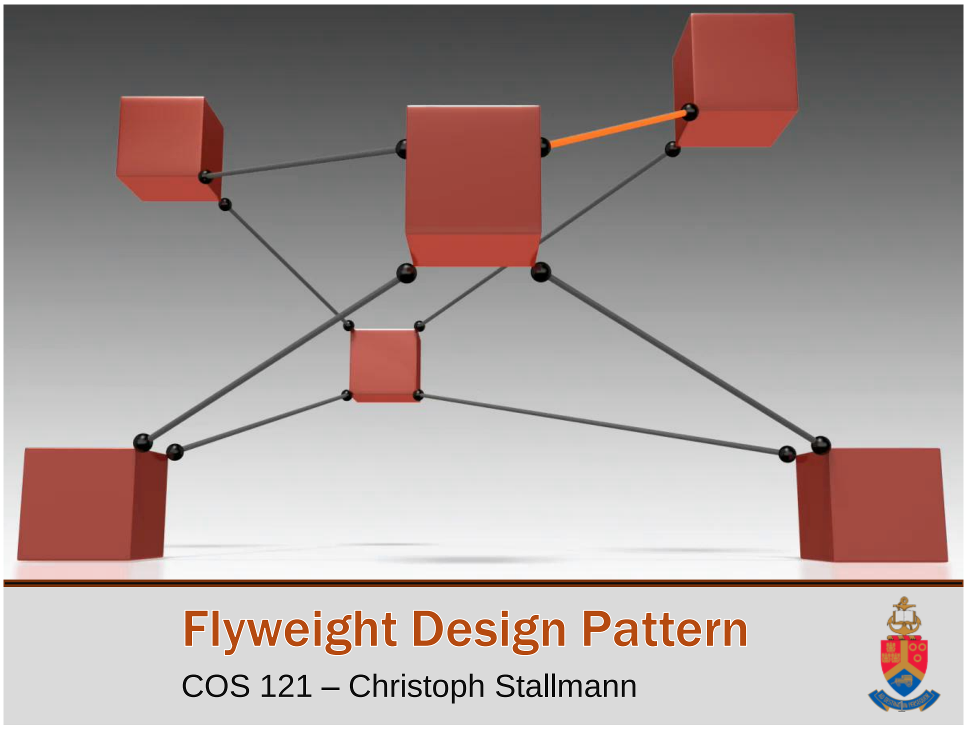
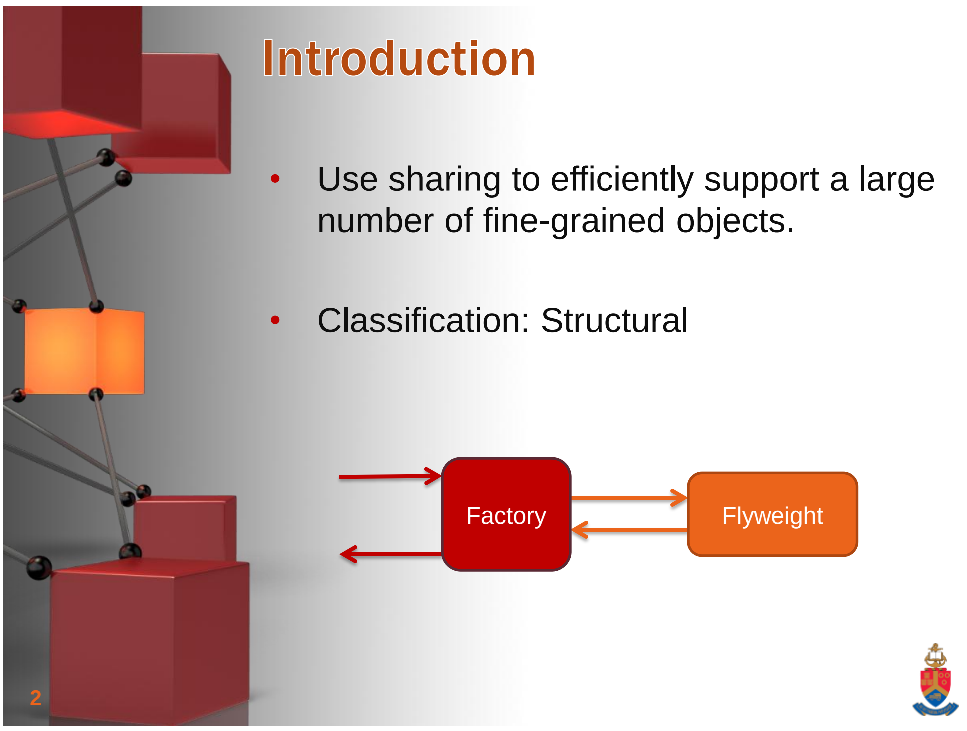


Flyweight Design Pattern

COS 121 – Christoph Stallmann

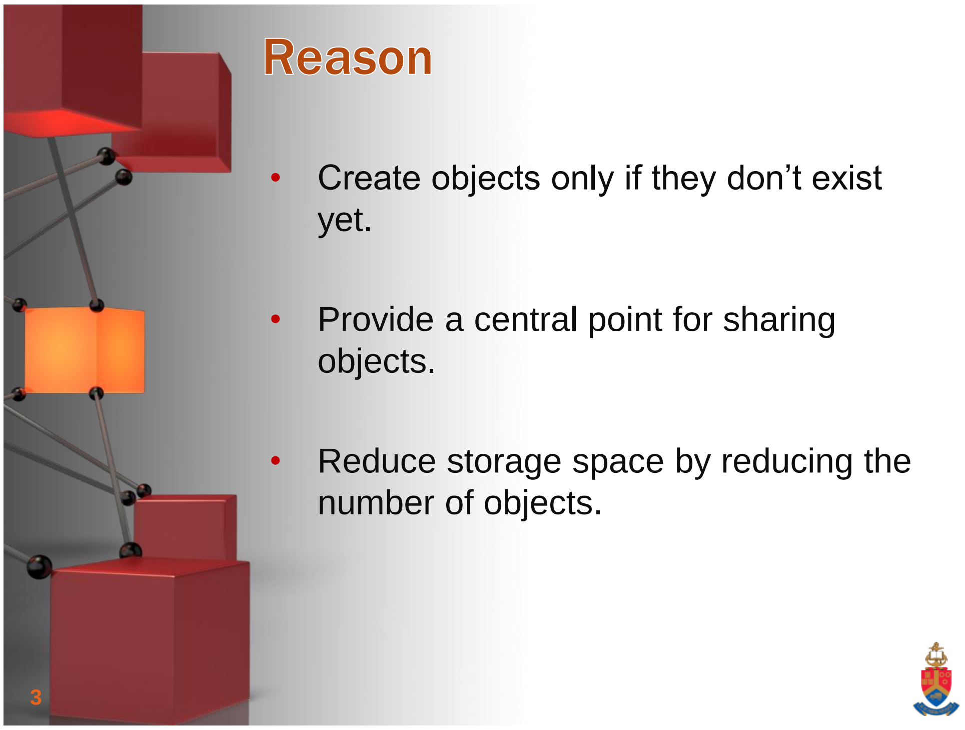


Introduction

- Use sharing to efficiently support a large number of fine-grained objects.
- Classification: Structural

Factory
Flyweight

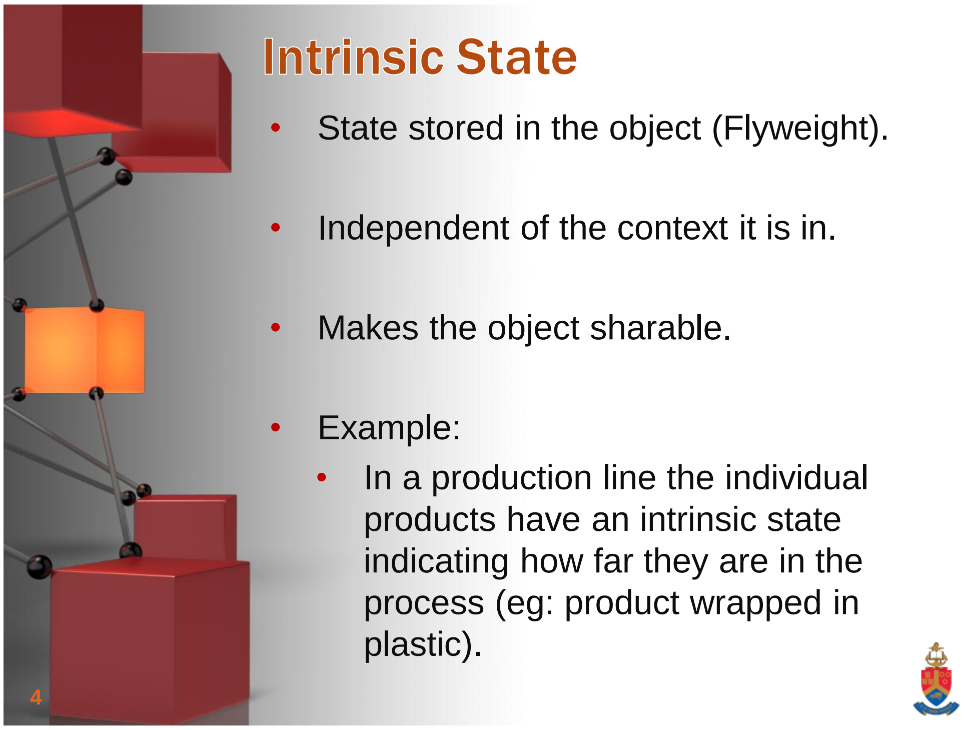
2



Reason

- Create objects only if they don’t exist yet.
- Provide a central point for sharing objects.
- Reduce storage space by reducing the number of objects.

3



Intrinsic State

- State stored in the object (Flyweight).
- Independent of the context it is in.
- Makes the object sharable.
- Example:
  - In a production line the individual products have an intrinsic state indicating how far they are in the process (eg: product wrapped in plastic).

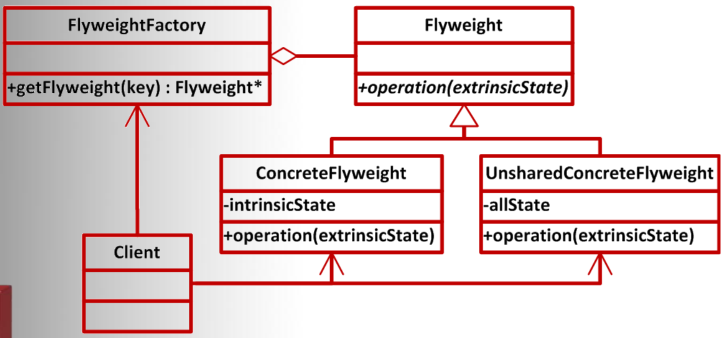
4

Extrinsic State

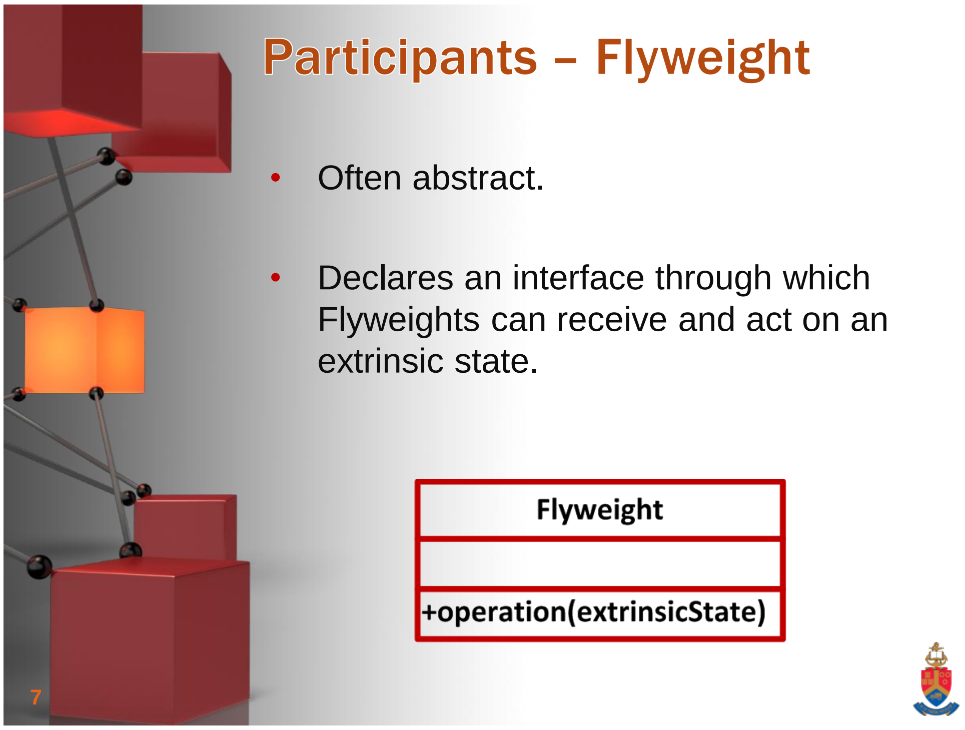
- State varies/changes with the context.
- Dependent on the context it is in.
- Object cannot be shared.
- Example:
  - Randomly every ± 1000th product's production number is scanned for a competition (lucky draw). This extrinsic state is not stored in the product.

5

Structure


FlyweightFactory
+getFlyweight(key) : Flyweight*
Flyweight
+operation(extrinsicState)
ConcreteFlyweight
-intrinsicState
+operation(extrinsicState)
UnsharedConcreteFlyweight
-allState
+operation(extrinsicState)
Client

6

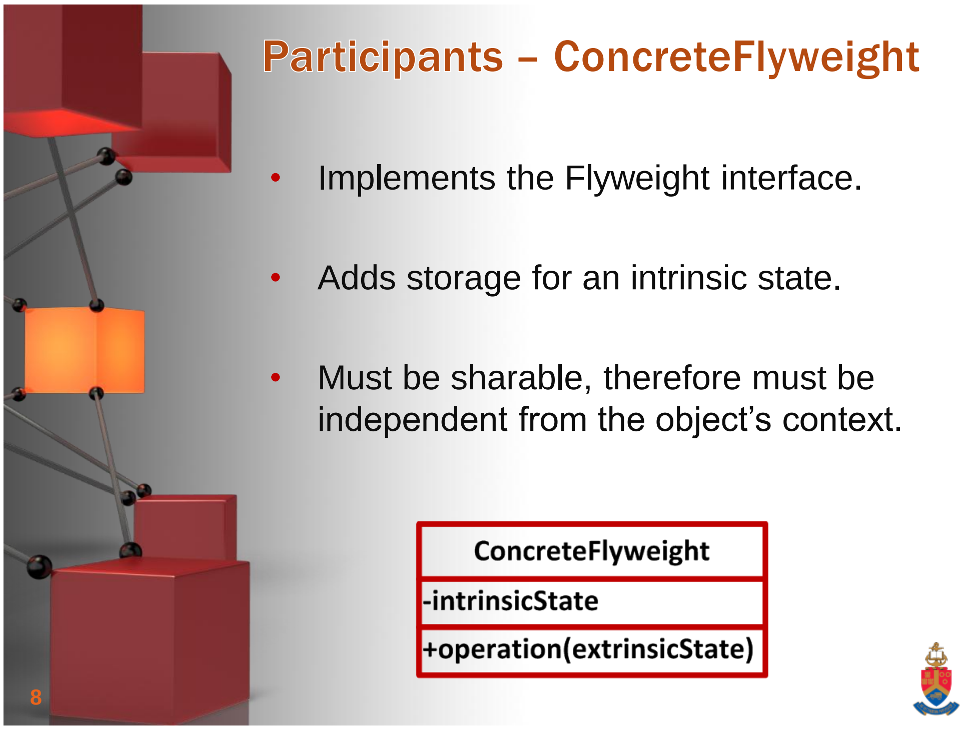


Participants – Flyweight

- Often abstract.
- Declares an interface through which Flyweights can receive and act on an extrinsic state.

Flyweight
+operation(extrinsicState)

7

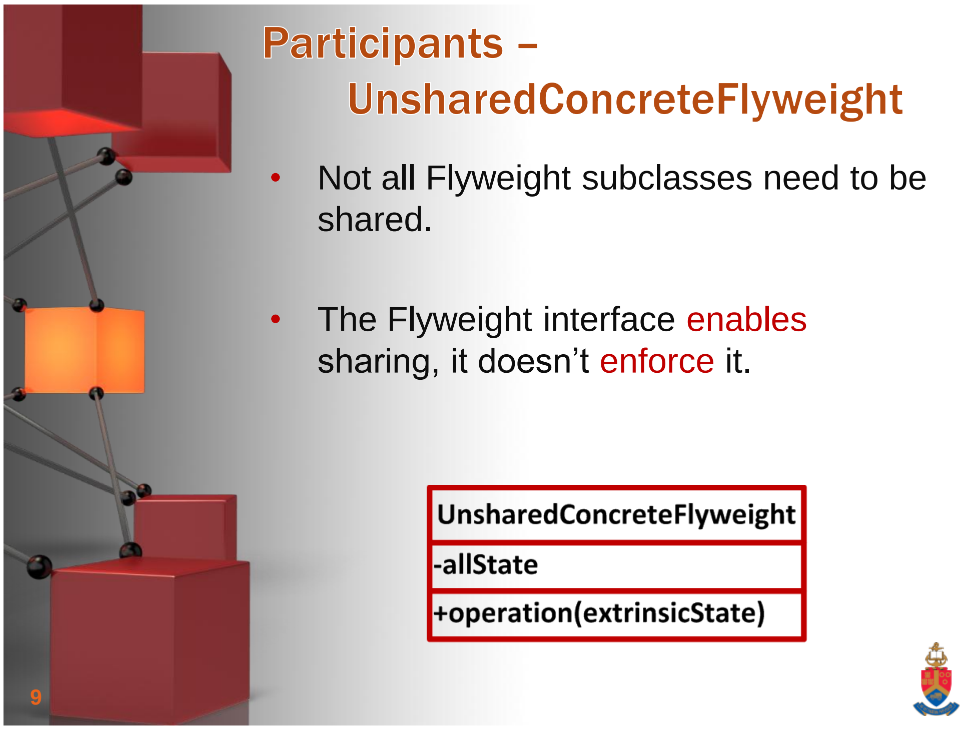


Participants – ConcreteFlyweight

- Implements the Flyweight interface.
- Adds storage for an intrinsic state.
- Must be sharable, therefore must be independent from the object's context.

ConcreteFlyweight
-intrinsicState
+operation(extrinsicState)

8

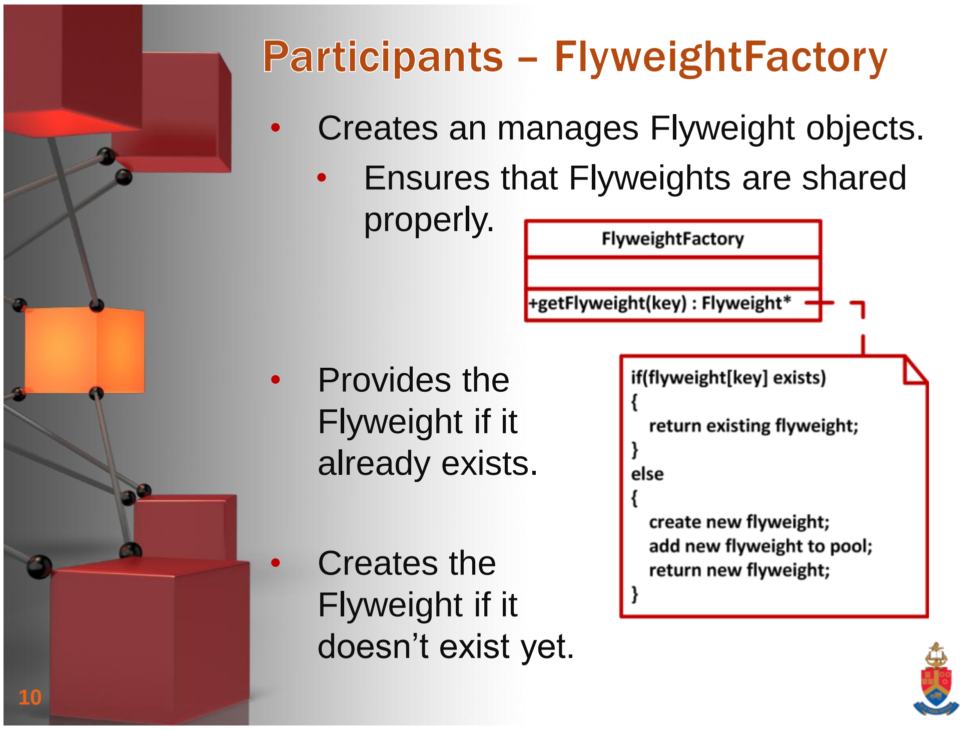


Participants – UnsharedConcreteFlyweight

- Not all Flyweight subclasses need to be shared.
- The Flyweight interface enables sharing, it doesn't enforce it.

UnsharedConcreteFlyweight
-allState
+operation(extrinsicState)

9



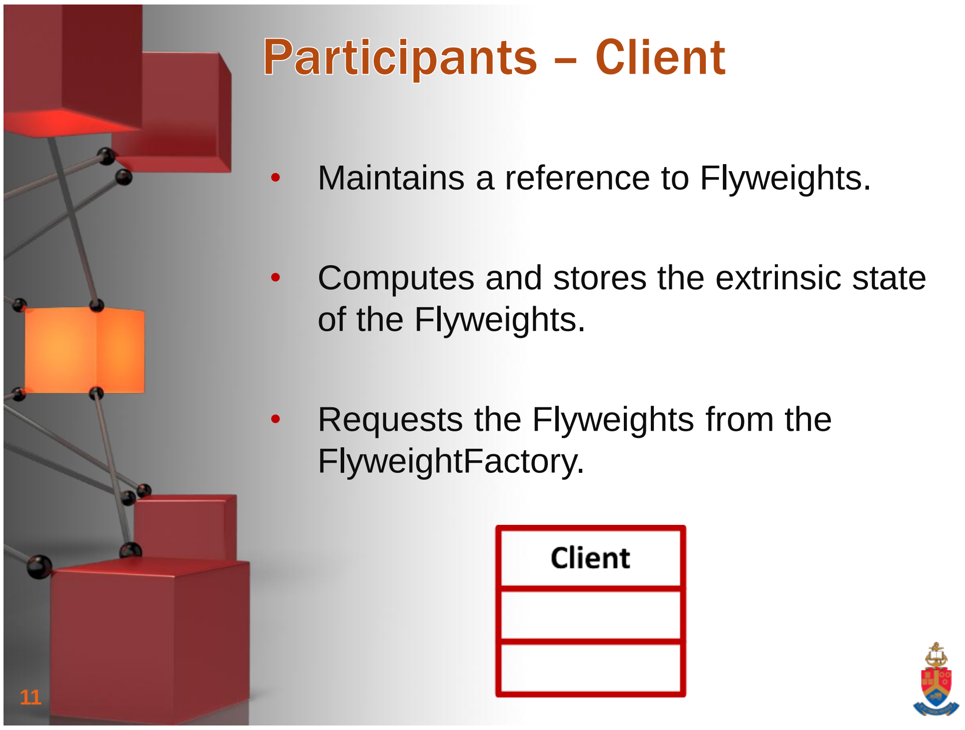
Participants – FlyweightFactory

- Creates an manages Flyweight objects.
  - Ensures that Flyweights are shared properly.
- Provides the Flyweight if it already exists.
- Creates the Flyweight if it doesn't exist yet.

FlyweightFactory
+getFlyweight(key) : Flyweight*

```
if(flyweight[key] exists)
{
   return existing flyweight;
}
else
{
   create new flyweight;
   add new flyweight to pool;
   return new flyweight;
}
```

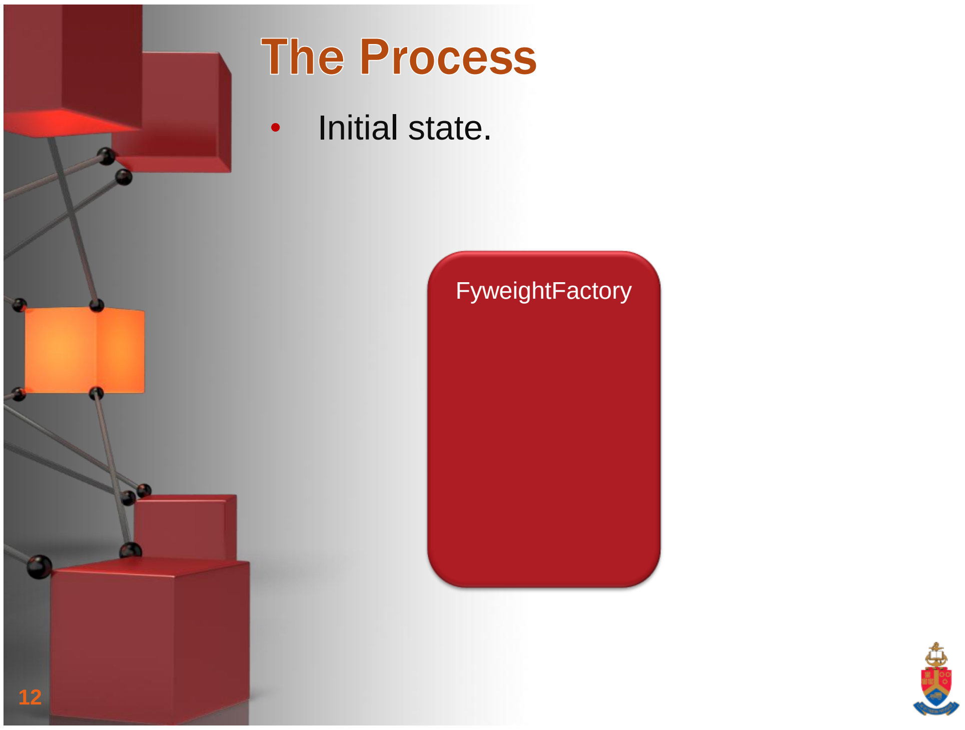
10



Participants – Client

- Maintains a reference to Flyweights.
- Computes and stores the extrinsic state of the Flyweights.
- Requests the Flyweights from the FlyweightFactory.

11

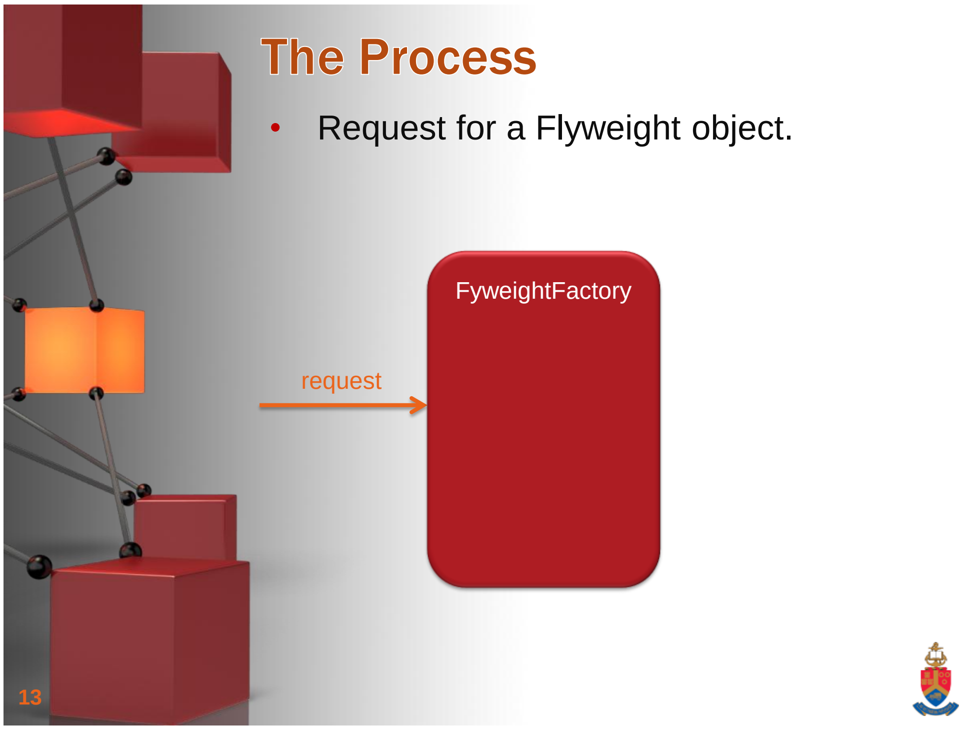


The Process

- Initial state.

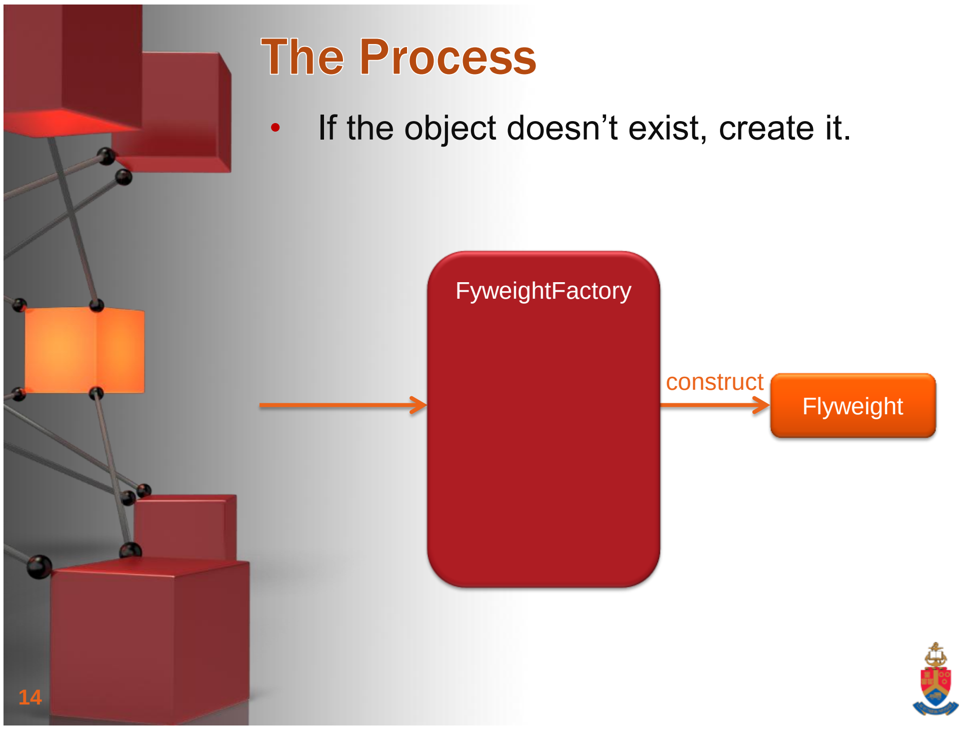
FyweightFactory

12



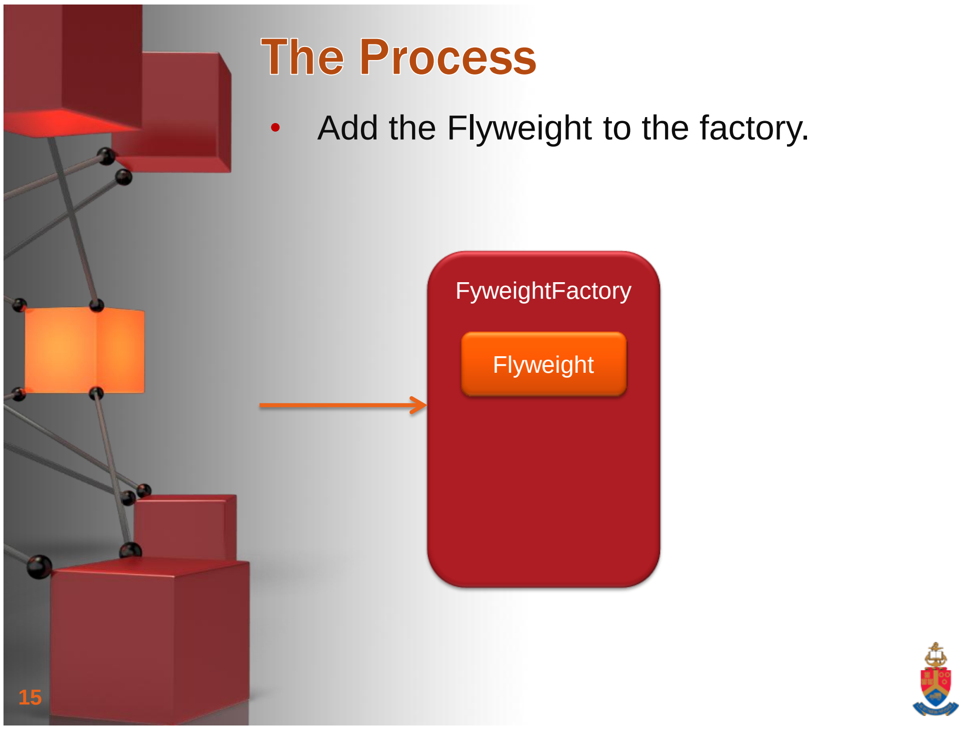
The Process

- Request for a Flyweight object.



The Process

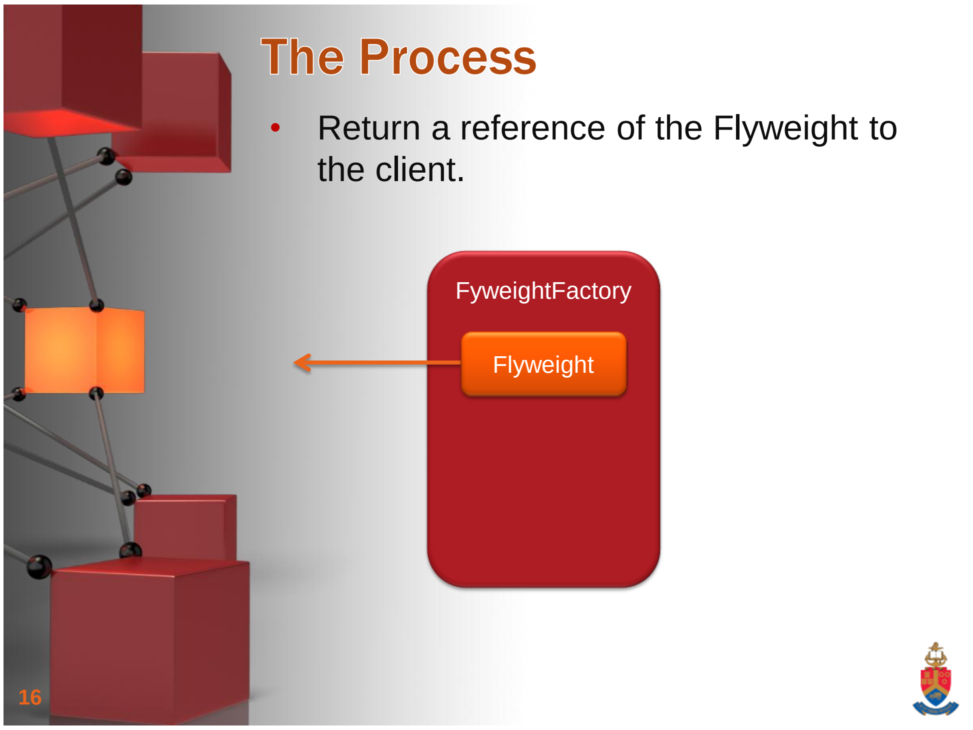
- If the object doesn’t exist, create it.



The Process

- Add the Flyweight to the factory.

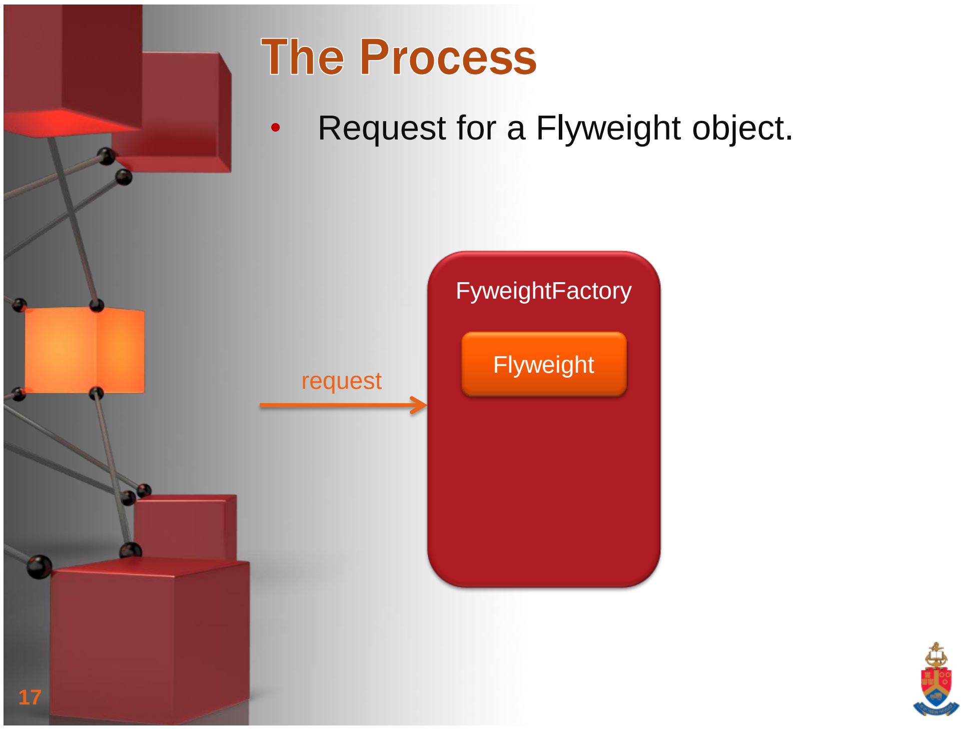
FyweightFactory
Flyweight
15



The Process

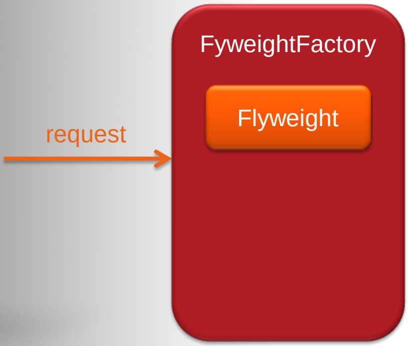
- Return a reference of the Flyweight to the client.

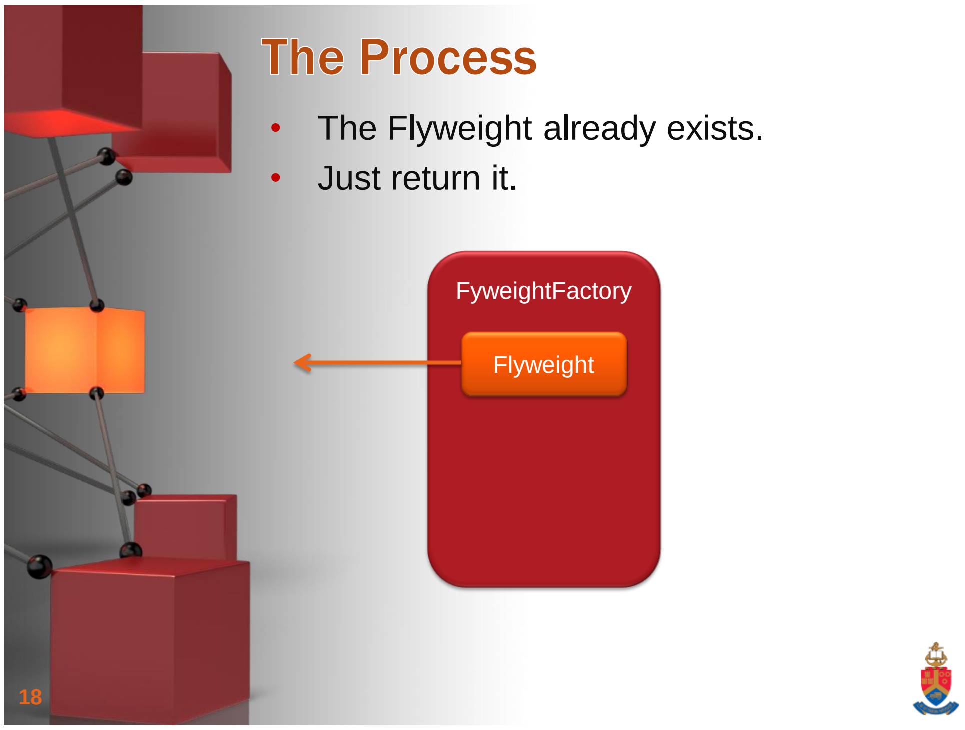
FyweightFactory
Flyweight
16



The Process

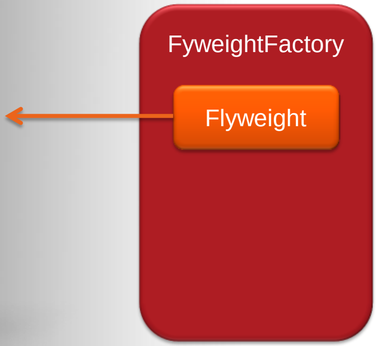
- Request for a Flyweight object.

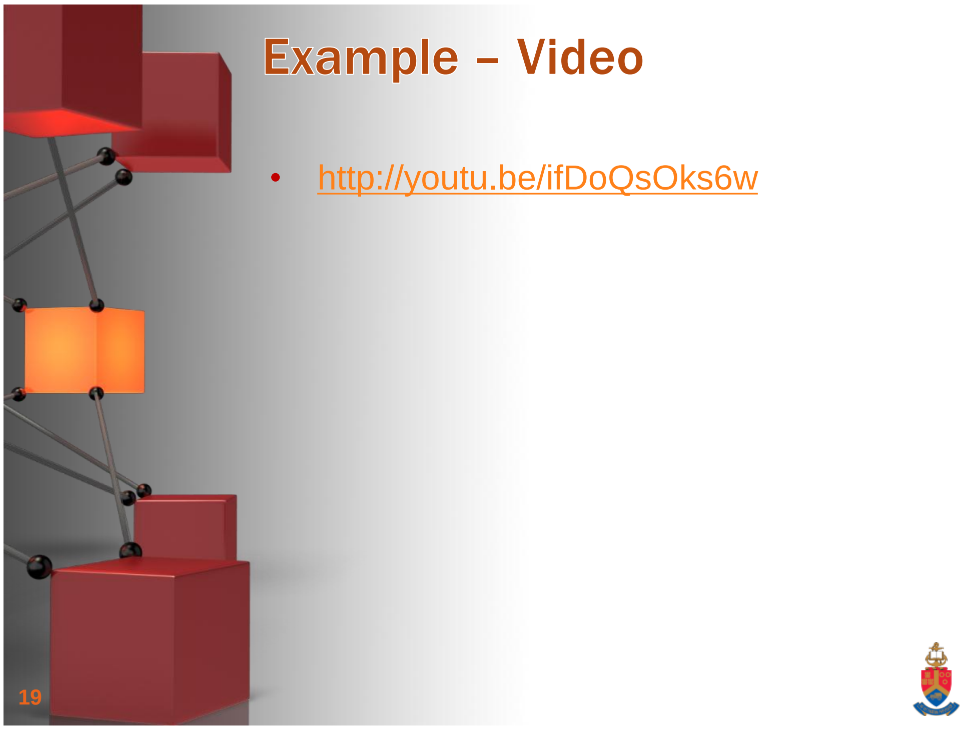




The Process

- The Flyweight already exists.
- Just return it.

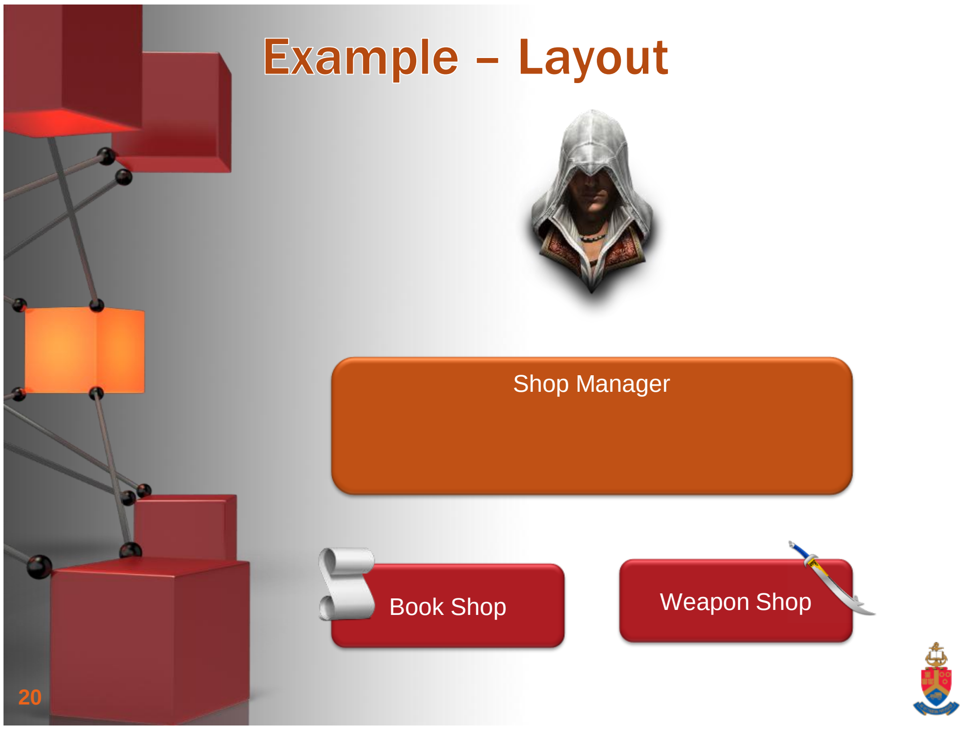




Example – Video

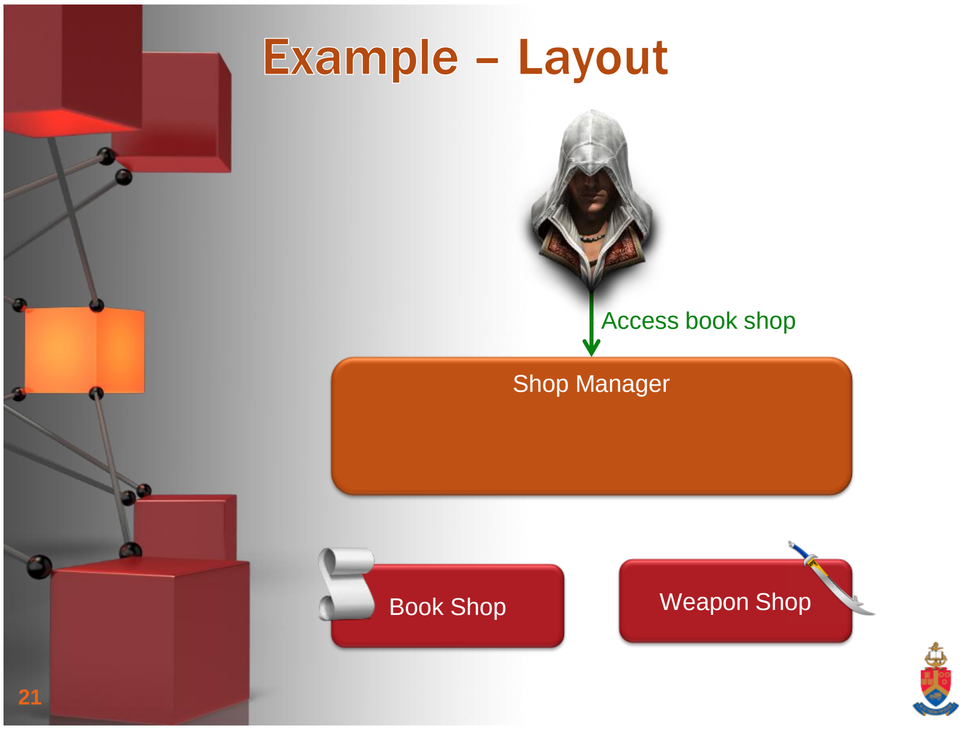
- http://youtu.be/ifDoQsOks6w

19

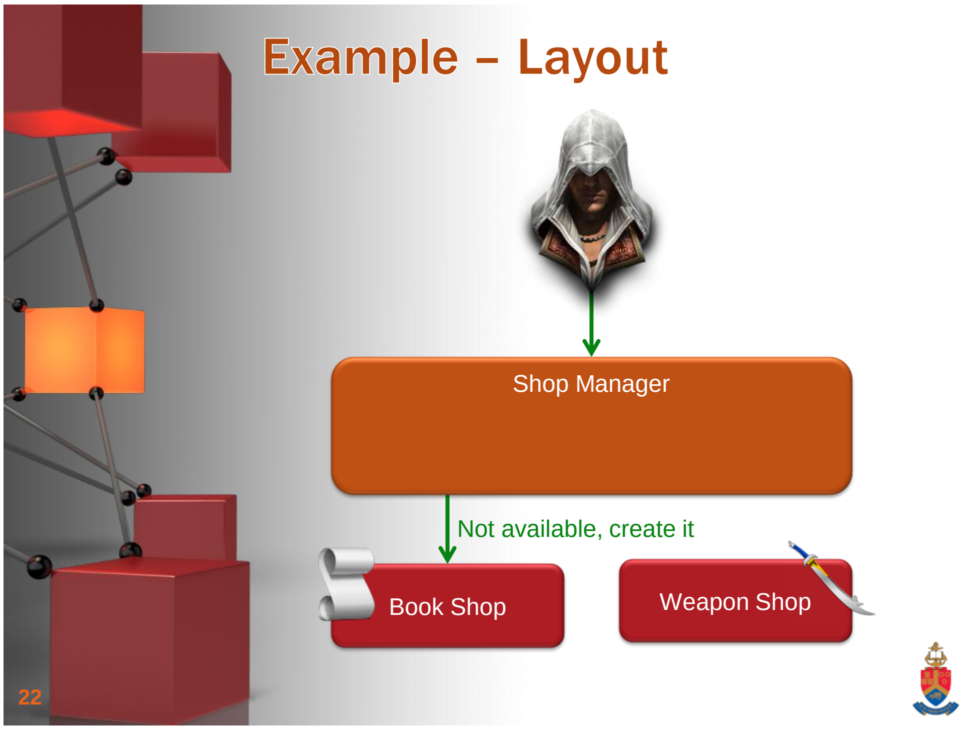


Example – Layout

20

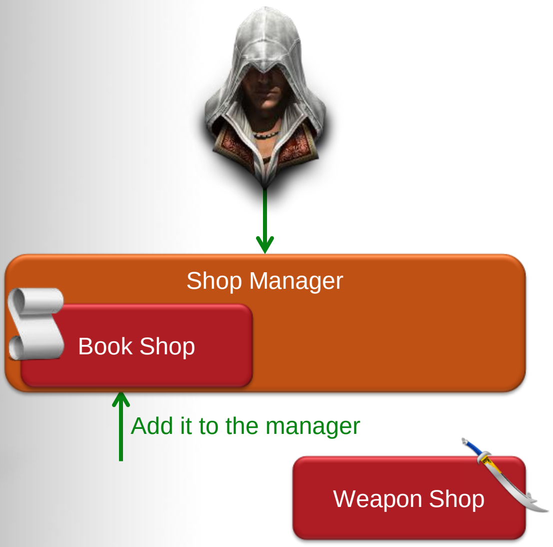


Example – Layout



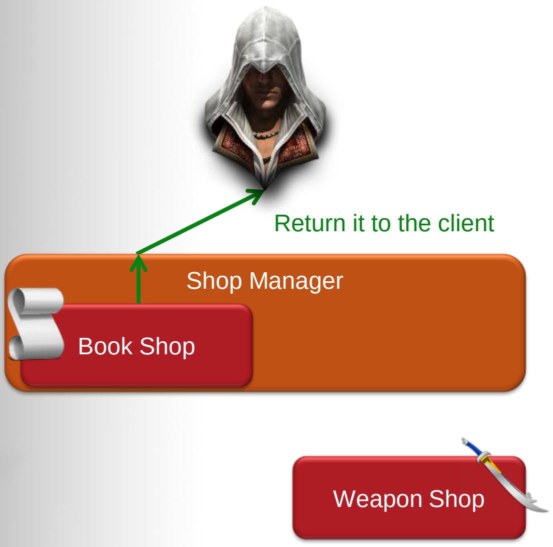
Example – Layout

Example – Layout

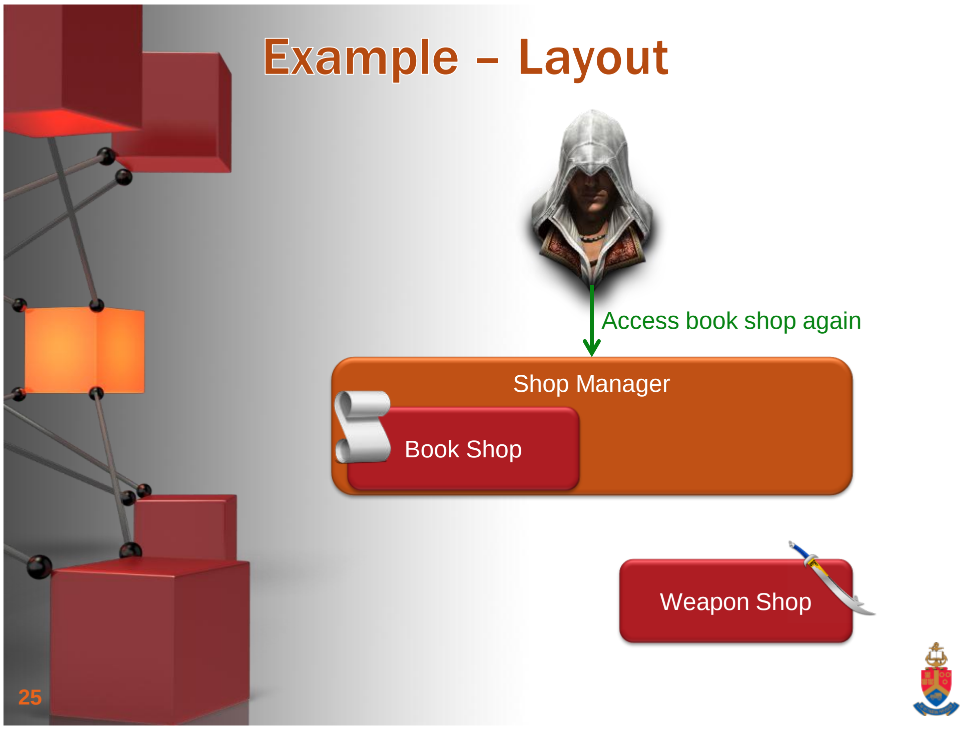


23

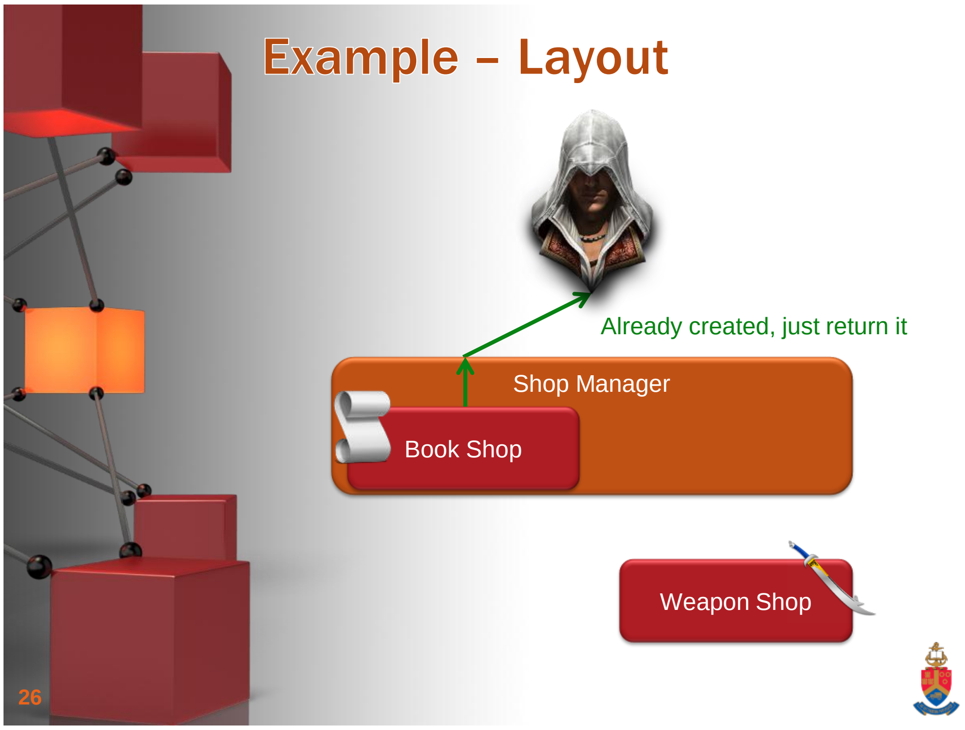
Example – Layout



24

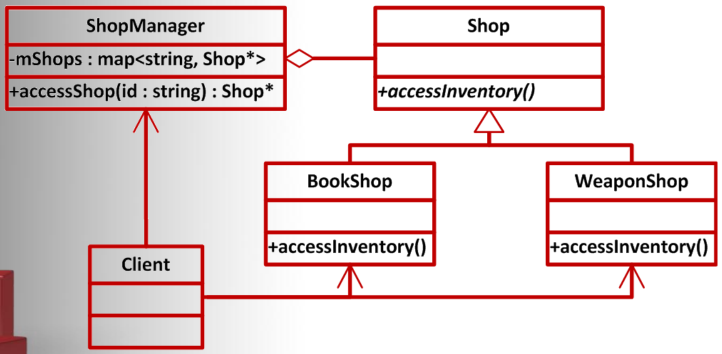


Example – Layout



Example – Layout

Example – UML

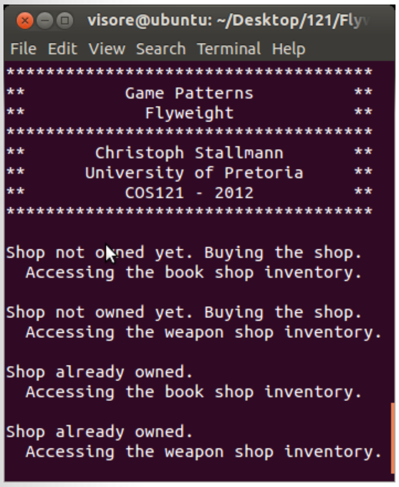


27

Example – Code

28

Example – Output

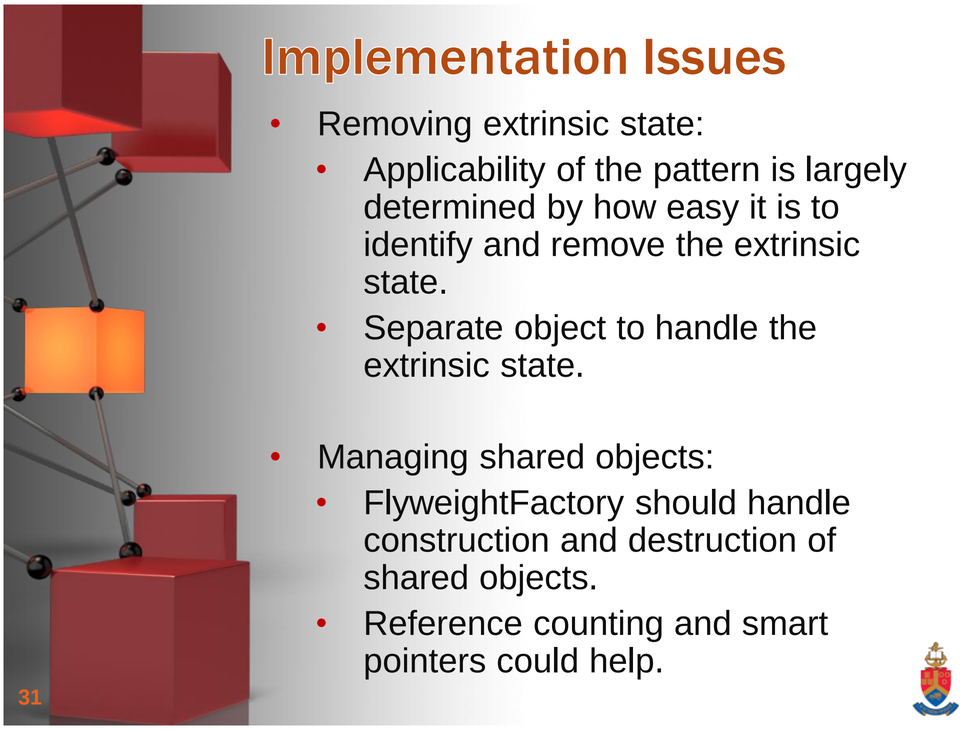


visore@ubuntu: ~/Desktop/121/Fly

```
************************************
**          Game Patterns         **
**            Flyweight           **
************************************
**       Christoph Stallmann      **
**      University of Pretoria    **
**          COS121 - 2012         **
************************************

Shop not owned yet. Buying the shop.
  Accessing the book shop inventory.

Shop not owned yet. Buying the shop.
  Accessing the weapon shop inventory.

Shop already owned.
  Accessing the book shop inventory.

Shop already owned.
  Accessing the weapon shop inventory.
```

Improvements Achieved

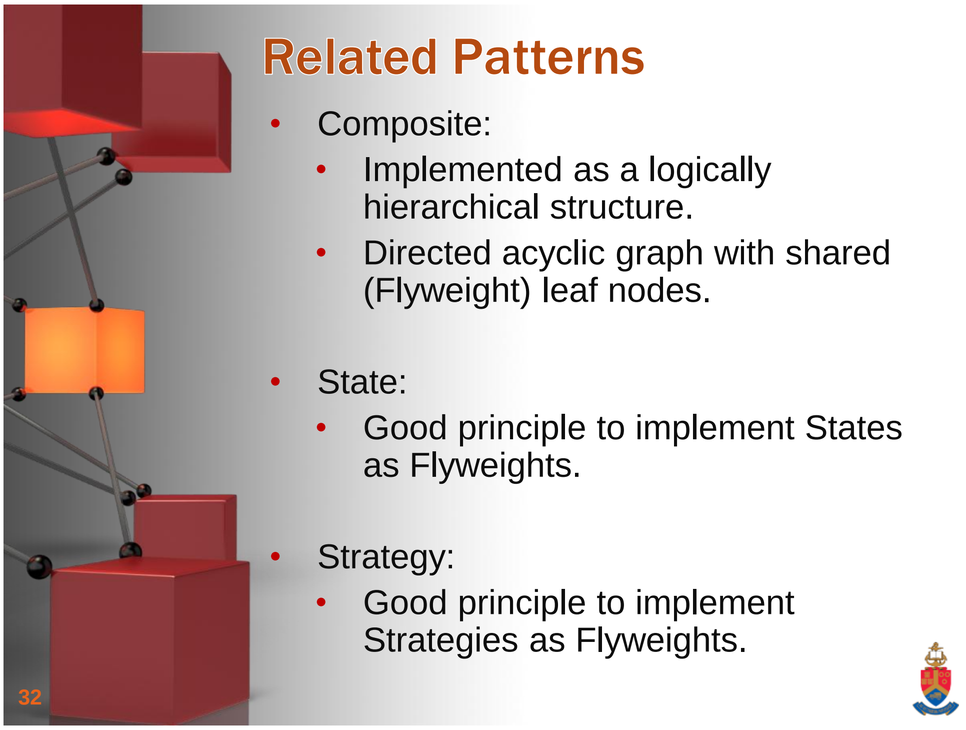
- Central access control to a collection of Flyweights.
- Only create objects when they are needed.
  - Reduce number of shared objects.
- Easily share objects across a system.



Implementation Issues

- Removing extrinsic state:
  - Applicability of the pattern is largely determined by how easy it is to identify and remove the extrinsic state.
  - Separate object to handle the extrinsic state.
- Managing shared objects:
  - FlyweightFactory should handle construction and destruction of shared objects.
  - Reference counting and smart pointers could help.

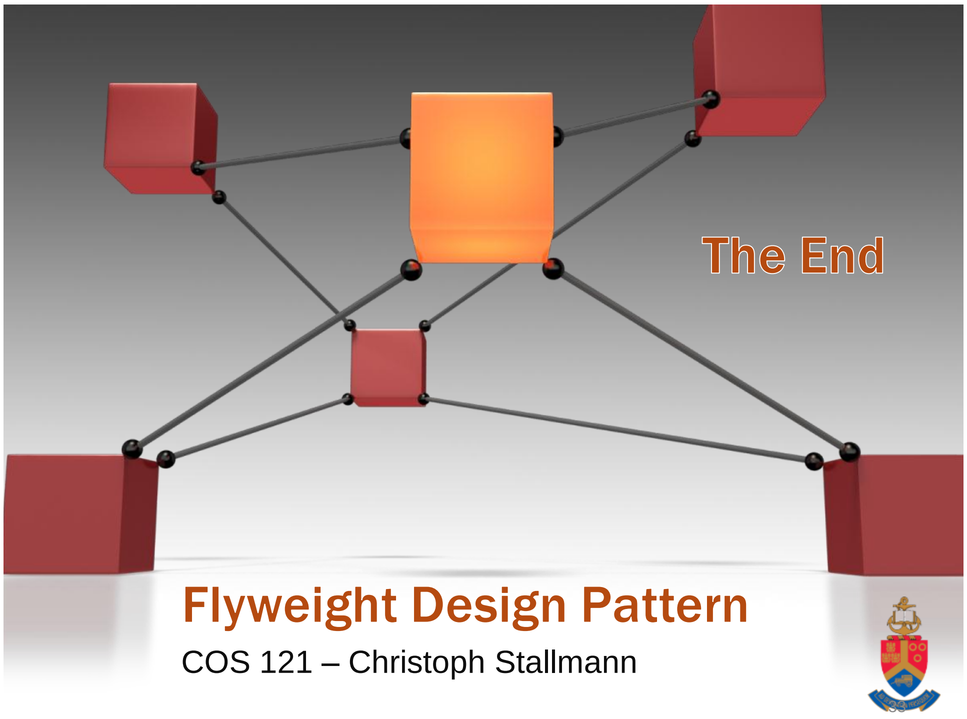
31



Related Patterns

- Composite:
  - Implemented as a logically hierarchical structure.
  - Directed acyclic graph with shared (Flyweight) leaf nodes.
- State:
  - Good principle to implement States as Flyweights.
- Strategy:
  - Good principle to implement Strategies as Flyweights.

32



The End

Flyweight Design Pattern

COS 121 – Christoph Stallmann