Design Patterns

MSc in Communications Software

Produced by

Eamonn de Leastar (edeleastar@wit.ie)

Department of Computing, Maths & Physics
Waterford Institute of Technology

http://www.wit.ie

http://elearning.wit.ie

Waterford Institute *of* Technology
INSTITIÚID TEICNEOLAÍOCHTA PHORT LÁIRGE

eLearning support unit

Design Patterns Categories

The Organisation of Pattern Catalogues

Organising a Catalogue

- Design patterns vary in their granularity and level of abstraction.
- Patterns can be classified into related families
- This classification can helps in learning the patterns & can direct efforts to find new patterns

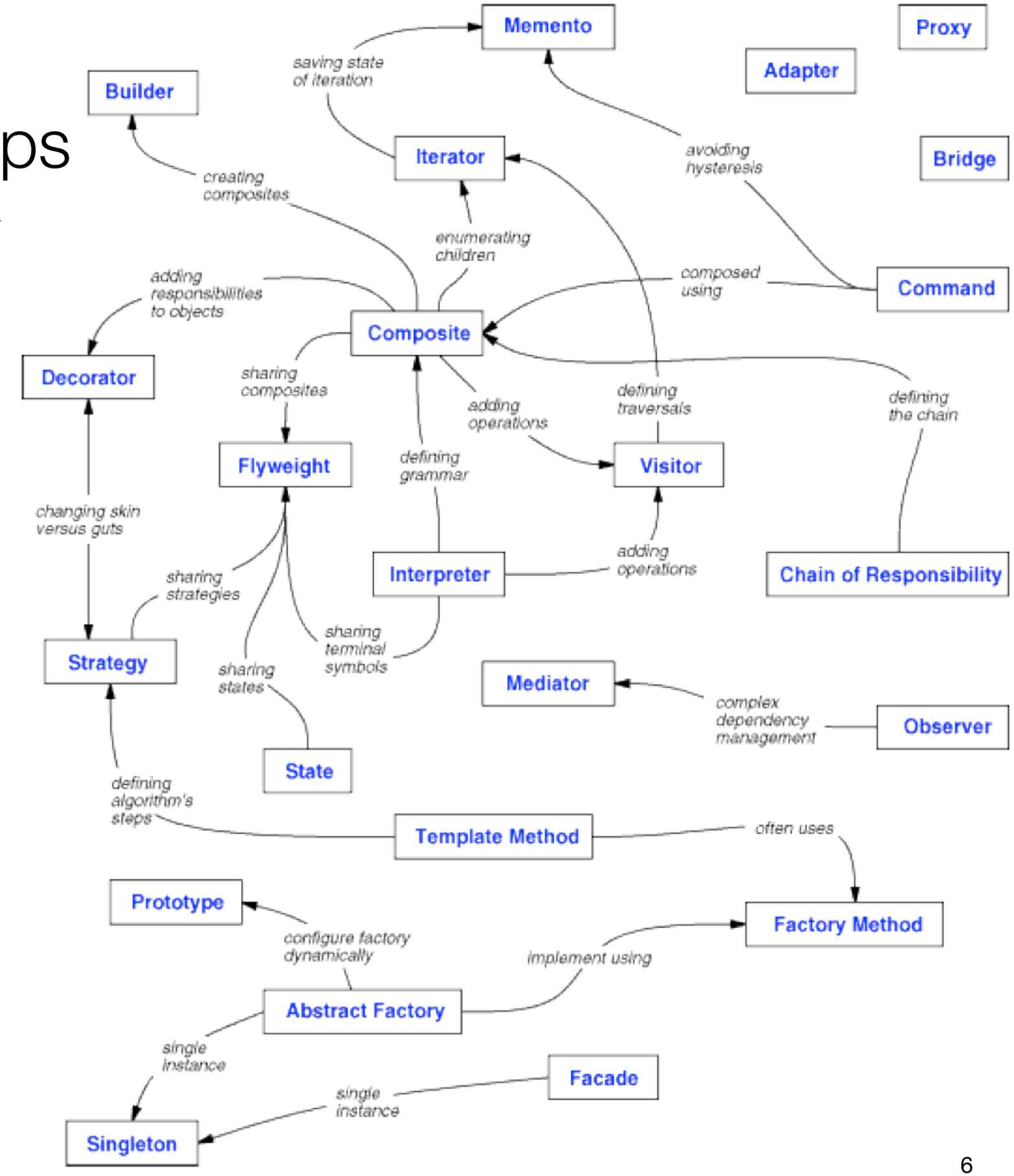
GoF Categorisation

		Purpose		
		Creational	**Structural**	**Behavioral**
Scope	**Class**	Factory Method	Adapter	Interpreter **Template Method**
	Object	Abstract Factory Builder Prototype **Singleton**	Adapter Bridge **Composite** Decorator Façade Proxy	Chain of Responsibility **Command** Iterator Mediator Memento Flyweight **Observer** State **Strategy** Visitor

Categorisation Criteria

- Purpose - reflects pattern's intent:
  - Creational: the process of object creation
  - Structural: the composition of classes or objects
  - Behavioural: ways classes/objects interact and distribute responsibility
- Scope - whether pattern applies primarily to classes or objects:
  - Class: deal with relationships between classes and their subclasses. These relationships are established through inheritance
  - Object: deal with object relationships, which can be changed at run-time and are more dynamic.
- Almost all patterns use inheritance to some extent- class patterns are those that focus on class relationships.
- Most patterns are in the Object scope (reflecting the favouring of composition over inheritance)

Pattern Relationships



Creational Patterns (1)

- Creational design patterns abstract the instantiation process
  - System becomes independent of how its objects are created, composed, & represented.
  - A class creational pattern uses inheritance to vary the class that's instantiated
  - A object creational pattern will delegate instantiation to another object.
- Creational patterns important as systems evolve to depend more on object composition than implementation inheritance
  - Emphasis shifts away from hard-coding a fixed set of behaviours toward defining a smaller set of fundamental behaviours that can be composed into any number of more complex ones.
  - Thus creating objects with particular behaviours requires more than simply instantiating a class

Creational Patterns (2)

- Two recurring themes:
  - encapsulate knowledge about which concrete classes the system uses.
  - hide how instances of these classes are created and put together.
- Creational patterns provide flexibility:
  - in what gets created
  - who creates it
  - how it gets created
  - and when
- A system can be configured with objects that vary widely in structure and functionality.
- Configuration can be static (that is, specified at compile-time) or dynamic (at run-time).

Structural Patterns

- Concerned with how classes and objects are composed to form larger structures
  - Class patterns use inheritance to compose interfaces or implementations
  - Object patterns describe ways to compose objects to realize new functionality.
- The added flexibility of object composition comes from the ability to change the composition at run-time, which is impossible with static class composition.

Behavioural Patterns

- Concerned with algorithms and the assignment of responsibilities between objects.

- Describe not just patterns of objects or classes but also the patterns of communication between them.

- They characterize complex control flow that may be difficult to follow at run-time.

- Shift focus away from flow of control to permit concentration on the way objects are interconnected

The GoF Patterns (1)

Abstract Factory

- Provide an interface for creating families of related or dependent objects without specifying their concrete classes.

Adapter

- Convert the interface of a class into another interface clients expect. Adapter lets classes work together that couldn't otherwise because of incompatible interfaces.

Bridge

- Decouple an abstraction from its implementation so that the two can vary independently.

Builder

- Separate the construction of a complex object from its representation so that the same construction process can create different representations.

Chain of Responsibility

- Avoid coupling the sender of a request to its receiver by giving more than one object a chance to handle the request. Chain the receiving objects and pass the request along the chain until an object handles it.

Command

- Encapsulate a request as an object, thereby letting you parameterize clients with different requests, queue or log requests, and support undoable operations.

The GoF Patterns (2)

Composite

- Compose objects into tree structures to represent part-whole hierarchies. Composite lets clients treat individual objects and compositions of objects uniformly.

Decorator

- Attach additional responsibilities to an object dynamically. Decorators provide a flexible alternative to inheritance for extending functionality.

Facade

- Provide a unified interface to a set of interfaces in a subsystem. Facade defines a higher-level interface that makes the subsystem easier to use.

Factory Method

- Define an interface for creating an object, but let subclasses decide which class to instantiate. Factory Method lets a class defer instantiation to subclasses.

Flyweight

- Use sharing to support large numbers of fine-grained objects efficiently.

Interpreter

- Given a language, define a representation for its grammar along with an interpreter that uses the representation to interpret sentences in the language.

The GoF Patterns (3)

Iterator

- Provide a way to access the elements of an aggregate object sequentially without exposing its underlying representation.

Mediator

- Define an object that encapsulates how a set of objects interact. Mediator promotes loose coupling by keeping objects from referring to each other explicitly, and it lets you vary their interaction independently.

Memento

- Without violating encapsulation, capture and externalize an object's internal state so that the object can be restored to this state later.

Observer

- Define a one-to-many dependency between objects so that when one object changes state, all its dependents are notified and updated automatically.

Prototype

- Specify the kinds of objects to create using a prototypical instance, and create new objects by copying this prototype.

Proxy

- Provide a surrogate or placeholder for another object to control access to it.

The GoF Patterns (4)

Singleton

- Ensure a class only has one instance, and provide a global point of access to it.

State

- Allow an object to alter its behaviour when its internal state changes. The object will appear to change its class.

Strategy

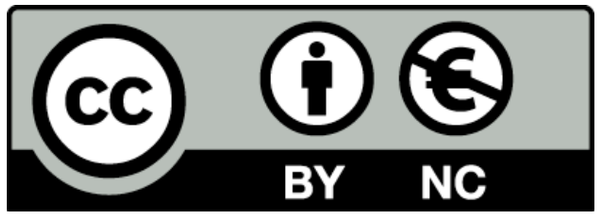
- Define a family of algorithms, encapsulate each one, and make them interchangeable. Strategy lets the algorithm vary independently from clients that use it.

Template Method

- Define the skeleton of an algorithm in an operation, deferring some steps to subclasses. Template Method lets subclasses redefine certain steps of an algorithm without changing the algorithm's structure.

Visitor

- Represent an operation to be performed on the elements of an object structure. Visitor lets you define a new operation without changing the classes of the elements on which it operates.



Waterford Institute *of* Technology
INSTITIÚID TEICNEOLAÍOCHTA PHORT LÁIRGE

eLearning support unit