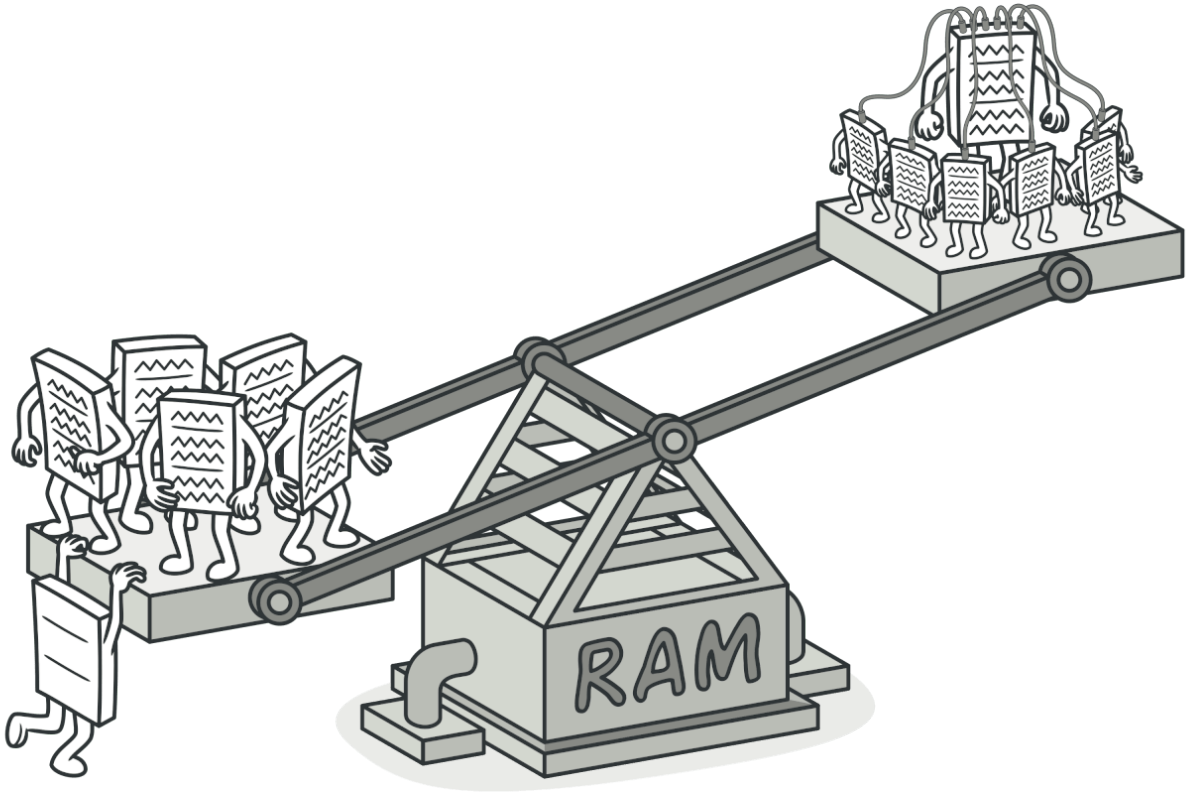
참고 자료

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

FlyweightPattern이란?

- structural design pattern
- 각 객체 별로 데이터를 유지하기보단, 공통의 요소를 공유하여 메모리를 절약하는 패턴 (다른 말로 캐싱)

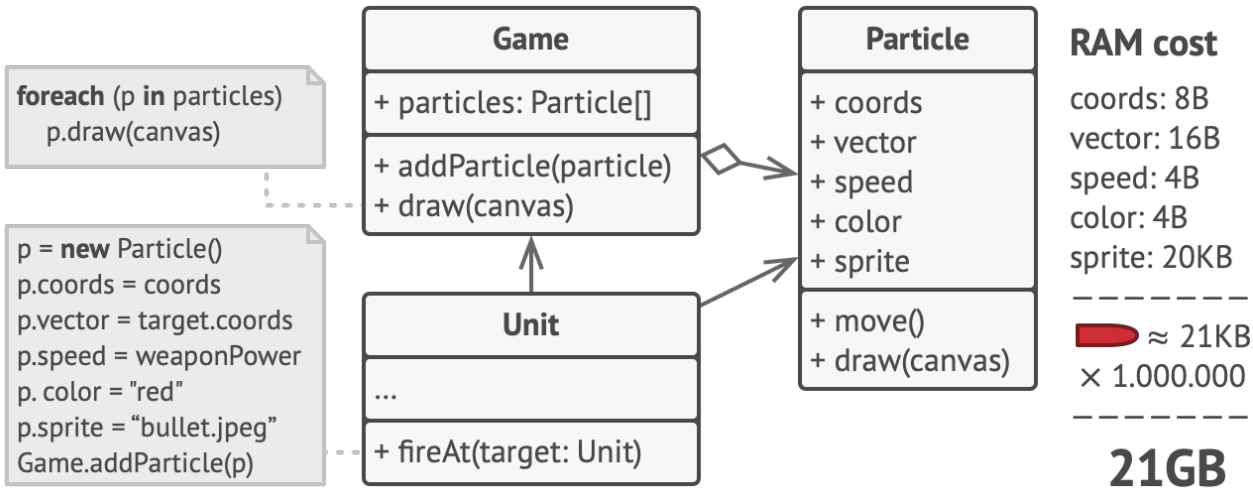


상황

- 플레이어간 총 싸움하는 게임이 있습니다.
- 이 게임의 탄(particle)의 종류엔 총알, 미사일, 샷건 총알이 있습니다.
- 그런데 실행한지 얼마 안되어 메모리 부족으로 인해 게임이 종료되었습니다.

문제의 원인

- 탄 하나당 x,y좌표, 벡터, 속도, 색, 이미지값을 가지는데, 매 총알 하나마다 이를 새로 생성해주었습니다.
- 그 탄 하나당 차지하는 용량은 아래 그림 오른쪽에 21KB정도 됩니다.
- 따라서 게임에 100만 발이 존재하게 되면 무려 메모리를 21GB를 차지하게 됩니다.



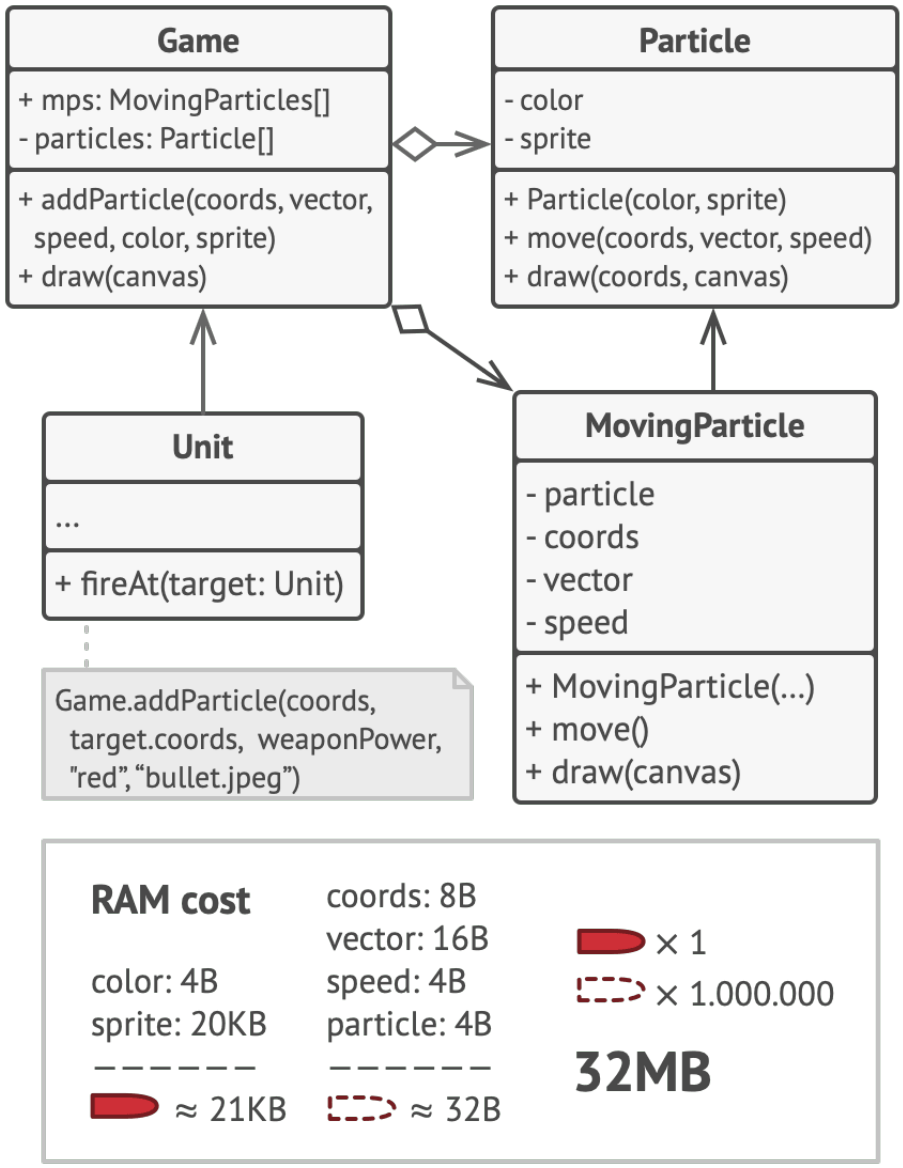
해결 방법 - 객체들의 공통 요소는 공유하라!

- 탄 중 bullet의 요소(속성)에 대해 생각해봅시다.
- bullet마다 `x,y좌표`, `vector`, `speed`는 모두 다를 수 있습니다. (이러한 객체마다 변하는 값들을 *extrinsic state*라 부릅니다.)
- 하지만 bullet마다 `색깔`과 `이미지`는 동일하지요. (이러한 고정적인 값들을 *intrinsic state*라 부릅니다.)

여기서, Flyweight 패턴은 **object 내에 오직 intrinsic state**만 가지도록 하며, 다른 문맥에 따라 **이 object를 재사용**하도록 합니다 그리고 extrinsic state를 객체 내에 가지게 하기보단, 메소드를 통해 전달 받아 메서드를 실행합니다.

한편, intrinsic state만 가지고 있는 object를 flyweight라 부릅니다.

위를 리팩토링하면 다음과 같습니다.



1. Particle(탄)엔 intrinsic요소인 color와 sprite만 가지고 있습니다.
2. extrinsic요소는 MovingParticle에 따로 빼놓습니다.
3. **particle 필드를 통해 Particle 객체를 참조**하도록 합니다. 이 객체를 생성하지 않음으로써 메모리를 절약할 수 있는 것이죠
4. 그리고 Game이라는 클래스에서 생성한 intrinsic object를 관리함으로써 Particle을 생성할 때 참조할 수 있도록 합니다.
5. 그리고 origin particle은 MovingParticles object에 의해 선택됩니다. 그리고 관련 동작은 extrinsic 값을 Particle의 메서드로 넘겨줌으로써 particle에서 실행되는 구조를 가지고 있지요.

구현시 주의점

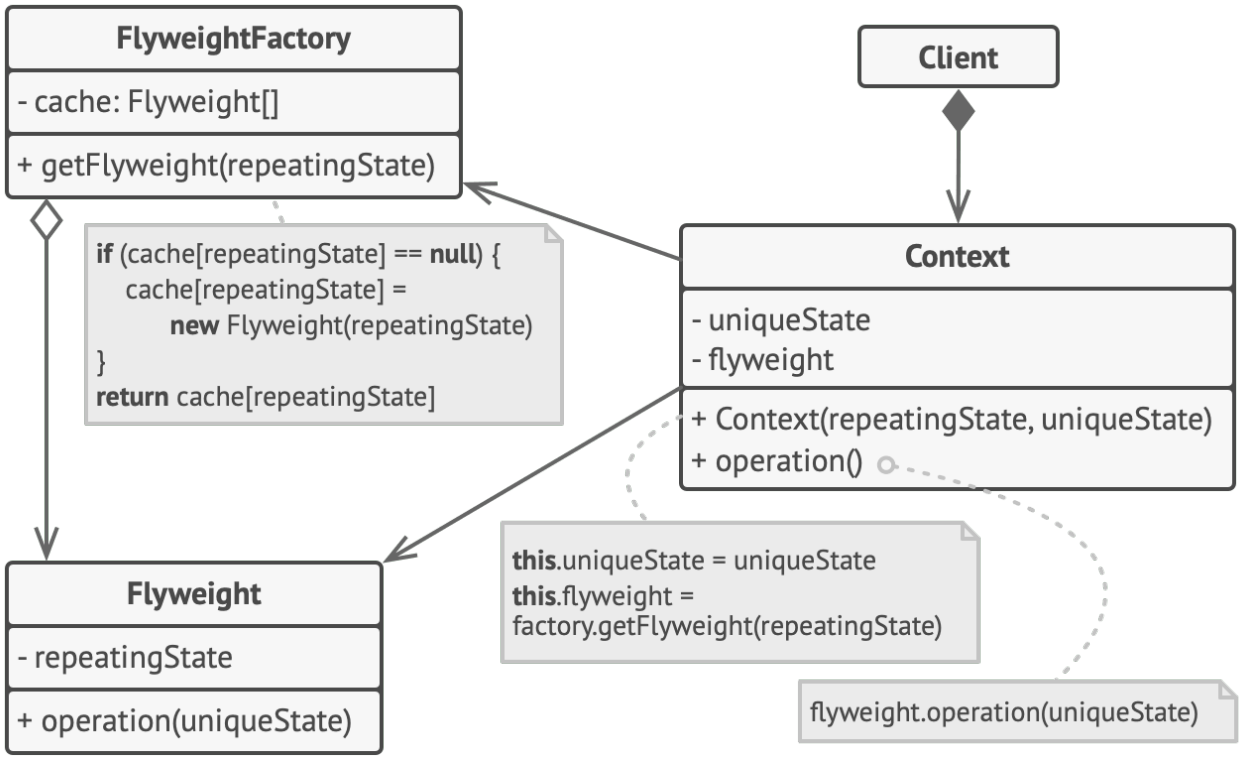
서로 다른 문맥에서 이 값을 재사용하고 있기 때문에 intrinsic state는 반드시 immutable해야 합니다.

구조

위 케이스의 경우 Game이 다른 책임을 가지고 있을 수 있음에 불구하고 리소스 관리 및 생성 책임을 맡고 있습니다. 따라서 해당 책임은 FactoryMethod패턴인 FlyweightFactory에 맡기는 것이 일반적입니다.

original object 표현

기존 object의 특성이 intrinsic과 extrinsic부분으로 찢어졌다라는 의미에서 original object라 표현.



1. FlyweightFactory

- flyweight를 생성하는 부분입니다.
- client가 생성하려는 값이 없을 경우에만 생성합니다.

2. Flyweight

- 공유되어지는 object의 상태 값을 가지고 있습니다.
- 관련 메서드는 unique state를 받아 실행합니다.

3. Context

- 예제에서 particle에 해당하는 부분입니다.
- Context의 생성은 intrinsic 부분과 extrinsic부분의 결합으로 생성됩니다.
- original object의 행동을 구현하는 방식은 2가지가 있습니다.
    1. flyweight에서 extrinsic state를 받아 operation를 실행하는 메서드를 작성합니다. 그리고 Context

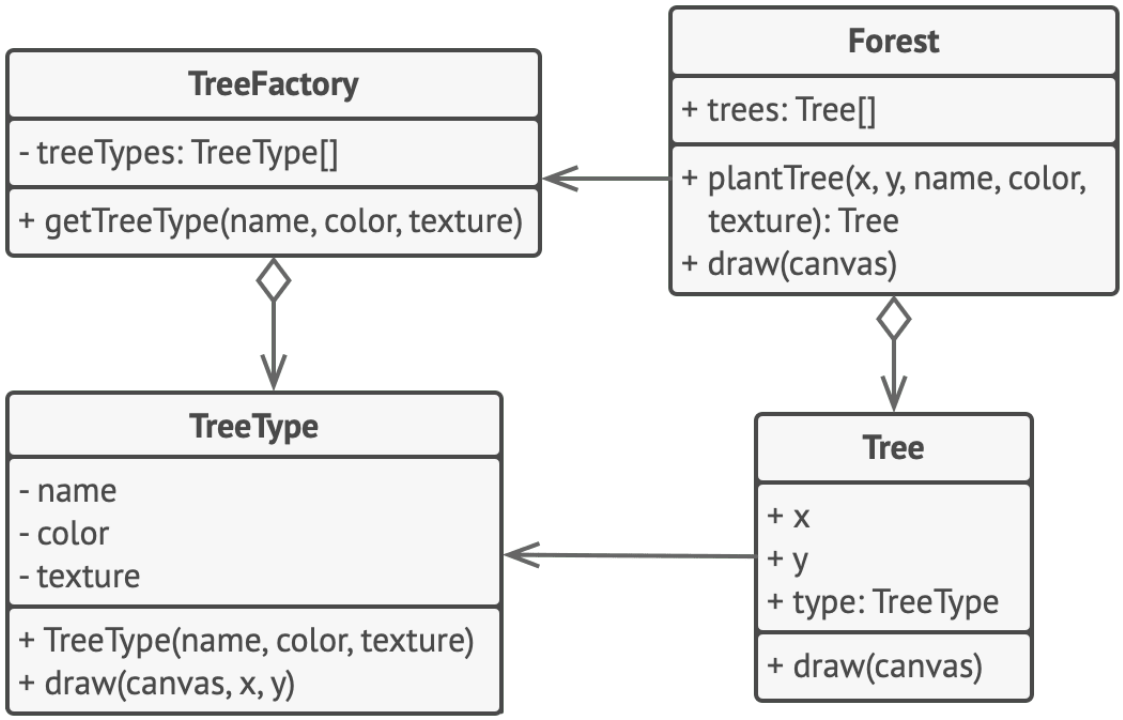
에서 flyweight에 extrinsic state를 담아 oepration을 flyweight에 위임합니다.
2. 해당 class에 operation을 둠으로써 original object의 기능을 구현합니다. 이때 flyweight는 단순히 데이터를 담는 object가 됩니다.

4. Client

- 클라이언트는 estrinsic state를 계산하고 저장합니다.
- 클라이언트 관점에서는 flyweight는 런타임 중에 특정 문맥적인 데이터의 값을 넣을 수 있는 템플릿이 됩니다.

  총게임 예를 들면, 유저의 캐릭터는 특정 위치(extrinsic)에서 총알을 발사합니다. 그러면 총알 이미지(intrinsic)가 그 방향으로 날라가게 되는 것이죠.

예시 코드 - 나무 그리기



Forest

```
public class Forest extends JFrame {
    private List<Tree> trees = new ArrayList<>();

    public void plantTree(int x, int y, String name, Color color, String
otherTreeData) {
        TreeType type = TreeFactory.getTreeType(name, color, otherTreeData);
        Tree tree = new Tree(x, y, type);
```

```
        trees.add(tree);
    }

    @Override
    public void paint(Graphics graphics) {
        for (Tree tree : trees) {
            tree.draw(graphics);
        }
    }
}
```

Tree

```
public class Tree {
    private int x;
    private int y;
    private TreeType type;

    public Tree(int x, int y, TreeType type) {
        this.x = x;
        this.y = y;
        this.type = type;
    }

    public void draw(Graphics g) {
        type.draw(g, x, y);
    }
}
```

TreeType

```
public class TreeType {
    private String name;
    private Color color;
    private String otherTreeData;

    public TreeType(String name, Color color, String otherTreeData) {
        this.name = name;
        this.color = color;
        this.otherTreeData = otherTreeData;
    }

    public void draw(Graphics g, int x, int y) {
        g.setColor(Color.BLACK);
```

```
        g.fillRect(x - 1, y, 3, 5);
        g.setColor(color);
        g.fillOval(x - 5, y - 10, 10, 10);
    }
}
```

TreeFactory

```
public class TreeFactory {
    static Map<String, TreeType> treeTypes = new HashMap<>();

    public static TreeType getTreeType(String name, Color color, String
otherTreeData) {
        TreeType result = treeTypes.get(name);
        if (result == null) {
            result = new TreeType(name, color, otherTreeData);
            treeTypes.put(name, result);
        }
        return result;
    }
}
```