Приспособленец

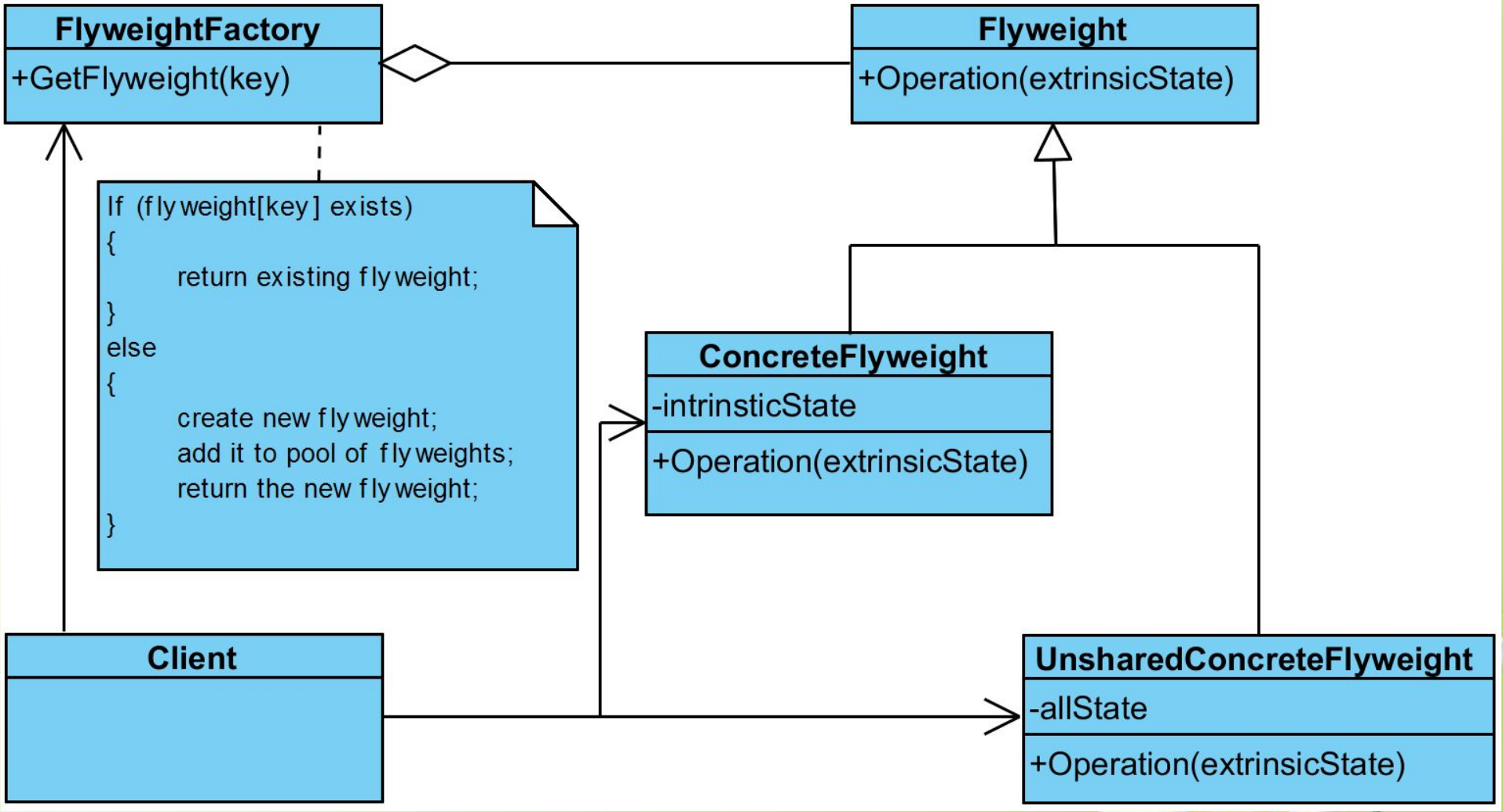
(англ. Flyweight, "легковесный (элемент)")

Цель

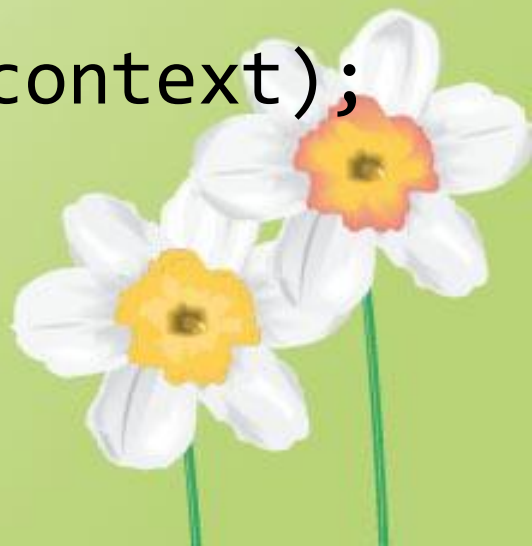
- Оптимизация работы с памятью путём предотвращения создания экземпляров элементов, имеющих общую сущность.

Описание

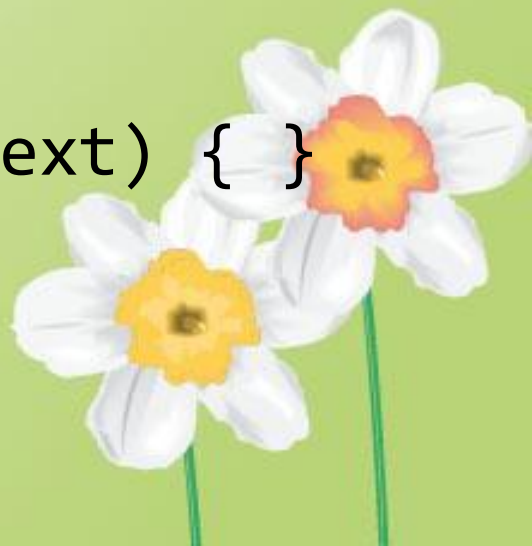
- Flyweight используется для уменьшения затрат при работе с большим количеством мелких объектов. При проектировании приспособленца необходимо разделить его свойства на внешние и внутренние. Внутренние свойства всегда неизменны, тогда как внешние могут отличаться в зависимости от места и контекста применения и должны быть вынесены за пределы приспособленца.


FlyweightFactory
+GetFlyweight(key)
Flyweight
+Operation(extrinsicState)
If (flyweight[key] exists)
{
return existing flyweight;
}
else
{
create new flyweight;
add it to pool of flyweights;
return the new flyweight;
}
ConcreteFlyweight
-intrinsticState
+Operation(extrinsicState)
Client
UnsharedConcreteFlyweight
-allState
+Operation(extrinsicState)

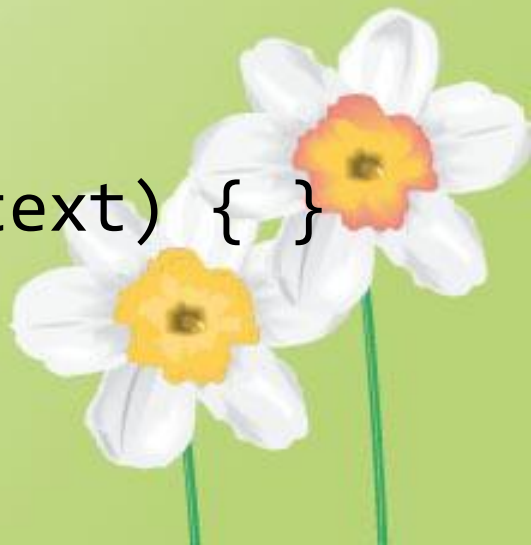
```
/*
 * Интерфейс приспособленца
 */
public interface Primitive {
   /*
    * Метод отрисовки примитива с передачей заданного контекста рисования
    */
   public void draw(Context context);
}
```



```
/*
 * Окружнсоть - разделяемый приспособленец.
Внутреннее состояние - радиус
 */
public class Circle implements Primitive {
  private int radius;

  public Circle(int radius) {
    this.radius = radius;
  }

 @Override
  public void draw(Context context) { }
}
```



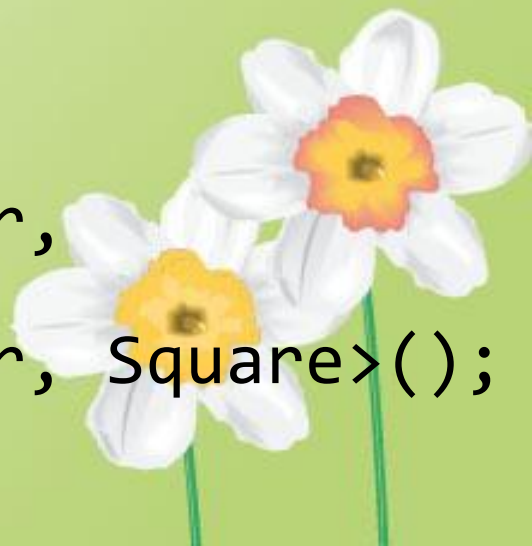
```
/*
 * Разделяемый приспособленец - Квадрат.
 * Внутренее состояние - высота, ширина.
 */
public class Square implements Primitive {
  private int height, width;

  public Square(int height, int width) {
    this.height = height;
    this.width = width;
  }

 @Override
  public void draw(Context context) { }
}
```



```
/*
 * Разделяемый приспособленец - точка
 */
 public class Point implements Primitive
{
    @Override
    public void draw(Context context) { }
 }
```

```
/*
 * Контекст рисования, передается клиентом примитиву для отрисовки последнего
 */
public final class Context {
    public final int x;
    public final int y;
    public final Color color;

    public Context(int x, int y, Color collor) {
        this.x = x;
        this.y = y;
        this.color = collor;
    }
}
```

```
/*
 * Фабрика приспособленцев.
 * Реализует разделение оных на основании их внутренних состояний.
 *
*/
 public abstract class PrimitiveFactory {

    private static Point onePoint;
    private static Map<Integer, Circle> circles;
    private static Map<Integer, Square> squares;

    static {
      circles = new HashMap<Integer, Circle>();
      squares = new HashMap<Integer, Square>();
    }
```



```
  public static synchronized Circle
createCircle(int radius) {
      if (circles.get(radius) == null) {
        circles.put(radius, new Circle(radius));
      }

      return circles.get(radius);
    }

    public static synchronized Square
createSquare(int height, int width) {
      if (squares.get(height*10+width) == null) {
        squares.put(height*10+width, new
Square(height, width));
      }

      return squares.get(height*10+width);
    }
```

```
public static synchronized Point
createPoint() {
      if (onePoint == null) {
        onePoint = new Point();
      }

      return onePoint;
    }
  }
```

Summary

- (+) Можно получить ощутимую экономию по памяти.
  (–) Однако, возможно, за это придется заплатить временем на поиск/передачу/вычисление внешнего состояния.
  (?) Важно понимать, что применимость данного паттерна определяется, в первую очередь тем, на сколько четко идентифицируются внутреннее и внешнее состояния объектов системы.