327: Object-oriented programming

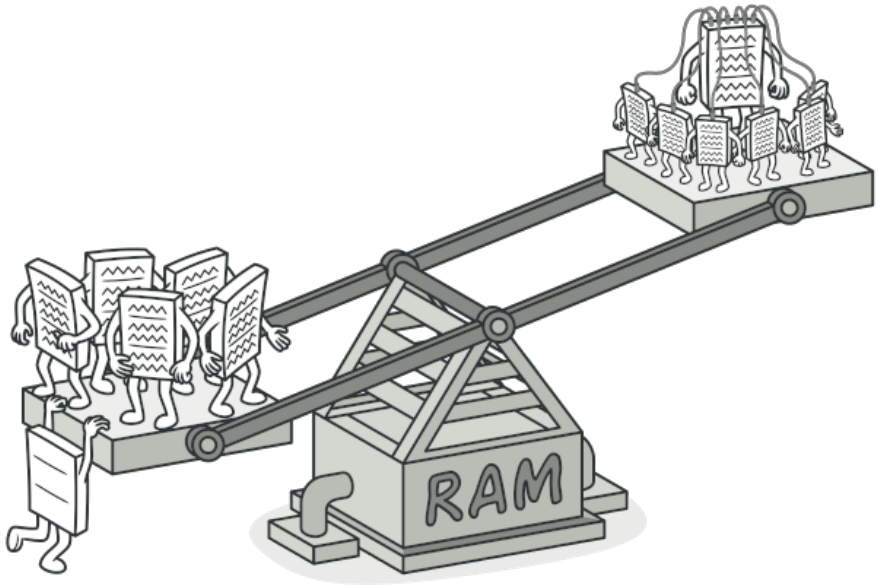
Lecture 21

11/17/2021

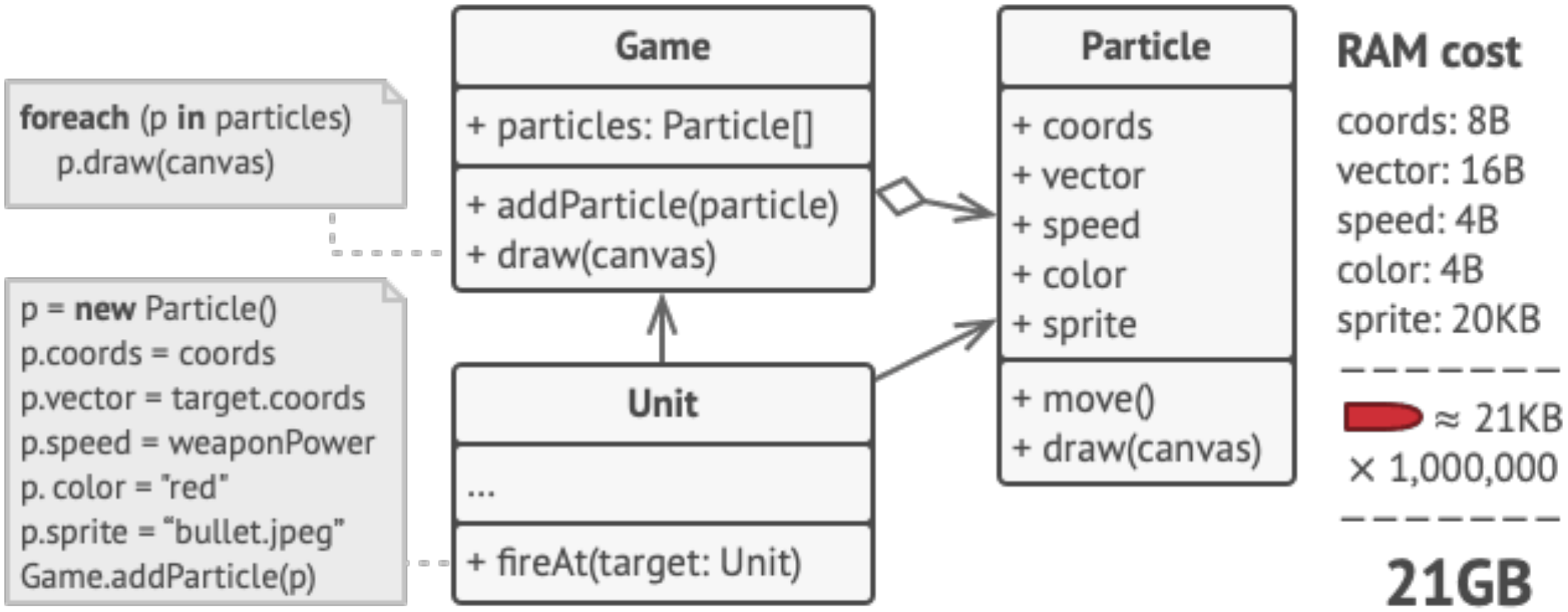
Professor Barron

Flyweight pattern

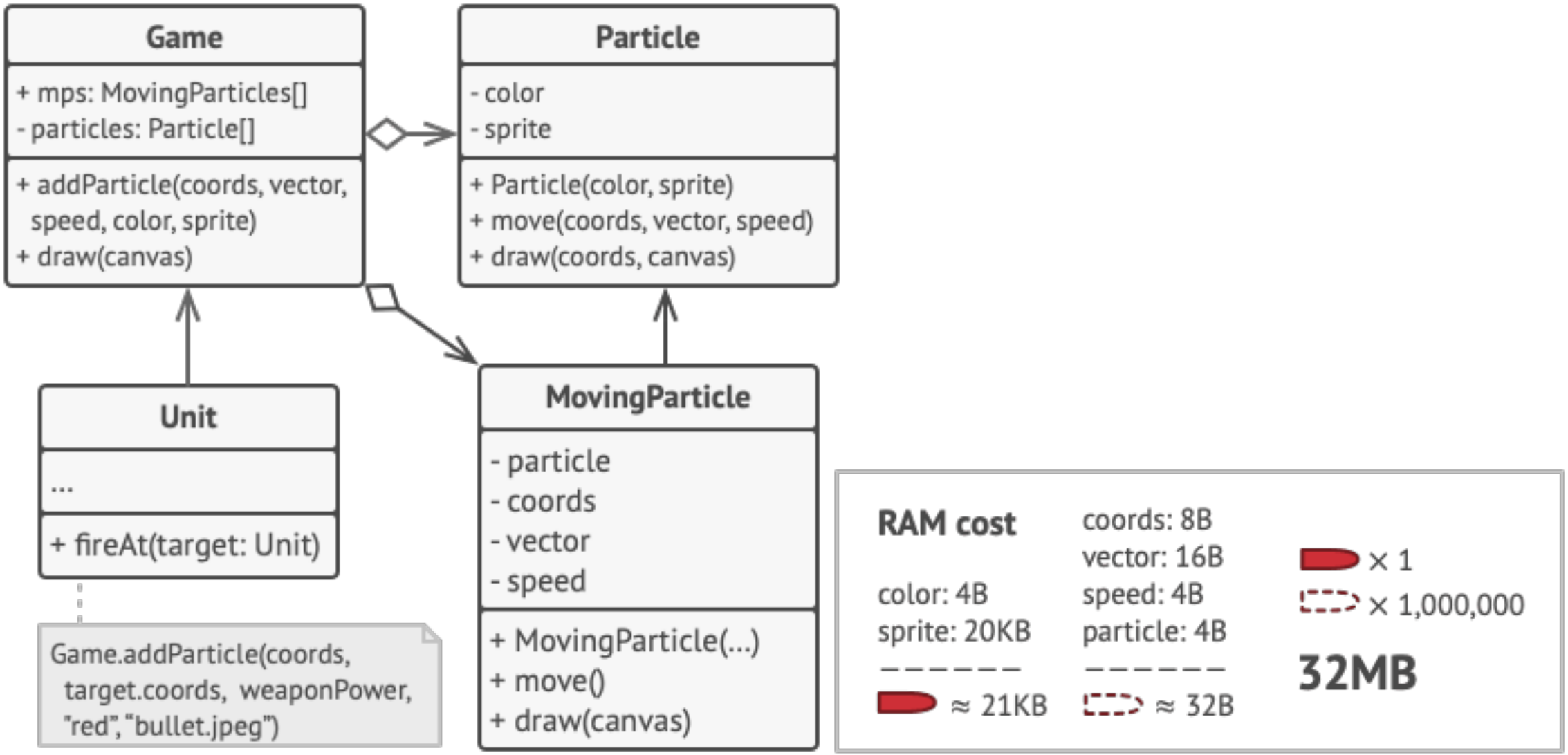
- Structural pattern
- making hundreds/thousands of separate objects from a class is very flexible (each can be changed individually)
- this uses lots of space for a lot of identical data
- flyweight saves memory by keeping shared state in a single object
- intrinsic state
  - independent from context
  - shared in flyweight object
- extrinsic state
  - specific context
  - passed to the flyweight when calling methods



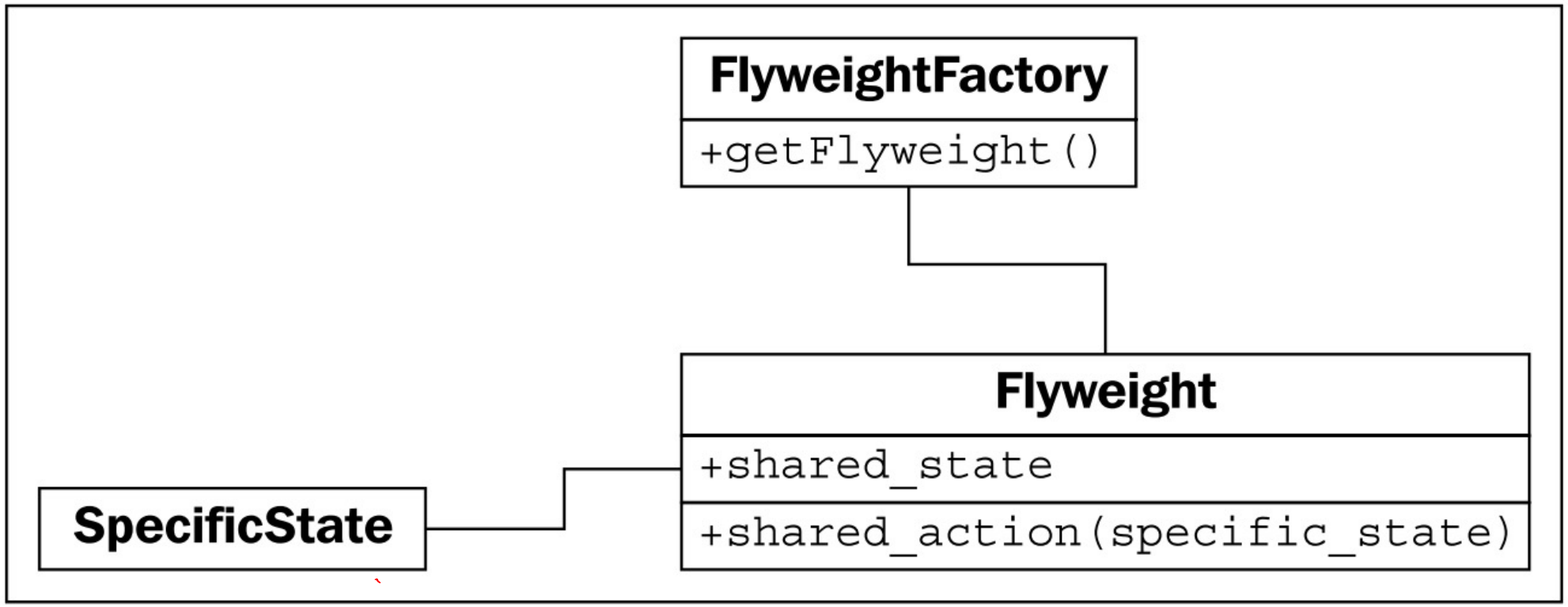
Flyweight example



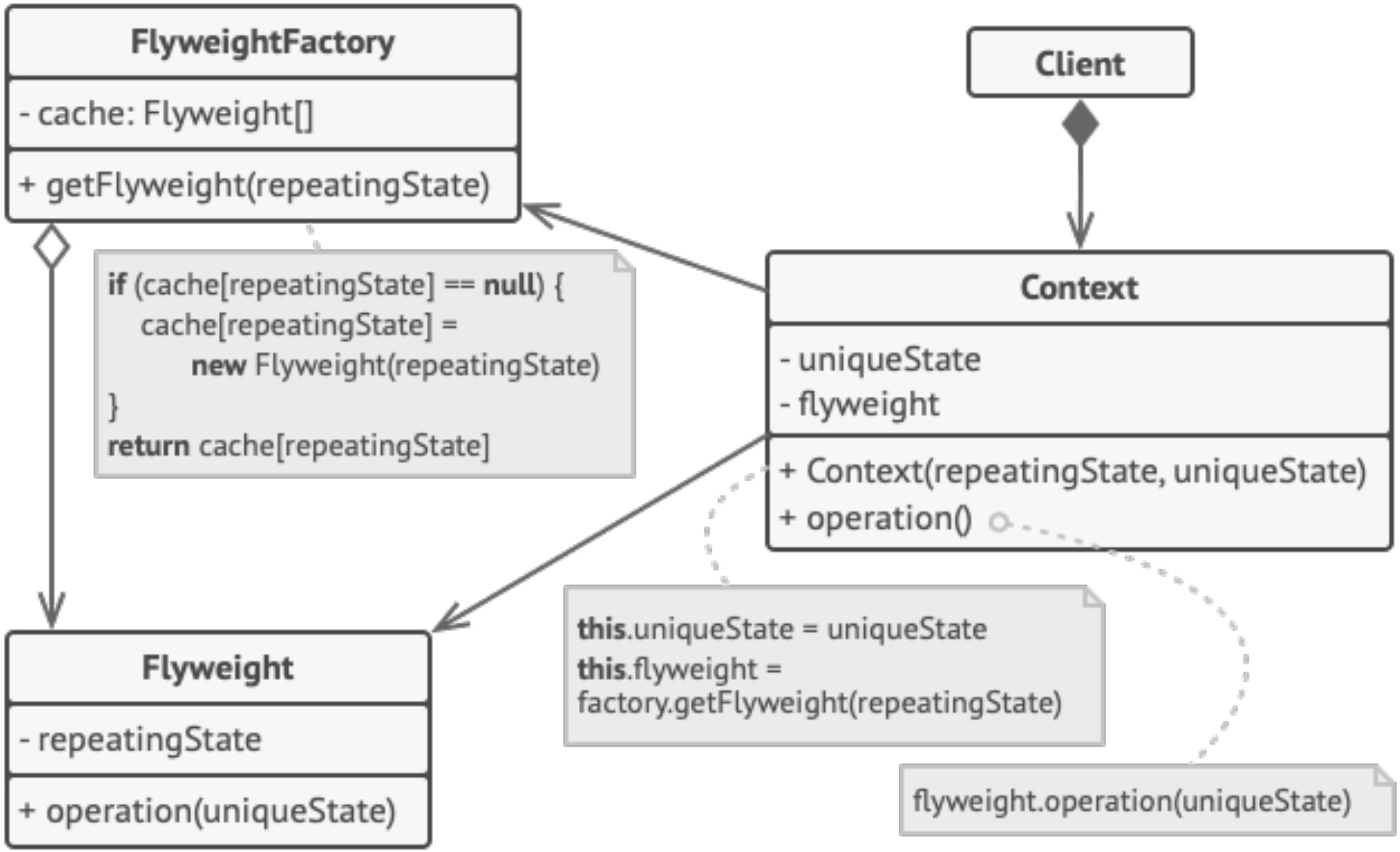
Flyweight example



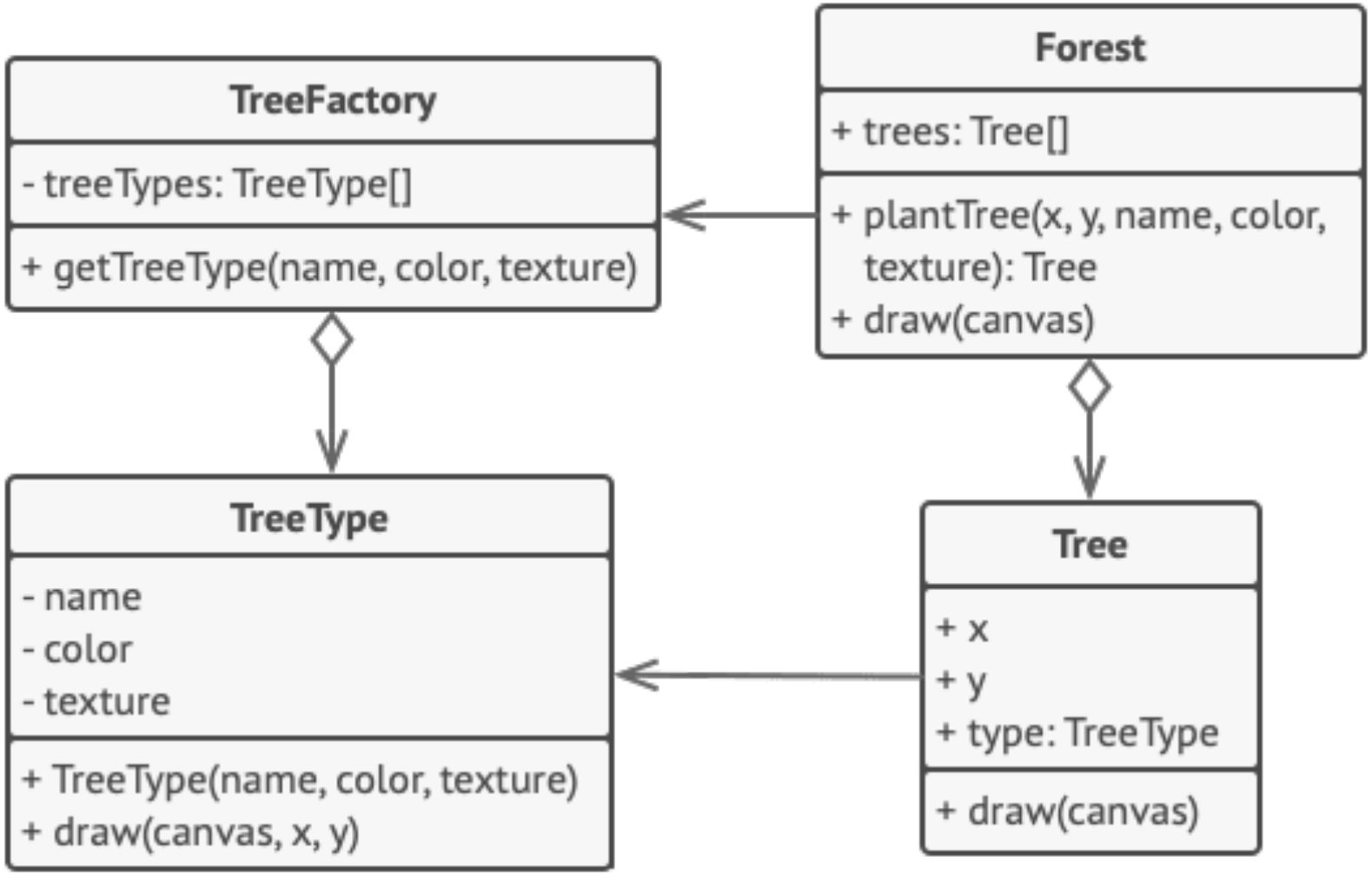
Class diagram



Class diagram



Example


TreeFactory
- treeTypes: TreeType[]
+ getTreeType(name, color, texture)
Forest
+ trees: Tree[]
+ plantTree(x, y, name, color, texture): Tree
+ draw(canvas)
TreeType
- name
- color
- texture
+ TreeType(name, color, texture)
+ draw(canvas, x, y)
Tree
+ x
+ y
+ type: TreeType
+ draw(canvas)

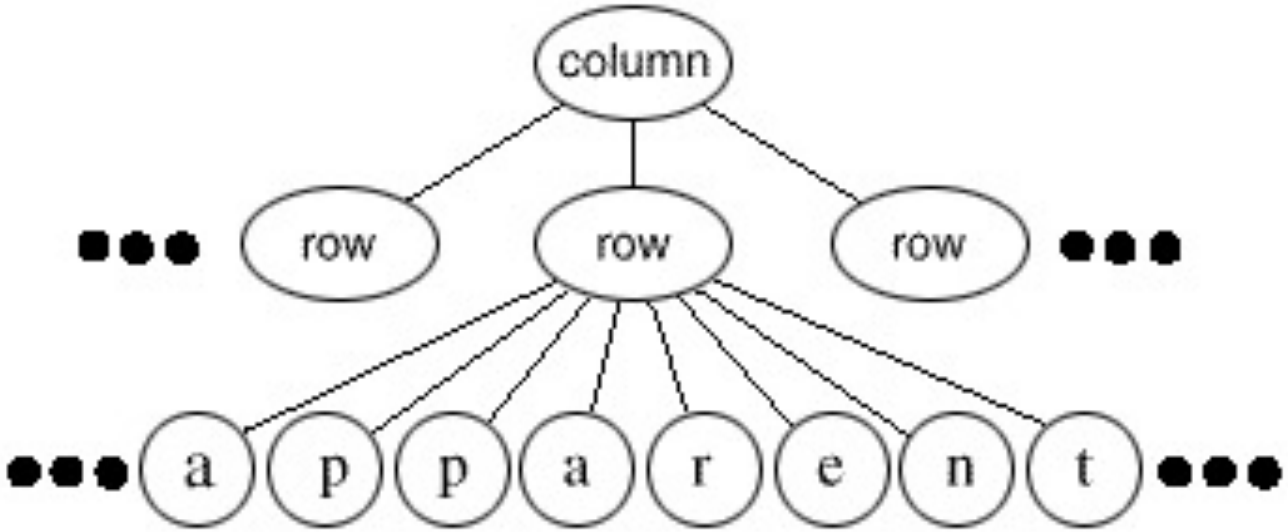
Example

- Images loaded in browser
- only needs to fetch the image once
- its position is extrinsic state, but the image data is intrinsic

Browser loads images just once and then reuses them from pool:

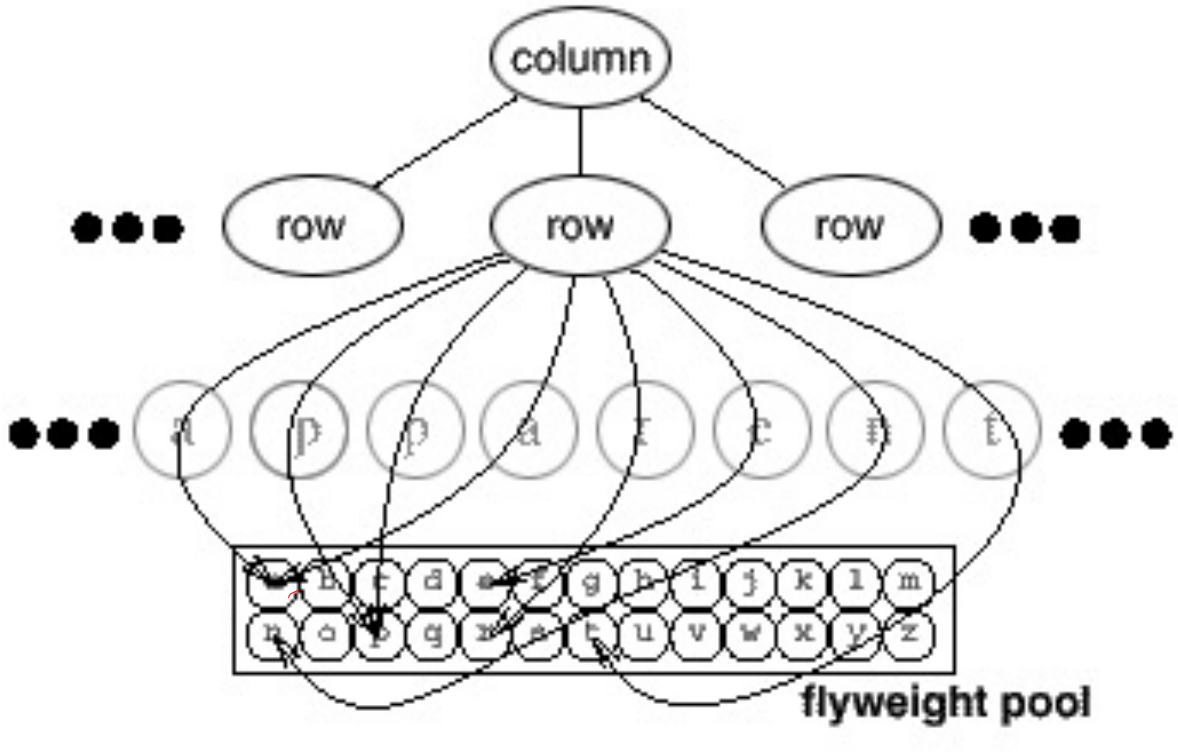
Example

- characters in a word processor…
- a character can have a position, font, color, size…
- should every character be its own object?



Example

- each character can be a flyweight object and we pass in, the necessary context to draw it in the window



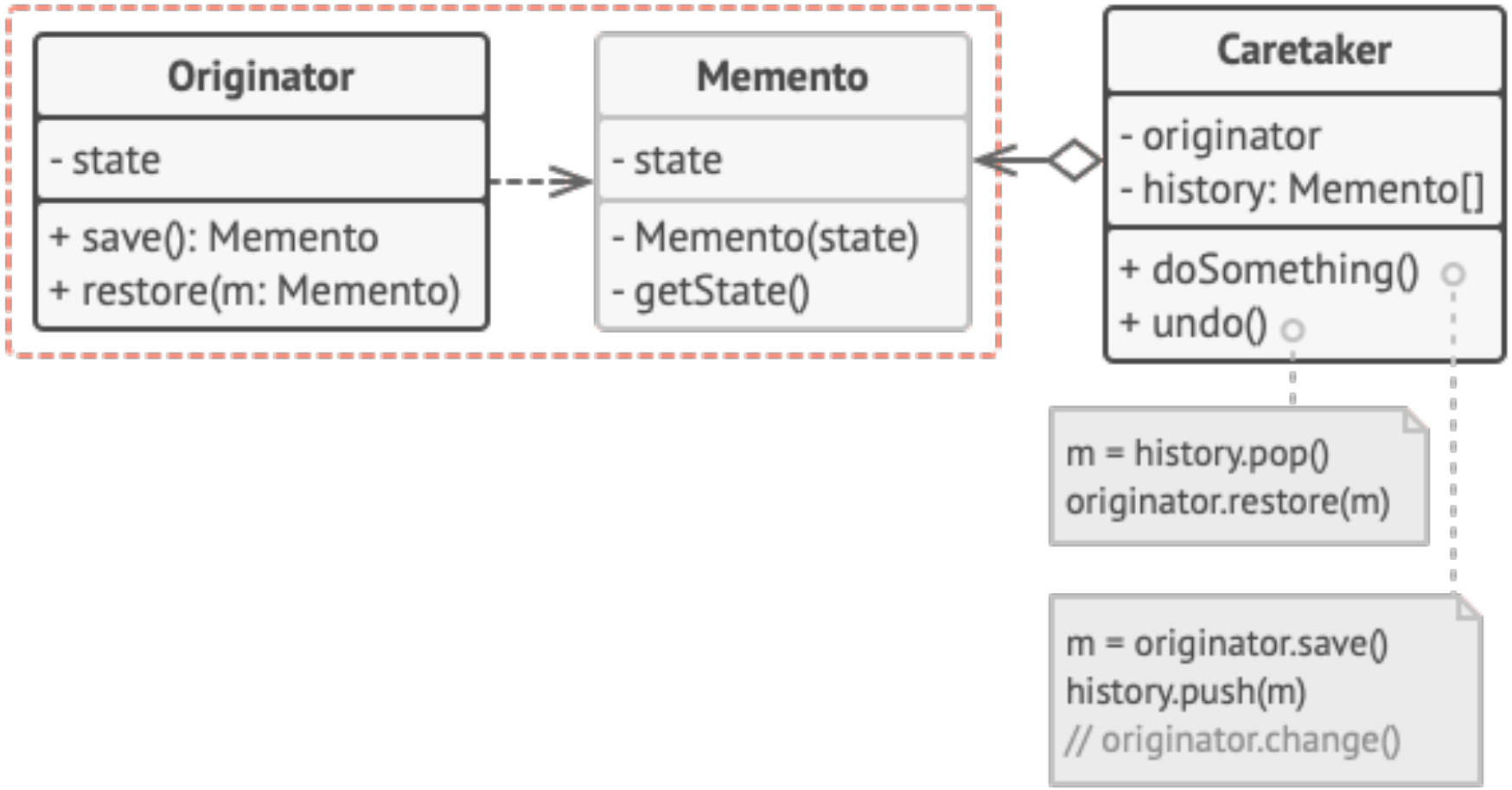
Flyweight discussion

- meant for memory *optimization*
- reminiscent of some data compression algorithms
- good in certain scenarios, but overkill for others
- increases complexity
- specific context is managed by the clients which could be easy or hard depending on the application

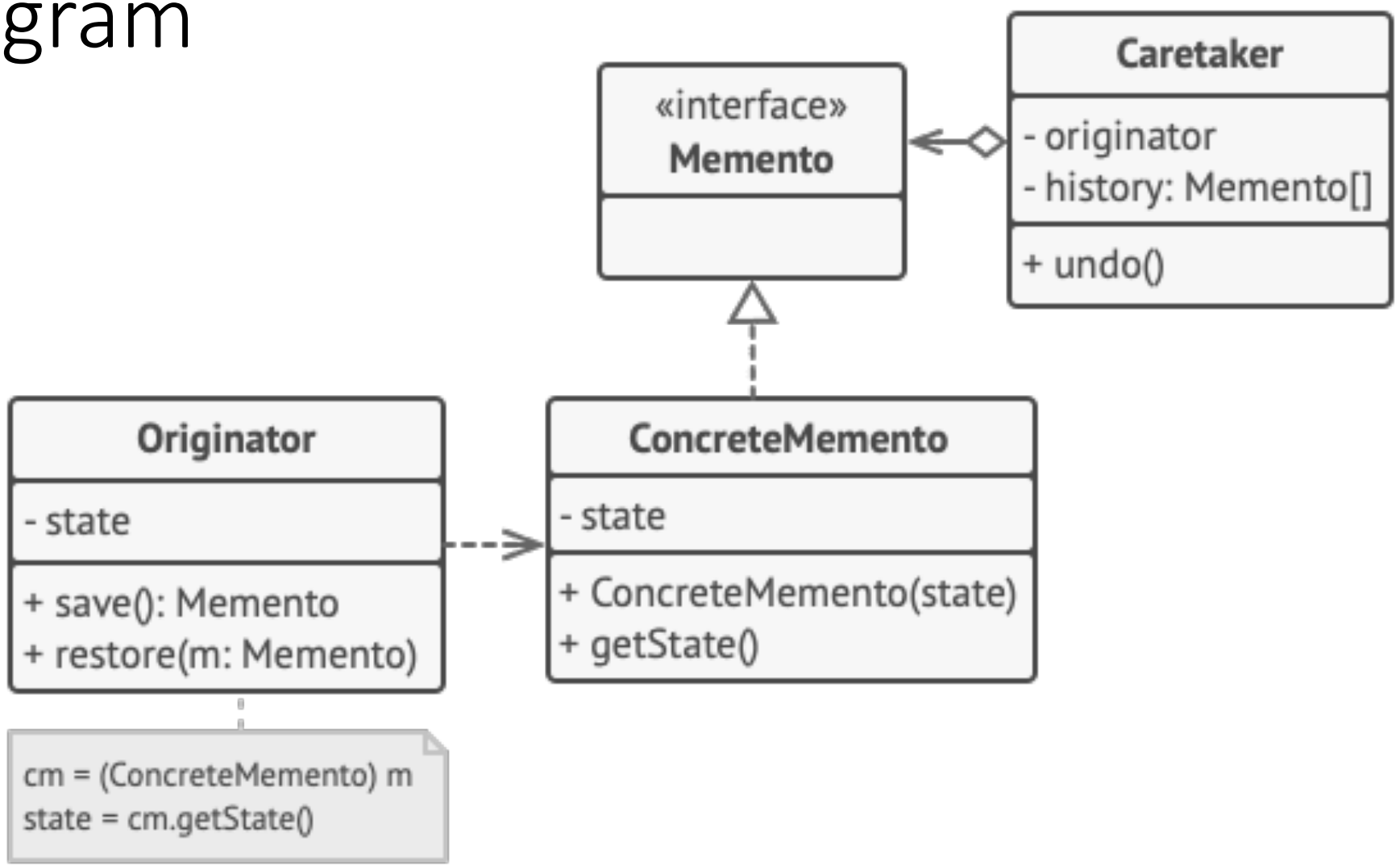
Memento pattern

- Behavioral pattern
- Save the state of an object in a snapshot
- Can restore a previous state from a snapshot (undo)
- Avoids exposing the details of the original class
- Lifetime of snapshots should be managed by caretaker code
- Can be used alongside the command pattern

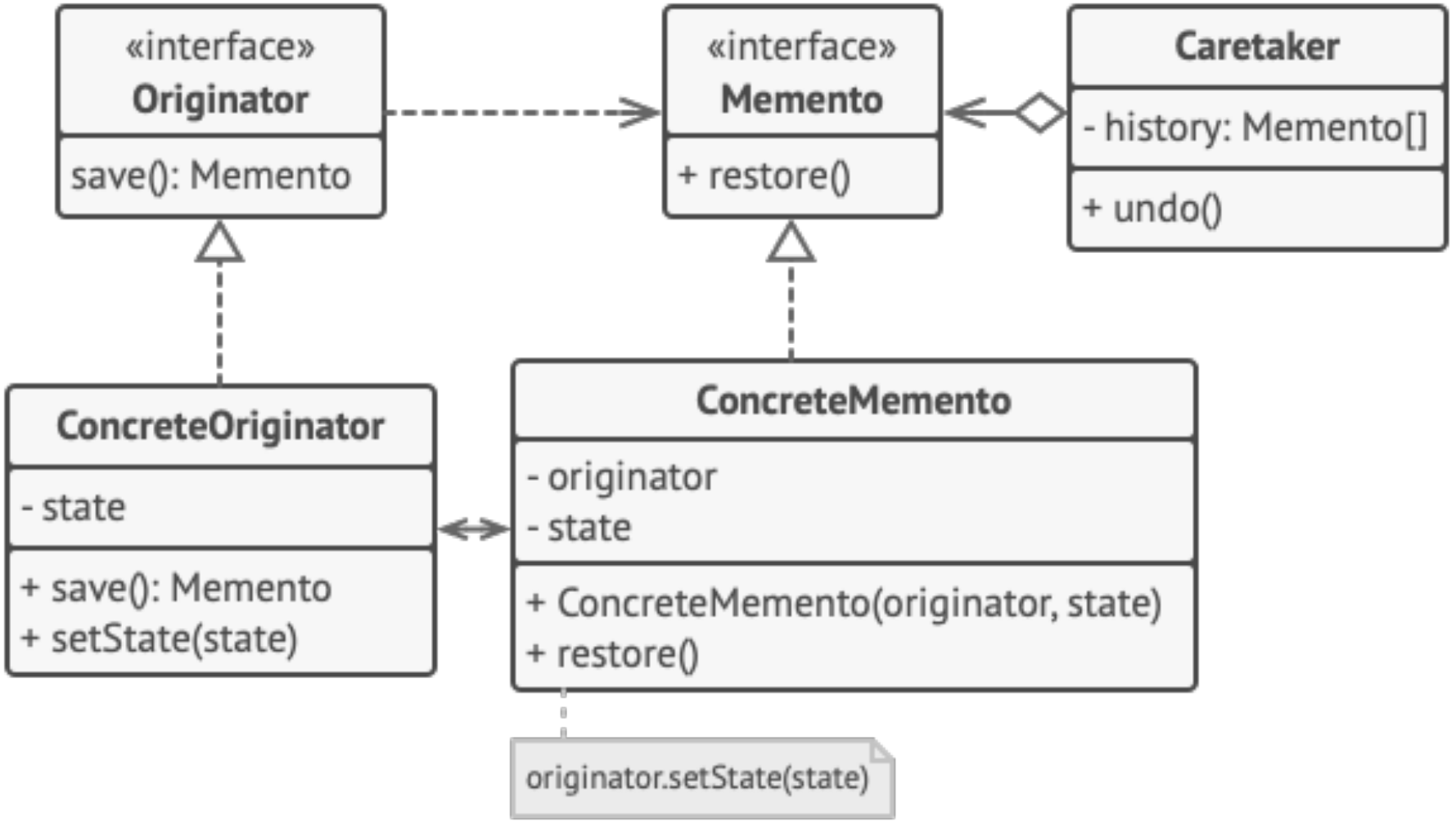
Class diagram



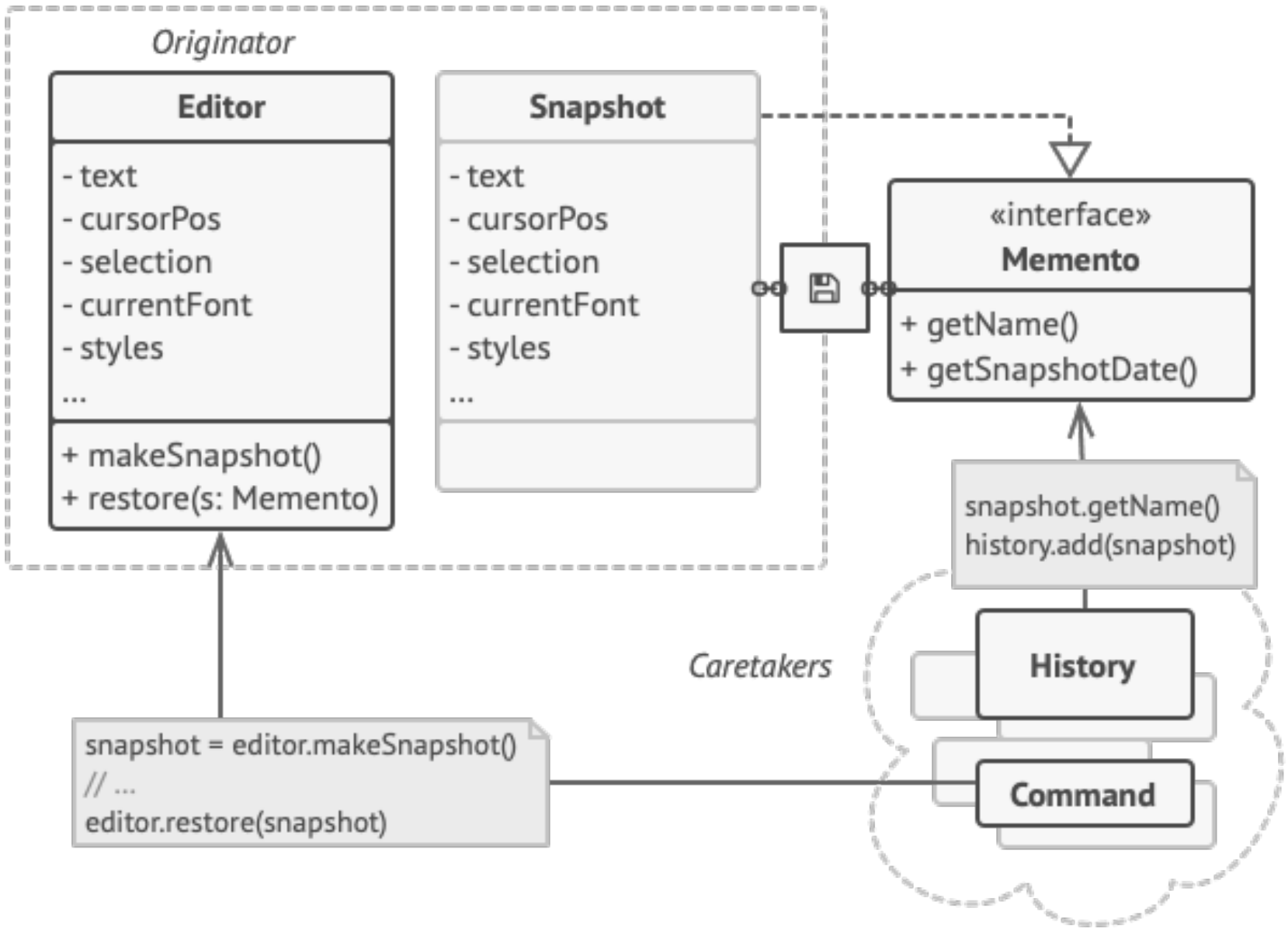
Class diagram



Class diagram



Example


Originator
Editor
- text
- cursorPos
- selection
- currentFont
- styles
...
+ makeSnapshot()
+ restore(s: Memento)
Snapshot
- text
- cursorPos
- selection
- currentFont
- styles
...
«interface»
Memento
+ getName()
+ getSnapshotDate()
snapshot.getName()
history.add(snapshot)
Caretakers
History
Command
snapshot = editor.makeSnapshot()
// ...
editor.restore(snapshot)

Concurrent programming

- Process
  - Everything needed to run
  - process id
  - virtual address space
  - code
  - handles to open files (and other resources)
  - security context
  - environment variables
  - at least one main thread

Concurrent programming

- Thread
  - Scheduled for execution on a core of the CPU (by the OS)
  - shared virtual address space
    - with other threads in the same process
  - execution context
    - program counter
    - machine registers
    - stack