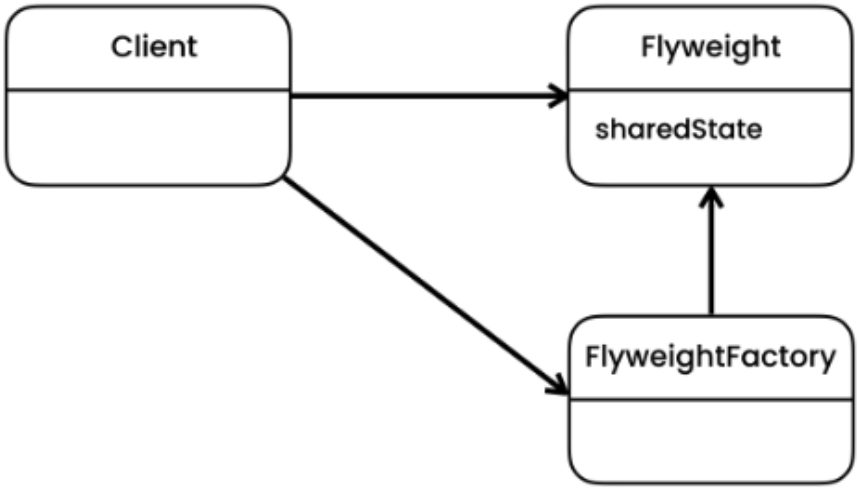
Flyweight Pattern:

Definition: Lets us 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.



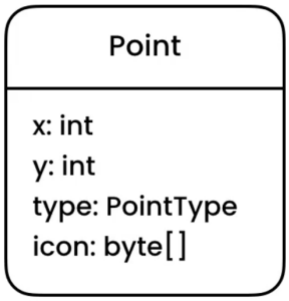
Scenario: We want to implement a point object which has x,y coordinates, type and icon.

Implementation:

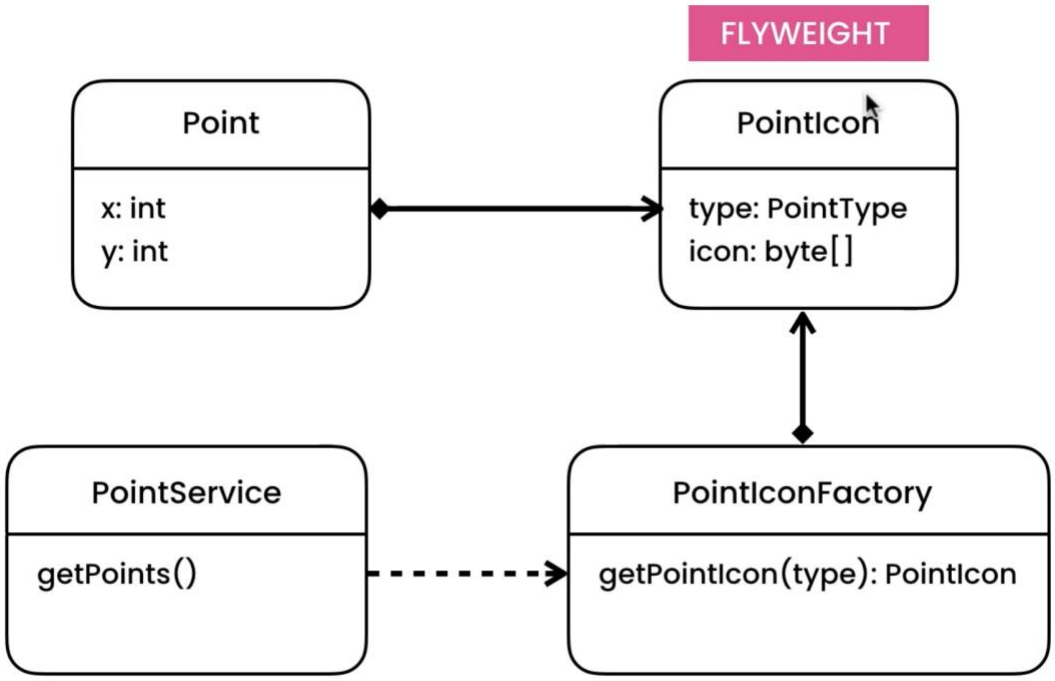
```
public class Point {
   private int x;
   private int y;
   private PointType iconType;
   private byte[] icon;


   public Point(int x, int y, PointType iconType, byte[] icon){
      this.x = x;
      this.y = y;
      this.iconType = iconType;
      this.icon = icon;
   }
   public void draw(){
      System.out.printf("Point is drawn at given coordinnates");
   }
}
```

Problem: Suppose the Point.PointType = café and it has 100 objects, so we will save 100 instaces of same café icon which will consume a lot of memory.



Solution: Create a class PointIcon which has the image and set the composition relation with Point class, with this implementation we can re-use PointIcon across multiple objects. To make sure if it has one instance in memory, ask PointIconFactory to give us the café PointIcon, it will create in memory if that icon didn't exist before, else return café pointIcon. So PointIcon behave as flyweight which we can share.




```
public class Point {
    private int x;
    private int y;
```

```
    private PointIcon icon;



    public Point(int x, int y, PointIcon icon){
        this.x = x;
        this.y = y;
        this.icon = icon;
    }
    public void draw(){
        System.out.printf("%s at (%d, %d)", icon.getType(), x, y);
    }
}

public class PointIconFactory {
    private Map<PointType, PointIcon> icons = new HashMap<>();
    public PointIcon getPointIcon(PointType type){
        if(!icons.containsKey(type)){
            var icon = new PointIcon(type, null);
            icons.put(type, icon);
        }
        return icons.get(type);
    }
}

public class PointService {
    private PointIconFactory iconFactory;
    public PointService(PointIconFactory iconFactory) {
        this.iconFactory = iconFactory;
    }

    public List<Point> getPoints(){
        var points = new ArrayList<Point>();
        var point = new Point(1,2, iconFactory.getPointIcon(PointType.CAFE));
        points.add(point);
        return points;
    }
}
```