We can use flyweight pattern in following scenarios:

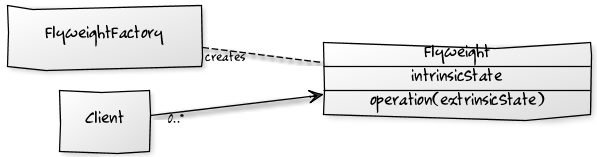
* When we need many similar objects that are unique in terms of only a few parameters and most of the stuffs are common in general.
* We need to control the memory consumption by large number of objects – by creating fewer objects and sharing them across.

Extrinsic and intrinsic attributes

A flyweight objects essentially has two kind of attributes – intrinsic and extrinsic.

An **intrinsic** state attribute is stored/shared in the flyweight object, and it is independent of flyweight’s context. As the best practice, we should make intrinsic states [immutable](https://howtodoinjava.com/java/basics/how-to-make-a-java-class-immutable/).

An **extrinsic** state varies with flyweight’s context, which is why they cannot be shared. Client objects maintain the extrinsic state, and they need to pass this to a flyweight object during object creation.



Exercise:

In given example, we are building a Paint Brush application where client can use brushes on three types – THICK, THIN and MEDIUM. All the thick (thin or medium) brush will draw the content in exact similar fashion – only the content color will be different.

Here are some of the classes you’ll need to implement the pattern.

* Pen.java (interface)
* BrushSize.java (enum)
* ThickPen.java
* ThinPen.java
* MediumPen.java
* PenFactory.java
* PainBrushClient.java (main)

Check at our repository of Flyweight exercise the code and add the missing classes to obtain the next output:

Graphical user interface, text

Description automatically generated

Check the complete solution at:

[Flyweight Design Pattern - Flyweight Pattern in Java- HowToDoInJava](https://howtodoinjava.com/design-patterns/structural/flyweight-design-pattern/)