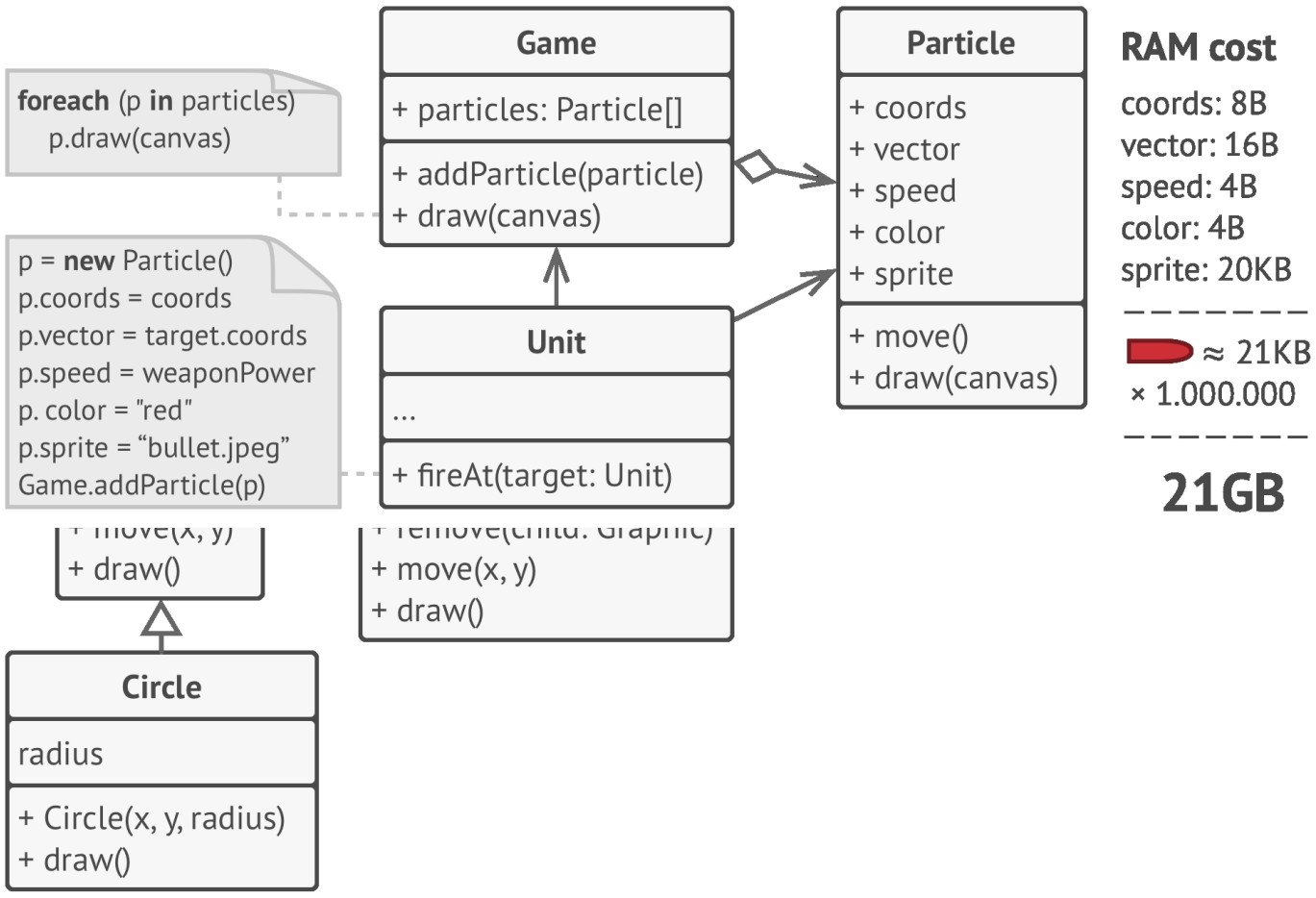
Flyweight

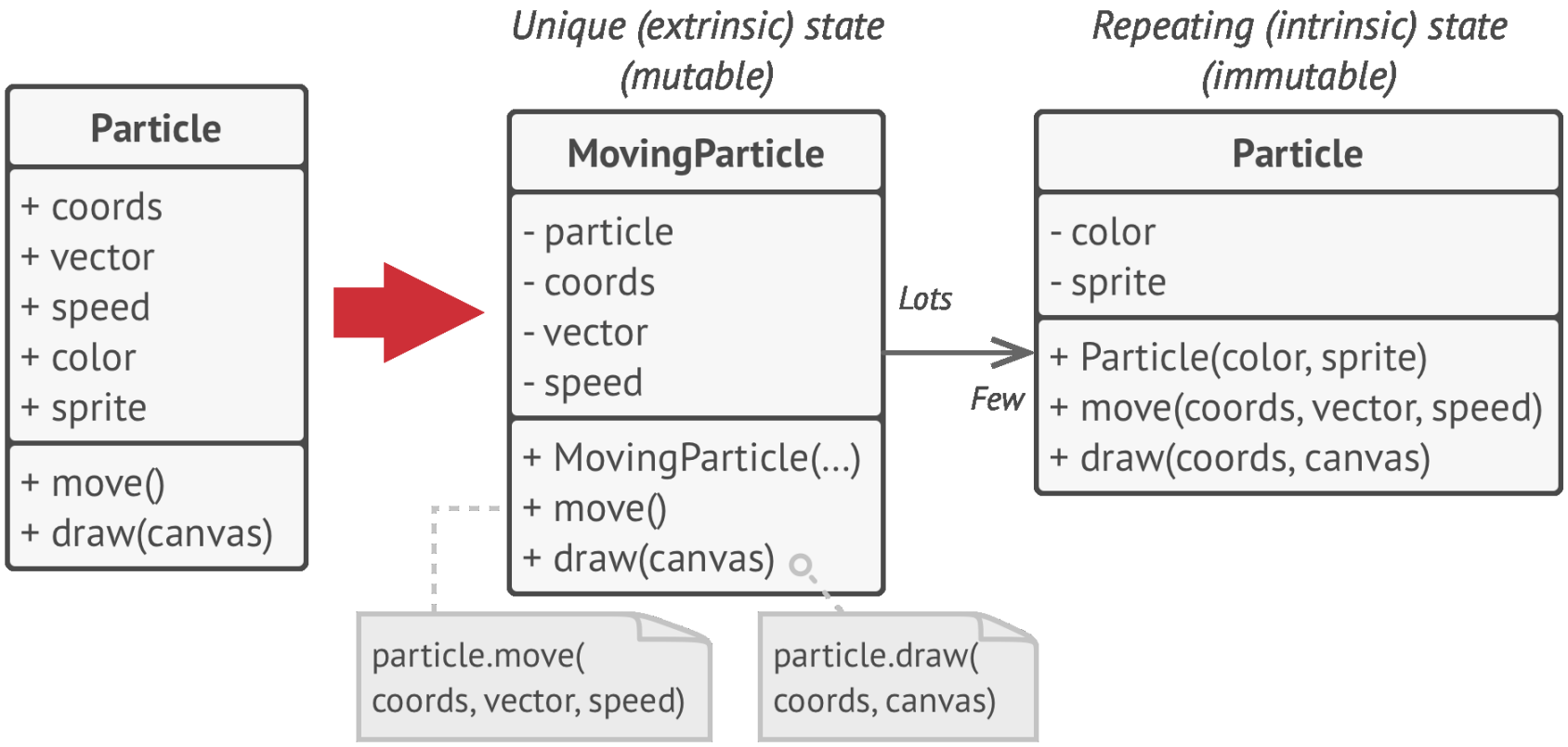
Also known as: Cache

Flyweight is a structural design pattern which allows to fit more objects into the available amount of RAM by sharing common parts of state between multiple objects instead of keeping all of the data in each object.

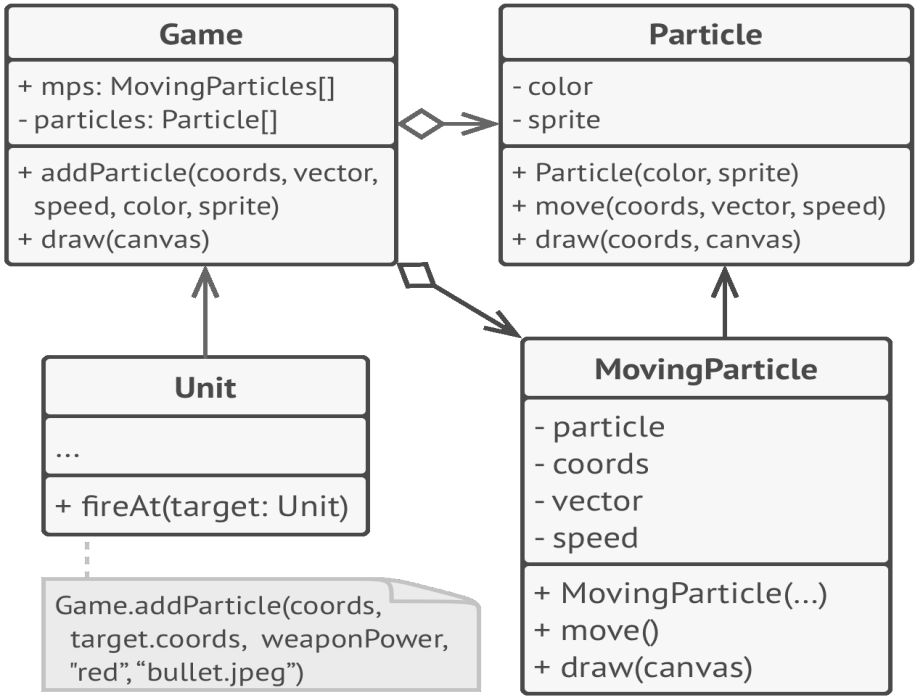
Problem

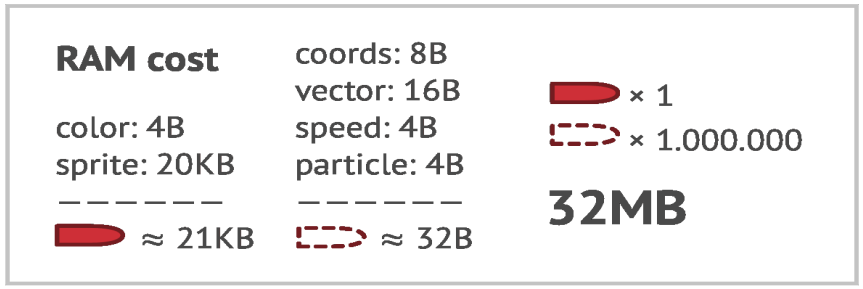

foreach (p in particles)
p.draw(canvas)
p = new Particle()
p.coords = coords
p.vector = target.coords
p.speed = weaponPower
p. color = "red"
p.sprite = “bullet.jpeg”
Game.addParticle(p)
Game
+ particles: Particle[]
+ addParticle(particle)
+ draw(canvas)
Particle
+ coords
+ vector
+ speed
+ color
+ sprite
+ move()
+ draw(canvas)
Unit
...
+ fireAt(target: Unit)
+ move(x, y)
+ draw()
+ draw()
Circle
radius
+ Circle(x, y, radius)
+ draw()
RAM cost
coords: 8B
vector: 16B
speed: 4B
color: 4B
sprite: 20KB
≈ 21KB
× 1.000.000
21GB

Solution

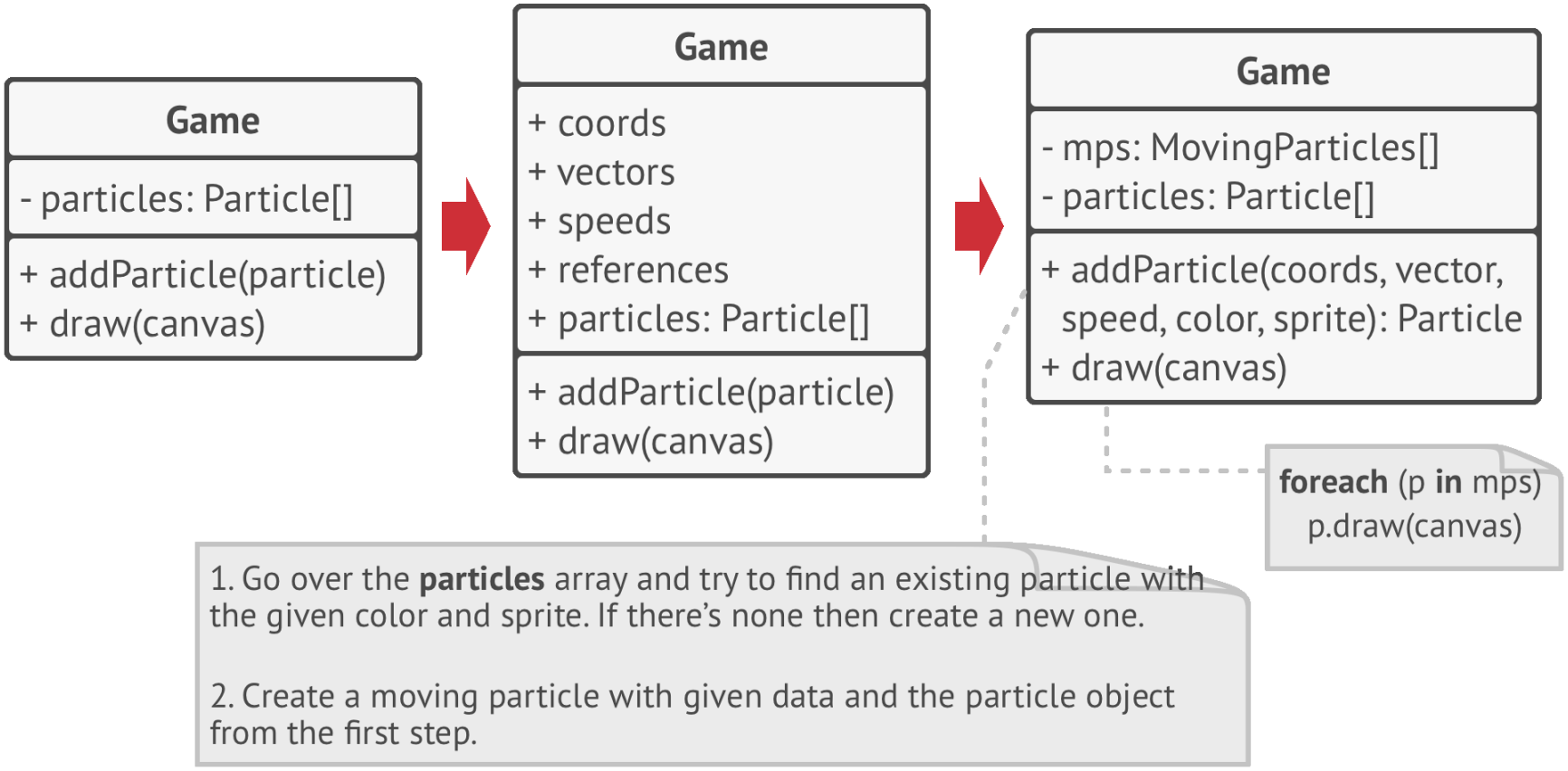

Particle
+ coords
+ vector
+ speed
+ color
+ sprite
+ move()
+ draw(canvas)
Unique (extrinsic) state
(mutable)
MovingParticle
- particle
- coords
- vector
- speed
+ MovingParticle(...)
+ move()
+ draw(canvas)
Lots
Few
Repeating (intrinsic) state
(immutable)
Particle
- color
- sprite
+ Particle(color, sprite)
+ move(coords, vector, speed)
+ draw(coords, canvas)
particle.move(
coords, vector, speed)
particle.draw(
coords, canvas)

Contd.

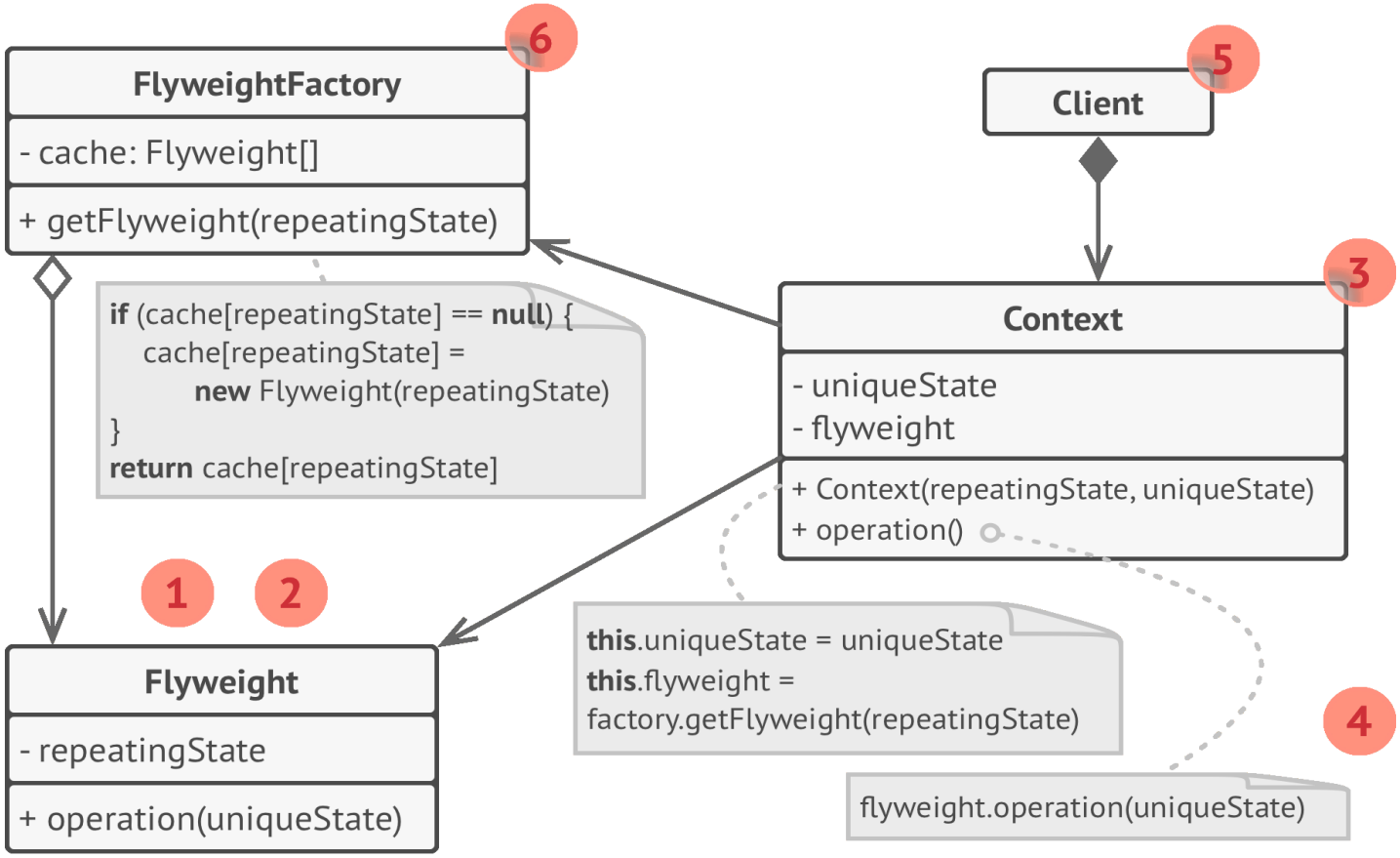

Game
+ mps: MovingParticles[]
- particles: Particle[]
+ addParticle(coords, vector, speed, color, sprite)
+ draw(canvas)
Particle
- color
- sprite
+ Particle(color, sprite)
+ move(coords, vector, speed)
+ draw(coords, canvas)
Unit
...
+ fireAt(target: Unit)
Game.addParticle(coords, target.coords, weaponPower, "red", "bullet.jpeg")
MovingParticle
- particle
- coords
- vector
- speed
+ MovingParticle(...)
+ move()
+ draw(canvas)


RAM cost
color: 4B
sprite: 20KB
≈ 21KB
coords: 8B
vector: 16B
speed: 4B
particle: 4B
≈ 32B
× 1
× 1.000.000
32MB

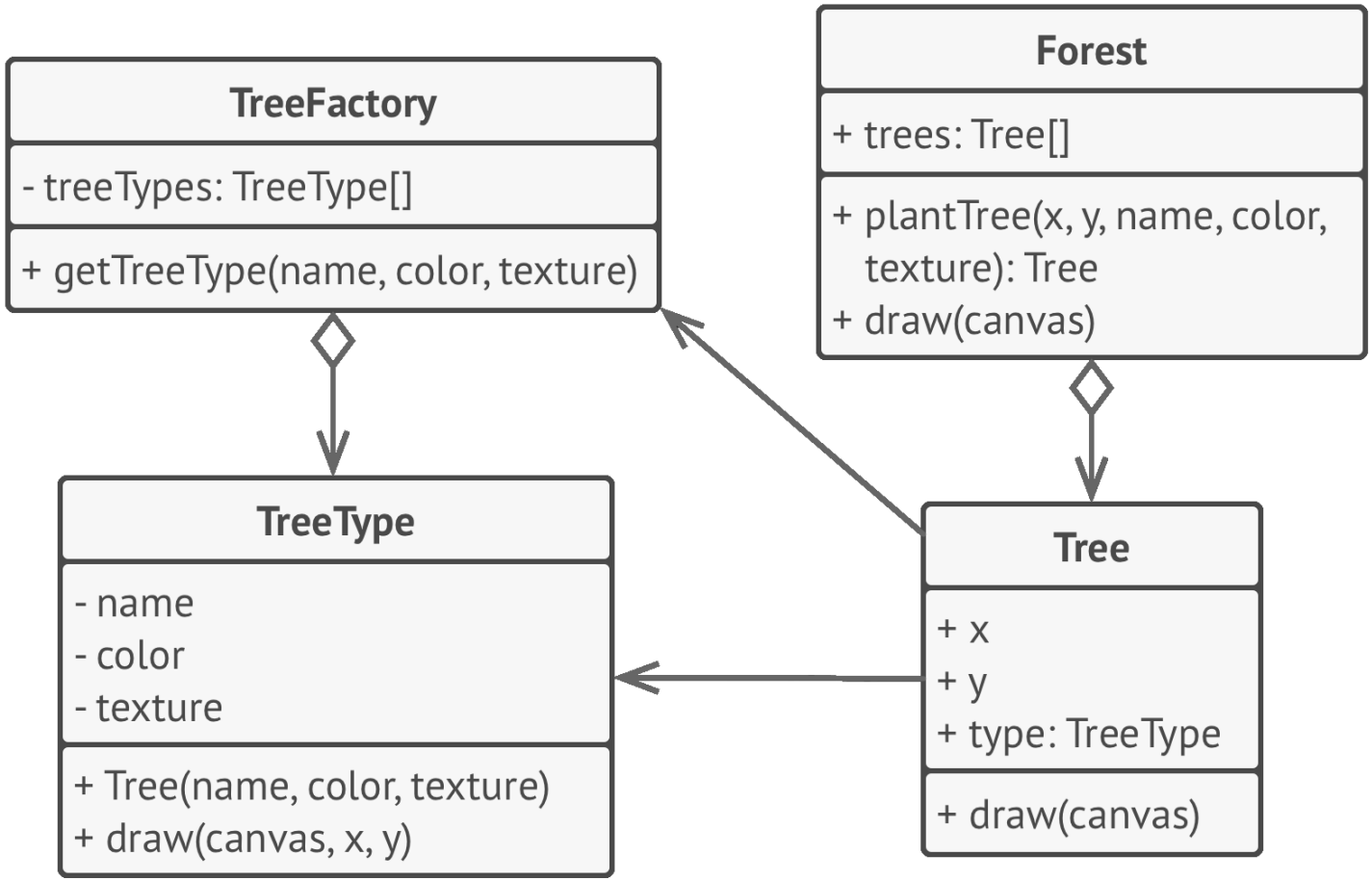
Contd.


Game
- particles: Particle[]
+ addParticle(particle)
+ draw(canvas)
Game
+ coords
+ vectors
+ speeds
+ references
+ particles: Particle[]
+ addParticle(particle)
+ draw(canvas)
Game
- mps: MovingParticles[]
- particles: Particle[]
+ addParticle(coords, vector, speed, color, sprite): Particle
+ draw(canvas)
foreach (p in mps) p.draw(canvas)
1. Go over the particles array and try to find an existing particle with the given color and sprite. If there's none then create a new one.
2. Create a moving particle with given data and the particle object from the first step.

Structure


6
FlyweightFactory
- cache: Flyweight[]
+ getFlyweight(repeatingState)
if (cache[repeatingState] == null) {
cache[repeatingState] =
new Flyweight(repeatingState)
}
return cache[repeatingState]
5
Client
3
Context
- uniqueState
- flyweight
+ Context(repeatingState, uniqueState)
+ operation()
1
2
Flyweight
- repeatingState
+ operation(uniqueState)
this.uniqueState = uniqueState
this.flyweight =
factory.getFlyweight(repeatingState)
4
flyweight.operation(uniqueState)

Pseudo code


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

Applicability

- Use the Flyweight pattern only when the program must support a huge number of objects which barely fit into available RAM.

Pros and Cons

- We can save lots of RAM, assuming that the program has tons of similar objects.
- We might be trading RAM over CPU cycles when some of the context data needs to be recalculated each time somebody calls a flyweight method.
- The code becomes much more complicated. New team members will always be wondering why the state of an entity was separated in such a way.