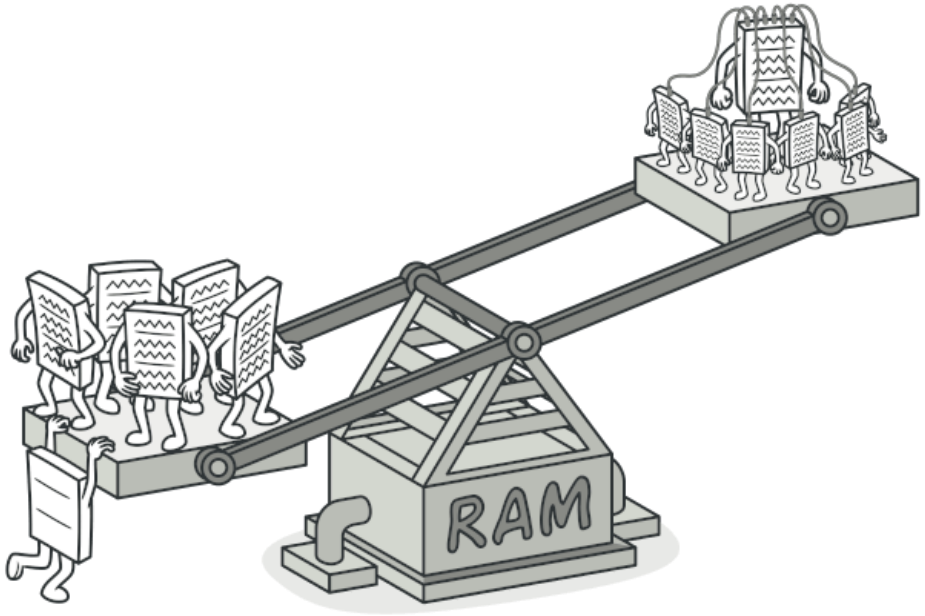
플라이웨이트 패턴
(Flyweight Pattern)

01. 플라이웨이트 패턴이란?



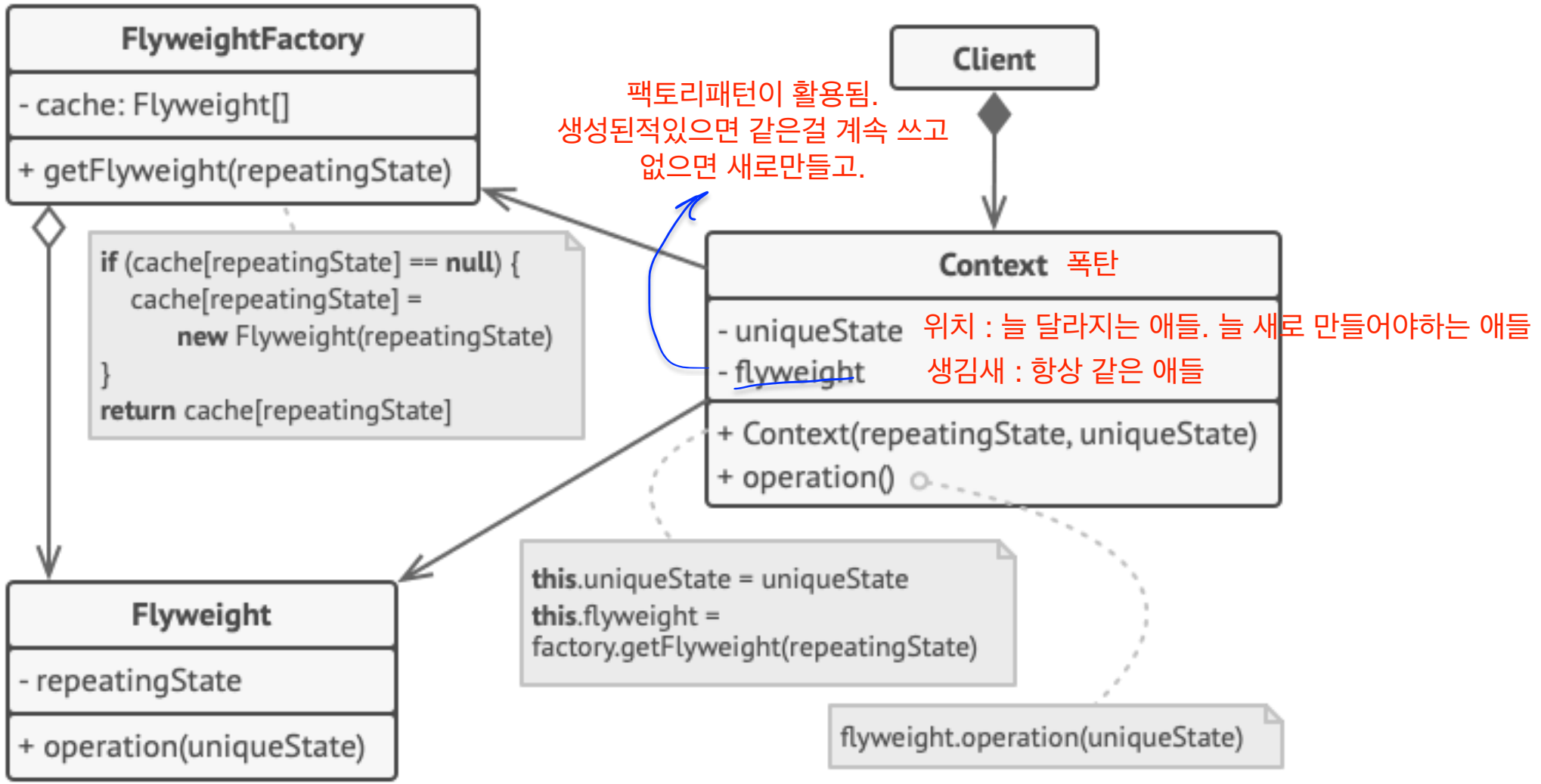
- 권투에서 가장 가벼운 체급인 플라이웨이트!(FlyWeight)
- 공통으로 사용되는 객체는 새로 생성해서 사용하지 않고 공유를 통해 효율적으로 자원을 활용, 메모리를 절약할 수 있게 하는 패턴
- 공통으로 사용되는 객체는 한 번만 생성되고 공유를 통해 풀(Pool)에 의해 관리, 사용된다.

01. 플라이웨이트 패턴이란?

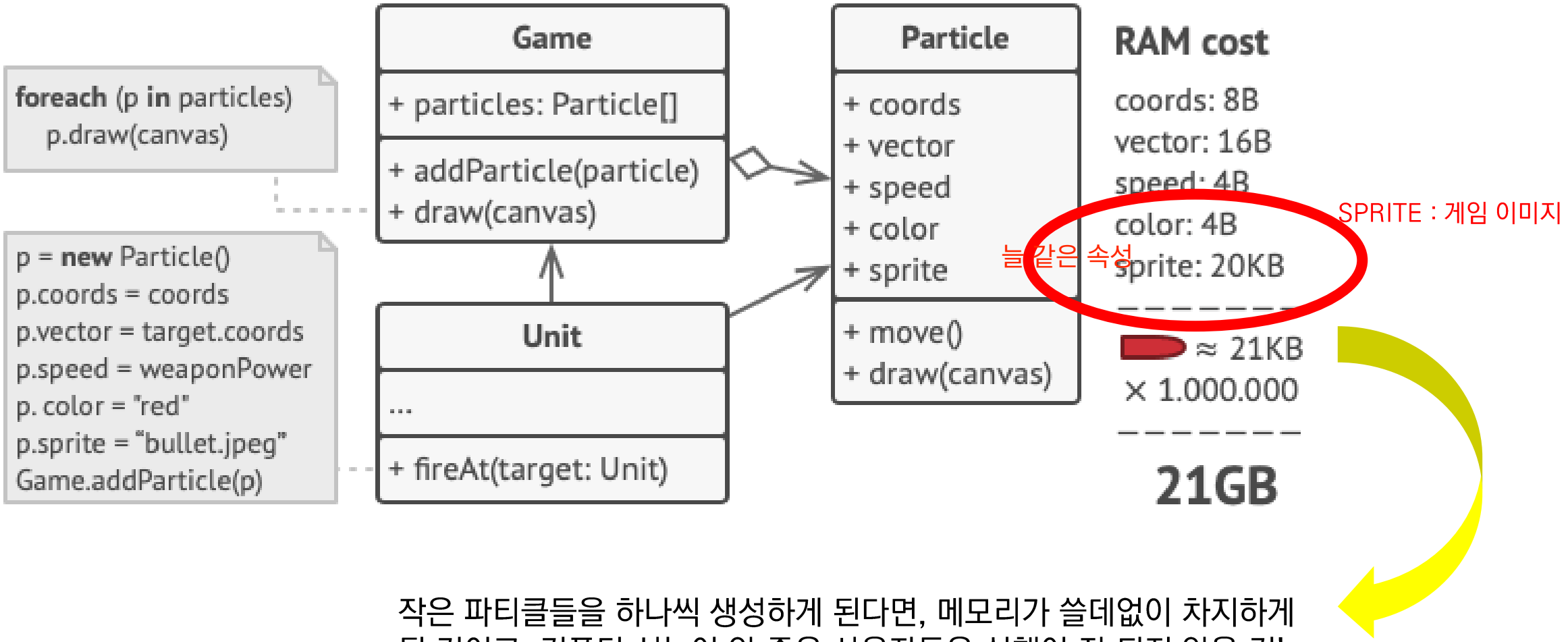
- 특정 객체들이 모두 다 똑같은 객체를 공유하고 있는 경우, 그 객체를 공유하는 방식으로 객체를 재구성하는 것!
  - 많은 객체를 만들 때 성능 향상 가능
  - 많은 객체를 만들 때 메모리 줄일 수 있음

- 많이 사용되는 예제..?
  - EX) 게임! 총알, 미사일 등이 맵에서 날아다녀야 한다면, 많아질수록 메모리 공간을 차지하니까 이를 방지함!

01. 플라이웨이트 패턴이란?

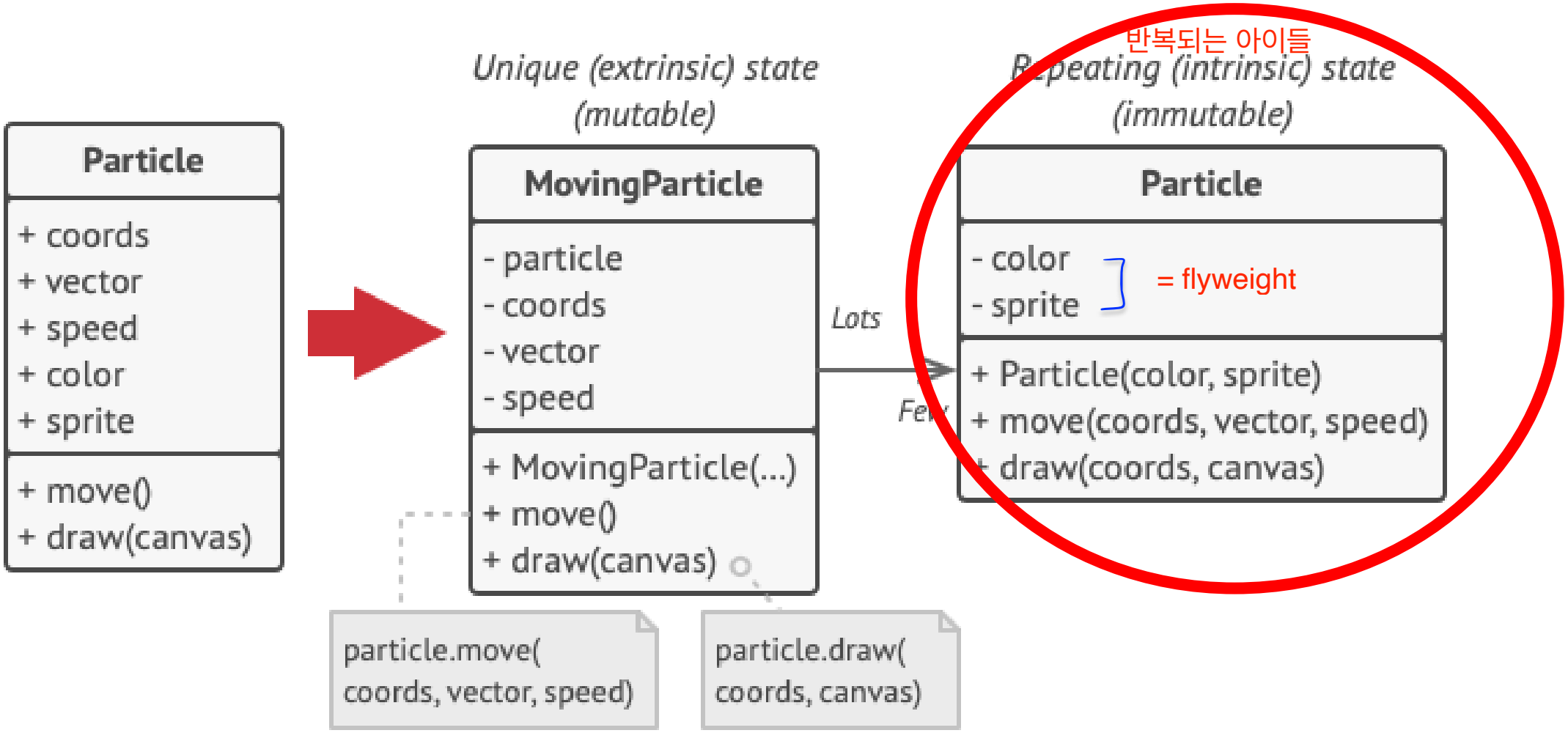


02. 플라이웨이트 패턴을 쓰는 이유?

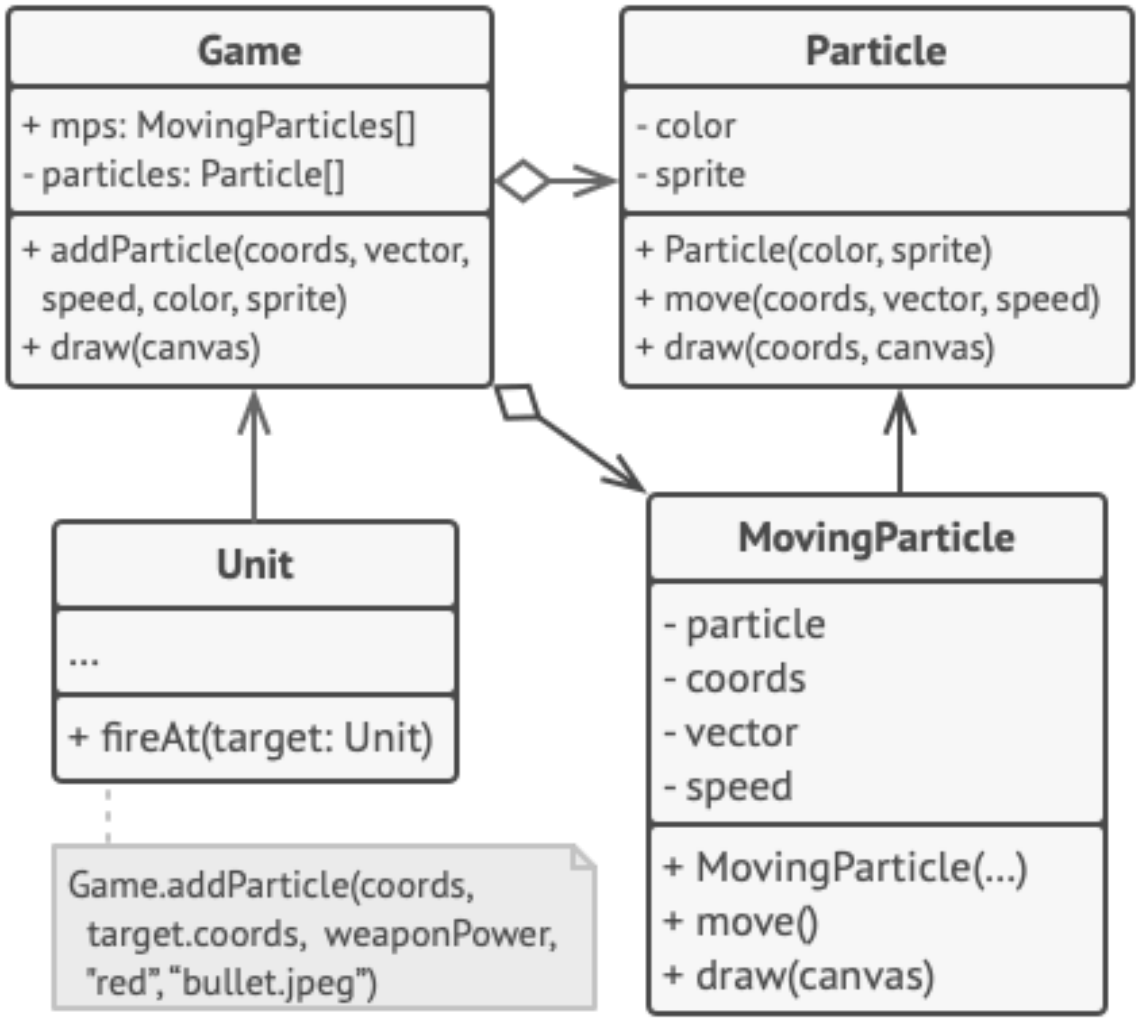


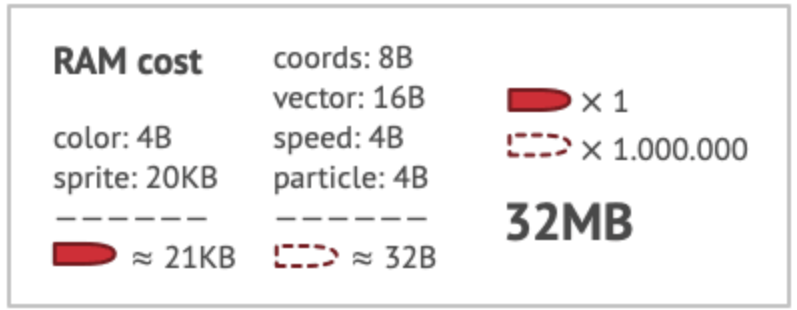
작은 파티클들을 하나씩 생성하게 된다면, 메모리가 쓸데없이 차지하게 될 것이고, 컴퓨터 성능이 안 좋은 사용자들은 실행이 잘 되지 않을 것!

02. 플라이웨이트 패턴을 쓰는 이유?

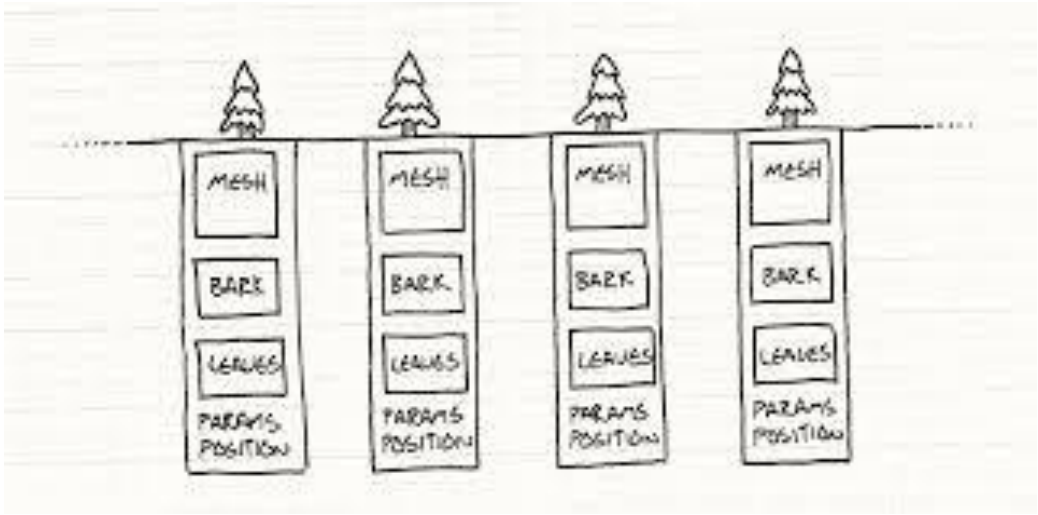


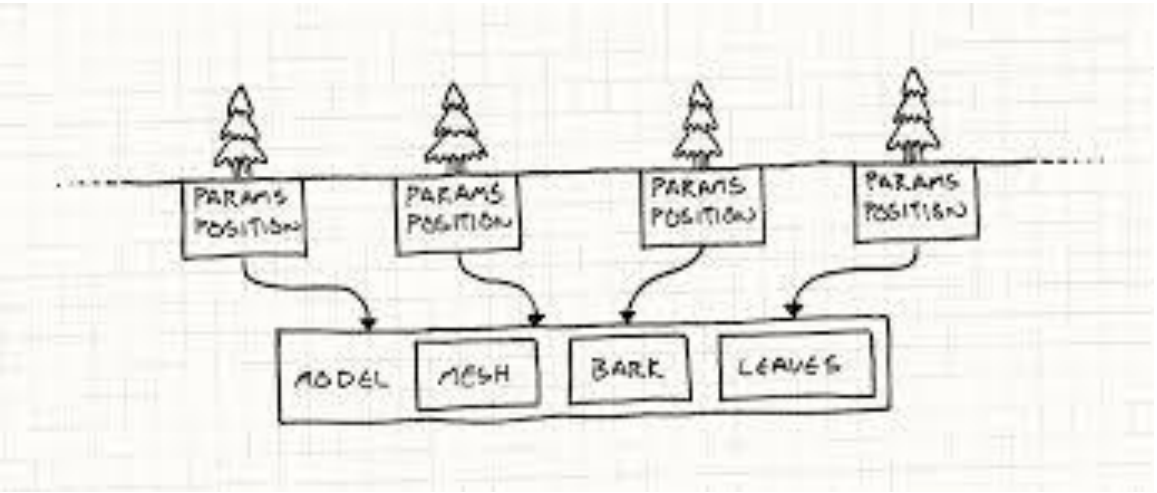
02. 플라이웨이트 패턴을 쓰는 이유?





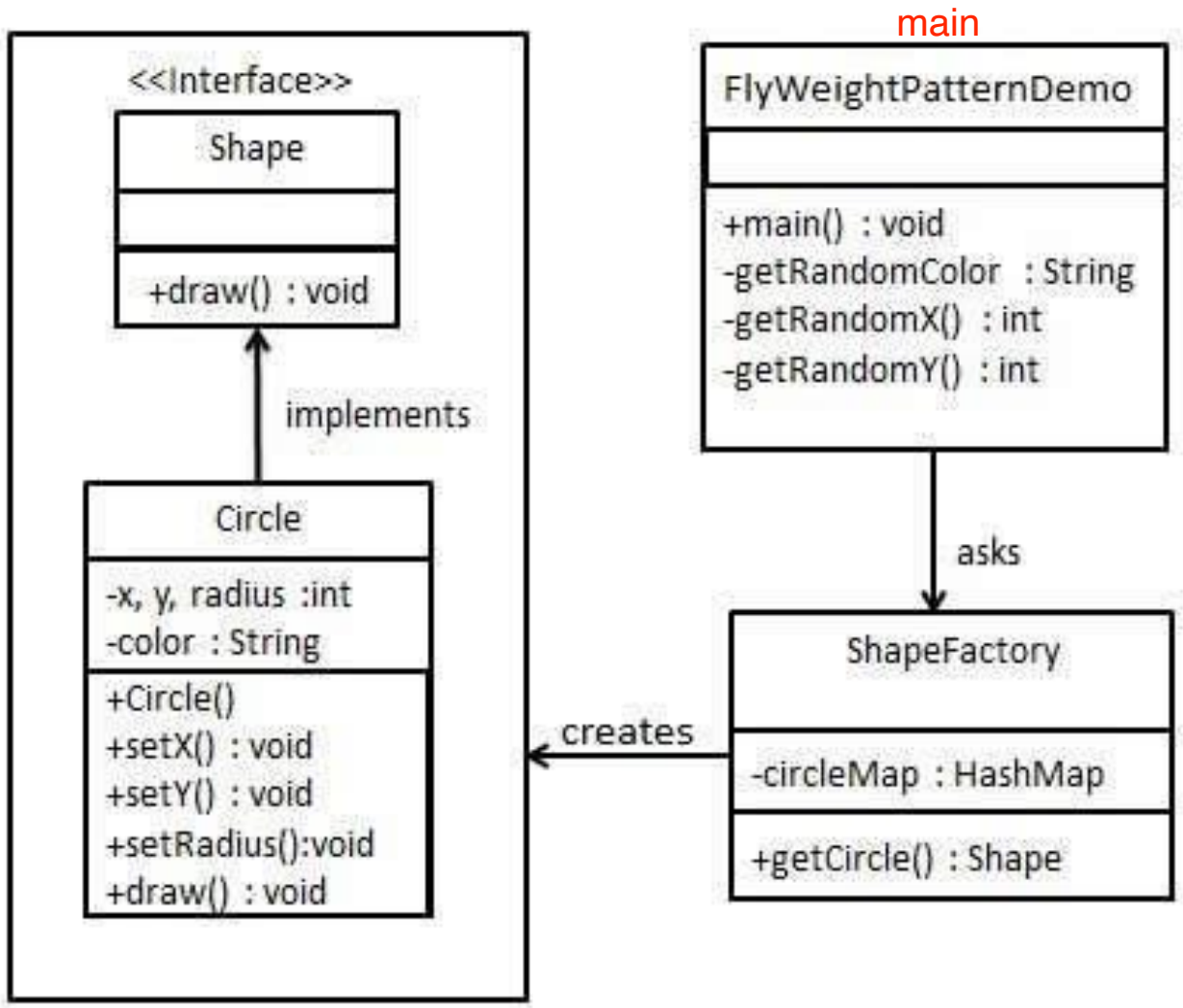
02. 플라이웨이트 패턴을 쓰는 이유?

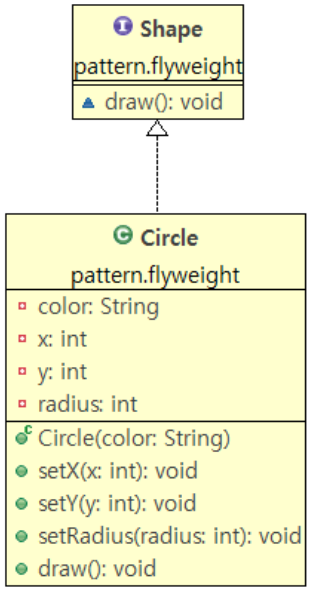


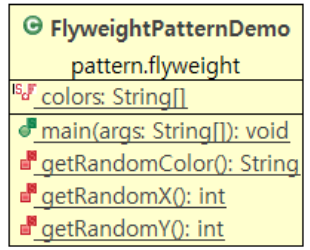


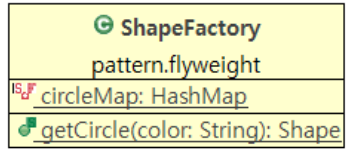
https://refactoring.guru/design-patterns/flyweight/java/example

03. 예제


<<Interface>>
Shape
+draw() : void
implements
Circle
-x, y, radius :int
-color : String
+Circle()
+setX() : void
+setY() : void
+setRadius():void
+draw() : void
main
FlyWeightPatternDemo
+main() : void
-getRandomColor : String
-getRandomX() : int
-getRandomY() : int
asks
ShapeFactory
-circleMap : HashMap
+getCircle() : Shape
creates


Shape
pattern.flyweight
draw(): void
Circle
pattern.flyweight
color: String
x: int
y: int
radius: int
Circle(color: String)
setX(x: int): void
setY(y: int): void
setRadius(radius: int): void
draw(): void


FlyweightPatternDemo
pattern.flyweight
colors: String[]
main(args: String[]): void
getRandomColor(): String
getRandomX(): int
getRandomY(): int


ShapeFactory
pattern.flyweight
circleMap: HashMap
getCircle(color: String): Shape

03. 예제

```java
public interface Shape {

    void draw();
}

```

```java
public class Circle implements Shape {

    private String color;
       private int x;
       private int y;
       private int radius;

       public Circle(String color){
          this.color = color;
       }

       public void setX(int x) {
          this.x = x;
       }

       public void setY(int y) {
          this.y = y;
       }

       public void setRadius(int radius) {
          this.radius = radius;
       }

       @Override
       public void draw() {
          System.out.println("Circle: Draw() [Color : " + color + ", x : "
                                  + x + ", y :" + y + ", radius :" + radius);
       }

}

```

color를 공유, x y radius는 항상 바뀐다.

03. 예제

```
3  import java.util.HashMap;
4
5  public class ShapeFactory {
6
7      private static final HashMap circleMap = new HashMap();
8
9      public static Shape getCircle(String color) {
10         Circle circle = (Circle)circleMap.get(color);
11
12         if(circle == null) {
13             circle = new Circle(color);
14             circleMap.put(color, circle);
15             System.out.println("Creating circle of color : " + color);
16         }
17         return circle;
18     }
19
20 }
```

final은 reference(주소값)을 변경할 수 없다.
즉, 값은 바꿀수 있지만 circleMap이 가리키는 주소값이 일정하다면
내용에 변화는 올 수 있다?

color, circle 맵

map이 generic이 안되있어서 입력값, 반환값 모두 object라 명시적 형변환해야함

03. 예제

```
3  public class FlyweightPatternDemo {
4      private static final String colors[] = { "Red", "Green", "Blue", "White", "Black" };
5          public static void main(String[] args) {
6
7              for(int i=0; i < 20; ++i) {
8                  Circle circle = (Circle)ShapeFactory.getCircle(getRandomColor());
9                  circle.setX(getRandomX());
10                 circle.setY(getRandomY());
11                 circle.setRadius(100);
12                 circle.draw();
13             }
14         }
15         private static String getRandomColor() {
16             return colors[(int)(Math.random()*colors.length)];
17         }
18         private static int getRandomX() {
19             return (int)(Math.random()*100 );
20         }
21         private static int getRandomY() {
22             return (int)(Math.random()*100);
23         }
24 }
```

빨간색 원이 이미 있으면 이미 있는 circle 객체를 반환
없으면 circle 생성자와 해당 color로 새 객체 생성

```
Creating circle of color : Black
Circle: Draw() [Color : Black, x : 36, y :71, radius :100
Creating circle of color : Green
Circle: Draw() [Color : Green, x : 27, y :27, radius :100
Creating circle of color : White
Circle: Draw() [Color : White, x : 64, y :10, radius :100
Creating circle of color : Red
Circle: Draw() [Color : Red, x : 15, y :44, radius :100
Circle: Draw() [Color : Green, x : 19, y :10, radius :100
Circle: Draw() [Color : Green, x : 94, y :32, radius :100
Circle: Draw() [Color : White, x : 69, y :98, radius :100
Creating circle of color : Blue
Circle: Draw() [Color : Blue, x : 13, y :4, radius :100
Circle: Draw() [Color : Green, x : 21, y :21, radius :100
Circle: Draw() [Color : Blue, x : 55, y :86, radius :100
Circle: Draw() [Color : White, x : 90, y :70, radius :100
Circle: Draw() [Color : Green, x : 78, y :3, radius :100
Circle: Draw() [Color : Green, x : 64, y :89, radius :100
Circle: Draw() [Color : Blue, x : 3, y :91, radius :100
Circle: Draw() [Color : Blue, x : 62, y :82, radius :100
Circle: Draw() [Color : Green, x : 97, y :61, radius :100
Circle: Draw() [Color : Green, x : 86, y :12, radius :100
Circle: Draw() [Color : Green, x : 38, y :93, radius :100
Circle: Draw() [Color : Red, x : 76, y :82, radius :100
Circle: Draw() [Color : Blue, x : 95, y :82, radius :100
```

04. 사용되는 곳

또, 대표적으로 사용되는 것이 바로 **Java의 String Pool** 입니다. Java에서는 String Pool을 별도로 두어 같은 문자열에 대해 다시 사용될 때에 새로운 메모리를 할당하는 것이 아니라 String Pool에 있는지 검사해서 있으면 가져오고 없으면 새로 메모리를 할당하여 String Pool에 등록한 후에 사용하도록 하고 있습니다.