PHYSICS: COLLISIONS

- Simple collision management
    - Ball + PickUpObj, Ball + borders
    - meet Bounding Boxes
- Go to the code: Step #04

STEP #05
MODELING & HANDLING EVENTS
WITH THE **OBSERVER PATTERN**

EVENTS & OBSERVER PATTERN

- Introducing the game state, that includes a score, and related *events*
  - score +1 for PickUp, -1 for borders
- Decoupling collision management from the effects
  - **Observer** pattern
    - https://java-design-patterns.com/patterns/observer/
    - ***async* variant**
- Go to the code: Step #05

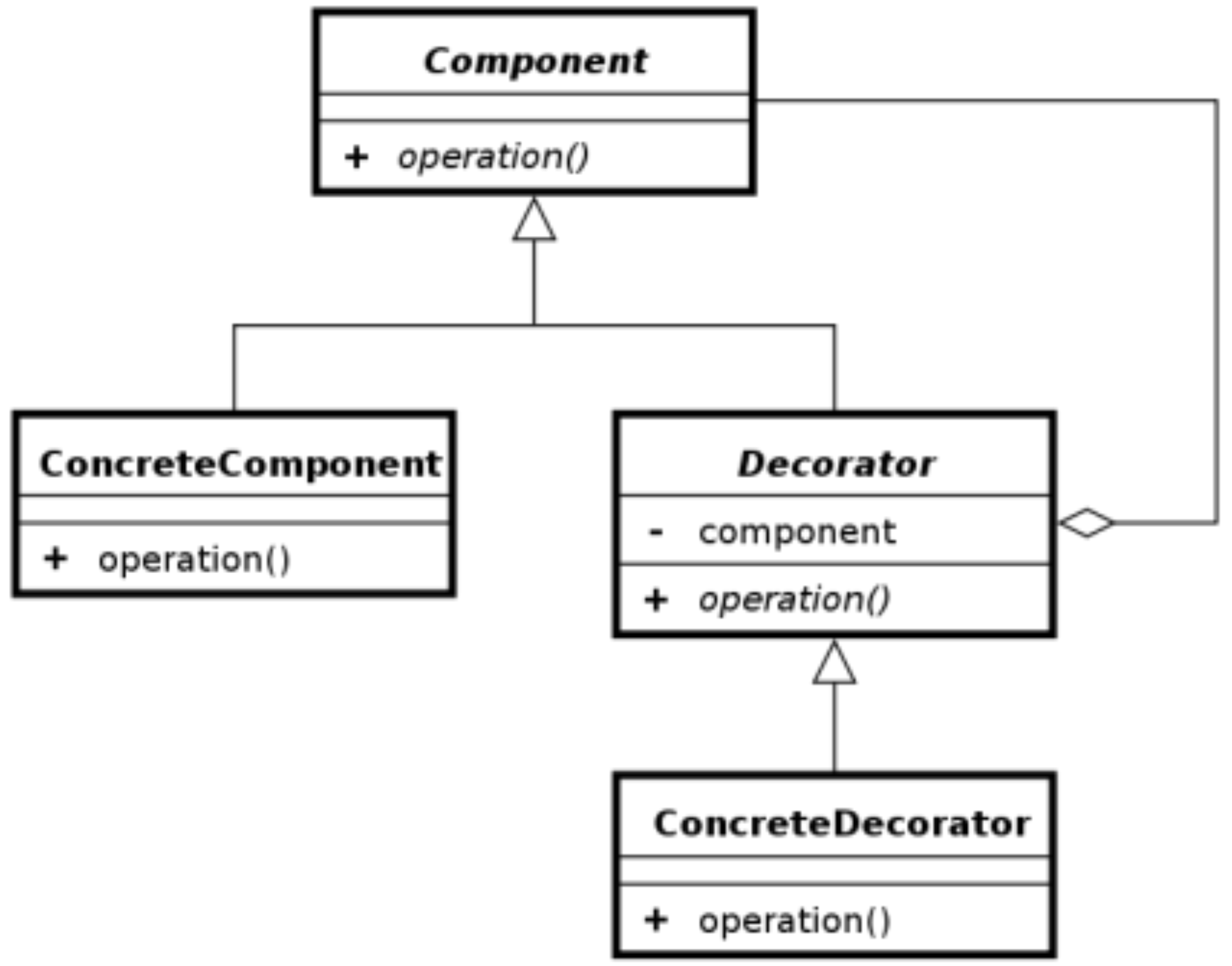


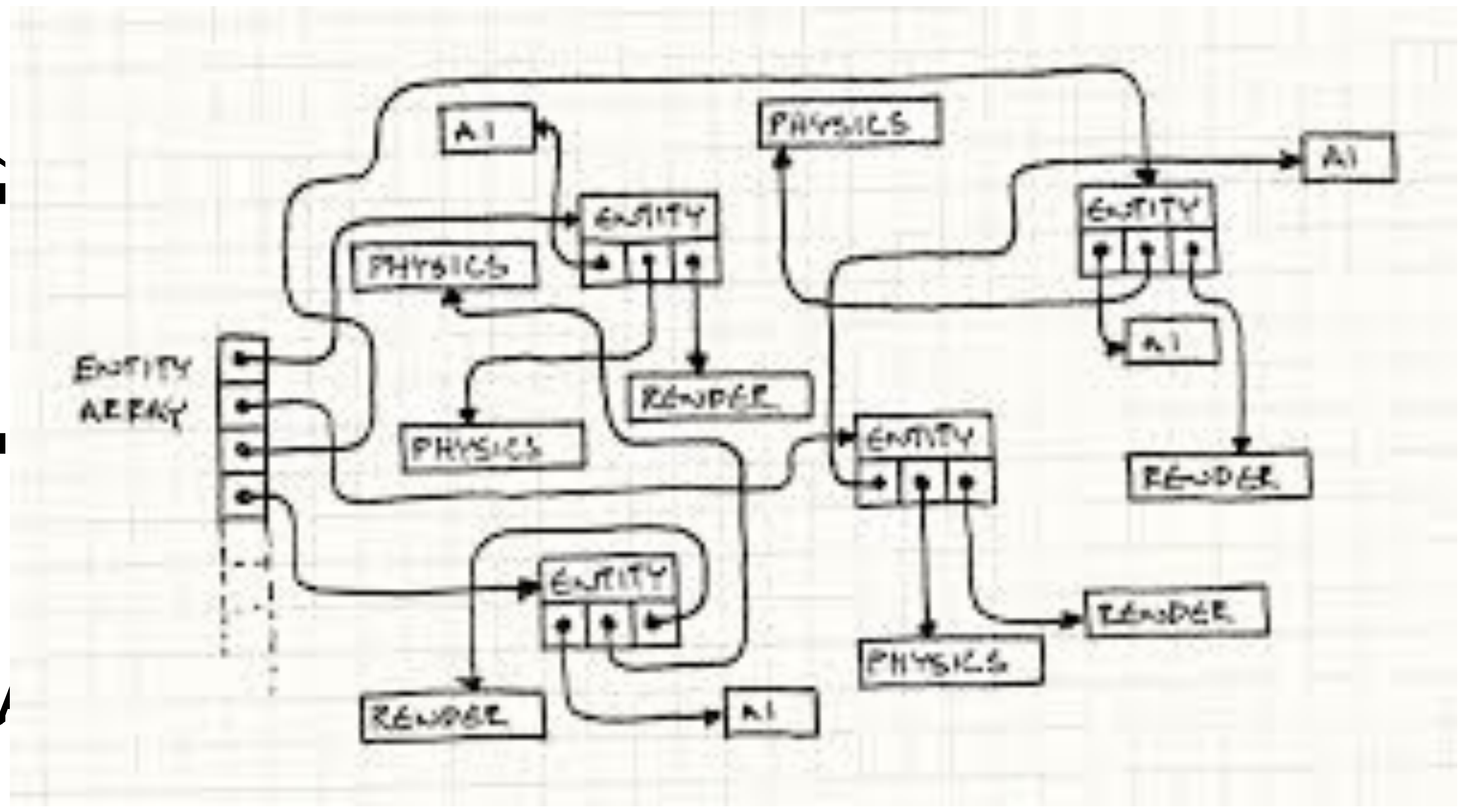
```
notifyObservers()
 for observer in observerCollection
  call observer.notify()
```

STEP #06
REFACTORING FOR FLEXIBLE EXTENSIBILITY WITH THE **COMPONENT** PATTERN

COMPONENT PATTERN

- **Component** pattern
  - *"Allow a single entity to span multiple domains without coupling the domains each other"* [GPP, CH 14]
- Major refactoring
  - full separation of concerns based on domains (physics, graphics, input)
  - pure Game Objects + Game Object Factory
- Go to the code: Step #06





(from [GPP])

STEP #07
GAME OVER

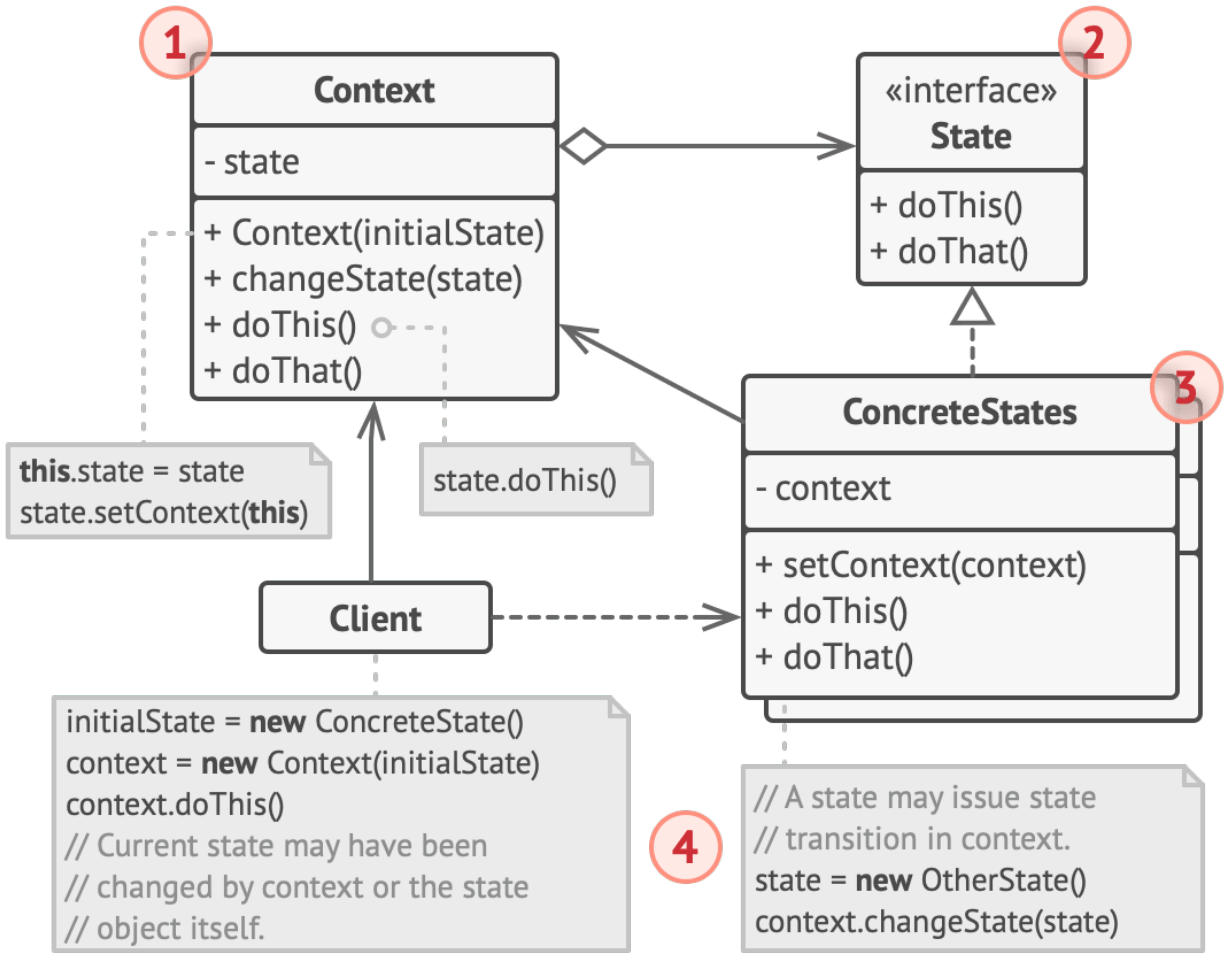
GAME OVER

- Back to the game objective: conditions to end the loop
  - based on GameState
- Game state based rendering
- Go to the code: Step #07

STEP #08
DESCRIBING AUTONOMOUS BEHAVIOURS WITH THE **STATE PATTERN**

STATE PATTERN FOR DEFINING AUTONOMOUS BEHAVIOURS (BOTS)

- Introducing a first dumb AI player *(MosquitoAIInputComponent)* controlling the ball, random movement
- Behaviour structured into simple finite state machine based on the **state pattern**
  - https://java-design-patterns.com/patterns/state/
- Go to the code: Step #08



```
initialState = new ConcreteState()
context = new Context(initialState)
context.doThis()
// Current state may have been
// changed by context or the state
// object itself.
```

```
// A state may issue state
// transition in context.
state = new OtherState()
context.changeState(state)
```

https://refactoring.guru/design-patterns/state

OTHER STEPS…

- **Step #09 — Refining physics with accelerations**
    - keyboard-based player*(PlayerInputComponentWithAcc)* that produces movements with accelerations
- **Step #10 — Managing multiple balls**
    - extending the world model with multiple balls, each one with its own component-based configuration
- **Step #11 — Managing multiple players**
    - adding the player as first-class game concept — encapsulating a score and a specific input control strategy