Design Pattern

27 mars 2020

Who

- Dev C#
- Arolla / Fnac Darty
- Not a prof

Evaluations

- Exposé x 2
- 1:1 code analysis

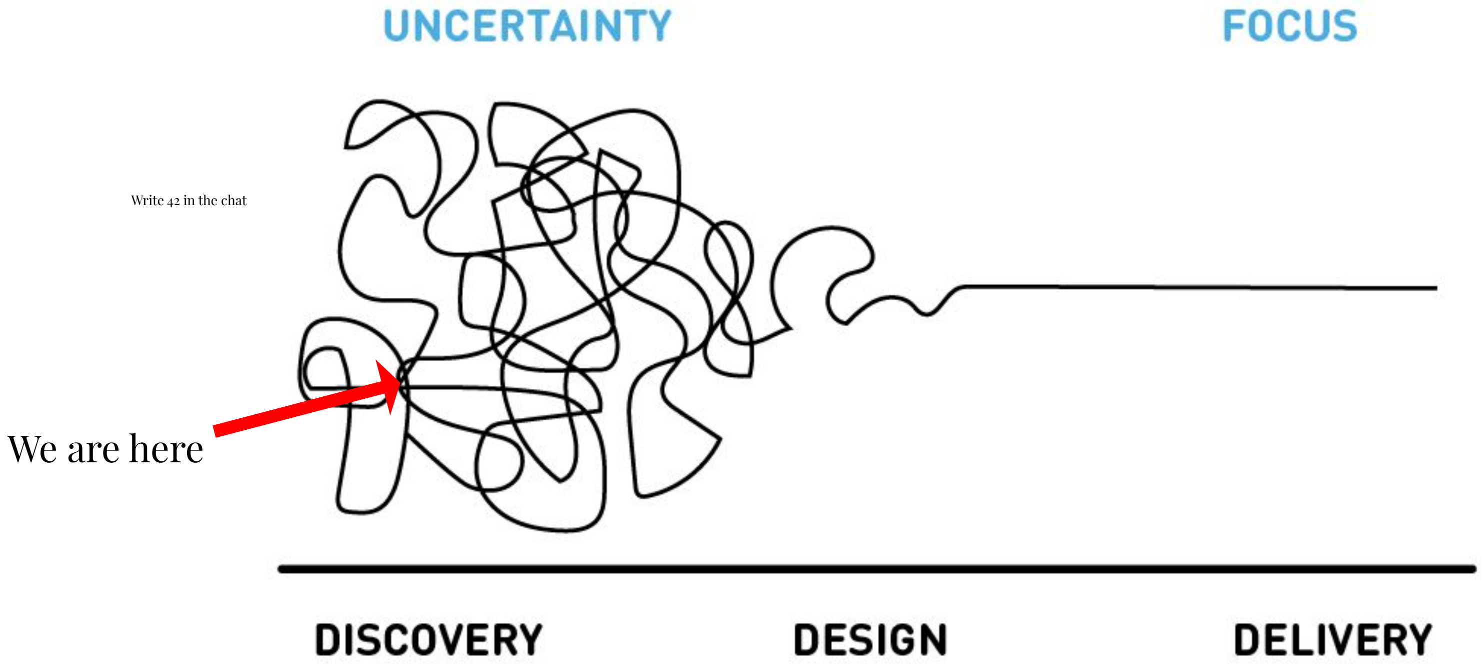
Projection 30h

27/03 - 6h	Introduction + test
17/04 - 12h	Exposé x 7
29/05 - 18h	Exposé x 7
19/06 - 24h	Exposé x 7
03/07 - 30h	1:1 - Apéro

Let’s meet

Spoiler Alert

Write 42 in the chat



Design Pattern - l’objectif

OBJECTIF : Apprendre les Design Pattern et Code Smell

Know when it stinks… ➻ Design issues !

https://sourcemaking.com/refactoring/smells

Know a way to fix it (most of the time for a fairly acceptable trade off)

https://sourcemaking.com/design_patterns

Exposé

Part 1 : Feedback on the collaboration
» Why did you choose this Smell&DesignPattern ?
» How do you work together ?

Part 2 : Smell
» Explain the smell be creative !
» It sticks !

Part 3 : Design Pattern
» Explain the Design Pattern be creative !
» A pattern should explain why that solution is good in the pattern's contexts.

Quizz : whats wrong here ?

```
public class Customer {

    public int Id { get; set; }

    public string Name { get; set; }

    public string FirstName { get; set; }

    public string Email { get; set; }

    public string Address { get; set; }

    public string ZipCode { get; set; }

 }
```

Smell : primitive obsession

```
public class Customer {

    public int Id { get; set; }

    public string Name { get; set; }

    public string FirstName { get; set; }

    public string Email { get; set; }

    public string Address { get; set; }

    public string ZipCode { get; set; }

  }
```

```
public class Customer {

    public int Id { get; set; }

    public Name  Name { get; set; }

    public Email Email { get; set; }

    public Address Address { get; set; }

  }
```

Smell : primitive obsession

```
public class Customer {

    public int Id { get; set; }

    public string Name { get; set; }

    public string FirstName { get; set; }

    public string Email { get; set; }

    public string Address { get; set; }

    public string ZipCode { get; set; }
```

Write boom in the chat

```
}
```

```
public class Customer {

    public int Id { get; set; }

    public Name  Name { get; set; }

    public Email Email { get; set; }

    public Address Address { get; set; }

}
```

Introduce "**Value Object**" Pattern

Design Pattern : Documentation

- **Pattern Name and Classification:** A descriptive and unique name that helps in identifying and referring to the pattern.
- **Intent:** A description of the goal behind the pattern and the reason for using it.
- **Also Known As:** Other names for the pattern.
- **Motivation (Forces):** A scenario consisting of a problem and a context in which this pattern can be used.
- **Applicability:** Situations in which this pattern is usable; the context for the pattern.
- **Structure:** A graphical representation of the pattern. Class diagrams and Interaction diagrams may be used for this purpose.
- **Participants:** A listing of the classes and objects used in the if you see this write miaou pattern and their roles in the design.
- **Collaboration:** A description of how classes and objects used in the pattern interact with each other.
- **Consequences:** A description of the results, side effects, and trade offs caused by using the pattern.
- **Implementation:** A description of an implementation of the pattern; the solution part of the pattern.
- **Sample Code:** An illustration of how the pattern can be used in a programming language.
- **Known Uses:** Examples of real usages of the pattern.
- **Related Patterns:**

https://en.wikipedia.org/wiki/Software_design_pattern

Design Pattern : Value Object

- **Pattern Name and Classification:** Value Object
- **Intent:** The goal is to have an immutable object easy to reason with pure calculation and no identity.
- **Also Known As:** -
- **Motivation (Forces):** A scenario consisting of a problem and a context in which this pattern can be used.
- **Applicability:** When you spot some "primitive obsession" or when modeling.
- **Structure:** https://github.com/iluwatar/java-design-patterns/tree/master/value-object
- **Participants:** -
- **Collaboration:** Always a part of an Entity can't live one it's own
- **Consequences:** Help to stay DRY, clarify intent of the code. The trade off is wrapping all the primitive could lead to large number of class.
- **Implementation:** Override equals and hashCode method, often the toString() is overridden as well. A Static Factory method named "ValueOf" is recommended to produce the Value Object.
- **Sample Code:** DateRange : https://dzone.com/articles/value-objects
- **Known Uses:** Used for Range, Address, Money, ZipCode, Email, Point,...
- **Related Patterns:** DataTransferObject, Entity

Further reading :
https://docs.microsoft.com/en-us/dotnet/architecture/microservices/microservice-ddd-cqrs-patterns/implement-value-objects

Design Pattern

1. Creational
2. Structural
3. Behavioral

Define type of Design Pattern - Creational

Ces patrons répondent aux problèmes autour des instances de classe et de la création d’objet. Ils ont tendance à user fortement de l’héritage et le polymorphisme.

Creational design patterns are Factory Method, Abstract Factory, Builder, Singleton, Object Pool and Prototype.

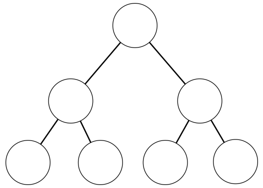
Define type of Design Pattern - Structural

Ces types répondent aux problèmes de communication et d’organisations des classes et objets pour former des structures plus large et offrir de nouvelles fonctionnalitées.

Ils vont nous permettre d’imbriquer nos différents modules ou composants ensemble.

Ils ont tendances à user fortement des interfaces et la composition.

Structural design patterns are Adapter, Bridge, Composite, Decorator, Facade, Flyweight, Private Class Data and Proxy.



Write compote in the chat

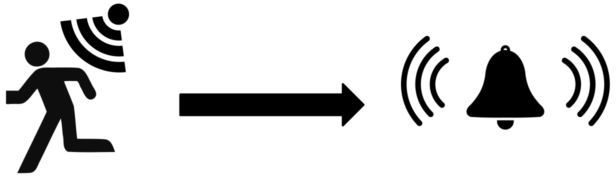
Define type of Design Pattern - Behavioural

Ces design pattern répondent aux problèmes de communication et de responsabilités des classes.

Ils ont tendances à favoriser l’encapsulation et le principe SRP*

*https://www.dotnetdojo.com/applications-solid-srp/

Behavioral patterns are Chain of responsibility, Command, Interpreter, Iterator, Mediator, Memento, Null Object, Observer, State, Strategy, Template method, Visitor



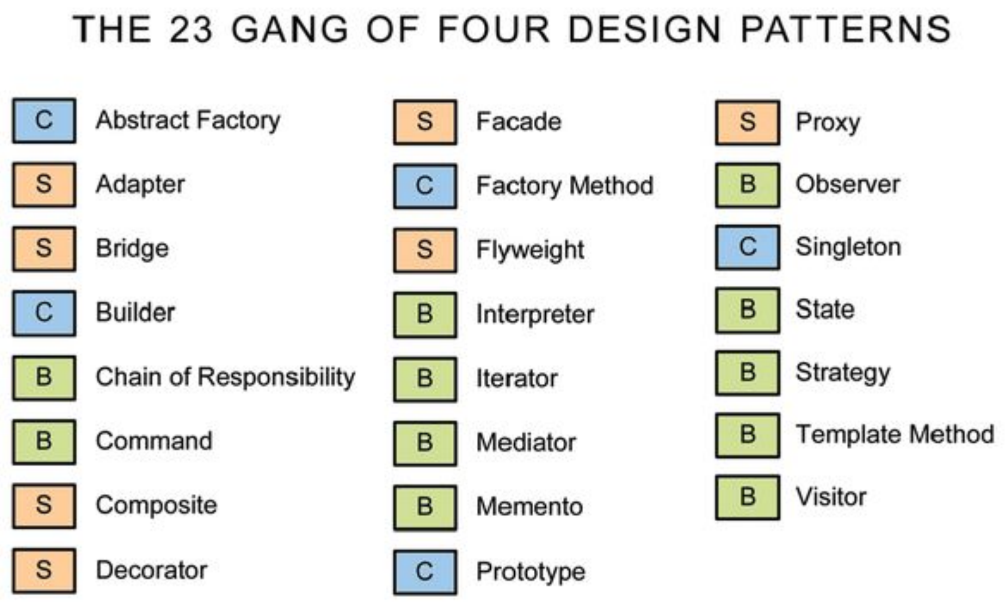
Summary!

Quand tu as un problème de “**Responsabilité**” ↣ BEHAVIOURAL PATTERN

Quand tu as un problème de “**Communication**” ↣ STRUCTURAL PATTERN

Quand tu as un problème “d'Instanciation” #**new** ↣ CREATIONAL PATTERN

Where should we start our journey …



THE 23 GANG OF FOUR DESIGN PATTERNS

C	Abstract Factory	S	Facade	S	Proxy
S	Adapter	C	Factory Method	B	Observer
S	Bridge	S	Flyweight	C	Singleton
C	Builder	B	Interpreter	B	State
B	Chain of Responsibility	B	Iterator	B	Strategy
B	Command	B	Mediator	B	Template Method
S	Composite	B	Memento	B	Visitor
S	Decorator	C	Prototype		

https://www.script-tutorials.com (write let's go)