FLYWEIGHT AND DECORATOR
USE SHARING to support large numbers of fine-grained objects efficiently.
http://www.gameprogrammingpatterns.com/flyweight.html#forest-for-the-trees
Flyweight splits the state of an object **INTO TWO**

**INTRINSIC STATE**
- Identical
- Shared
- Pooled
- Single instance

*The mesh and texture of the rendered tree.*

**EXTRINSIC STATE**
- Different
- Each has its own
- Multiple instances

*The position of the rendered tree.*
FlyweightFactory

GetFlyweight(key)

if (flyweight[key] exists) {
    return existing flyweight;
} else {
    create new flyweight;
    add it to pool of flyweights;
    return the new flyweight;
}

Flyweight

Operation(extrinsicState)

ConcreteFlyweight

Operation(extrinsicState)

intrinsicState

UnsharedConcreteFlyweight

Operation(extrinsicState)

allState

Client
DECORATOR PATTERN
Attach **ADDITIONAL RESPONSIBILITIES** to an object **DYNAMICALLY**.

Decorators provide a **FLEXIBLE ALTERNATIVE TO SUBCLASSING** for extending functionality.
HOMEWORK: 01.03 – 15.03

Read Flyweight chapter from Game Programming Patterns
http://www.gameprogrammingpatterns.com/command.html

Read Flyweight chapter from CGLearn
https://cglearn.eu/student/materials

Complete Flyweight Replay task in CGLearn