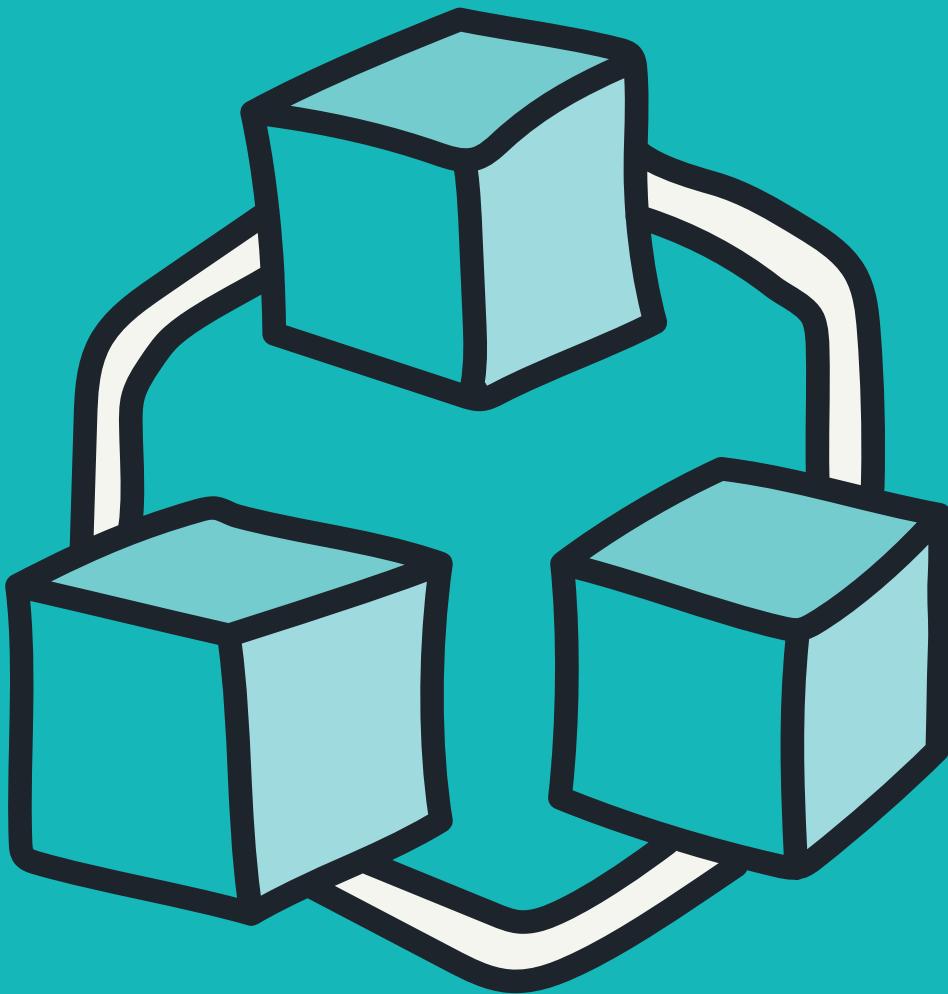


# PADRÕES DE PROJETO

APRENDA OS PADRÕES DE FORMA  
PRÁTICA

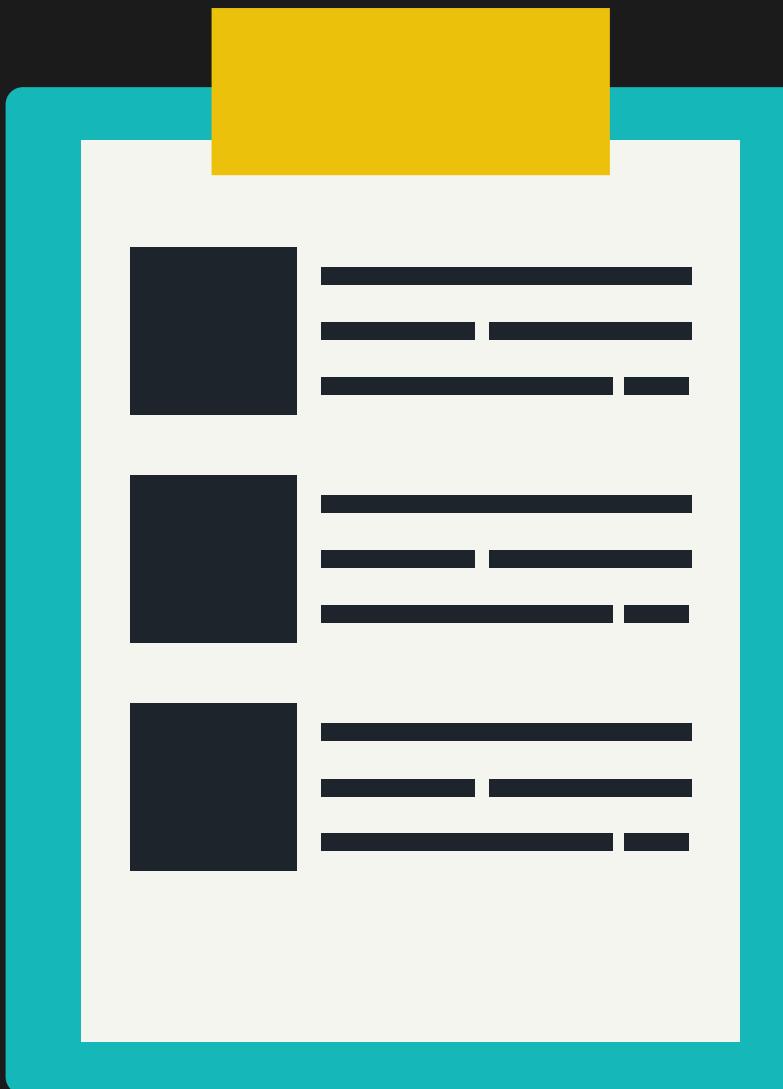
padrões de projeto



# VISÃO GERAL

- INTRODUÇÃO
- PADRÕES DE PROJETOS CRIACIONAIS
- PADRÕES DE PROJETOS ESTRUTURAIS
- PADRÕES DE PROJETOS COMPORTAMENTAIS

# VISÃO GERAL DO CURSO

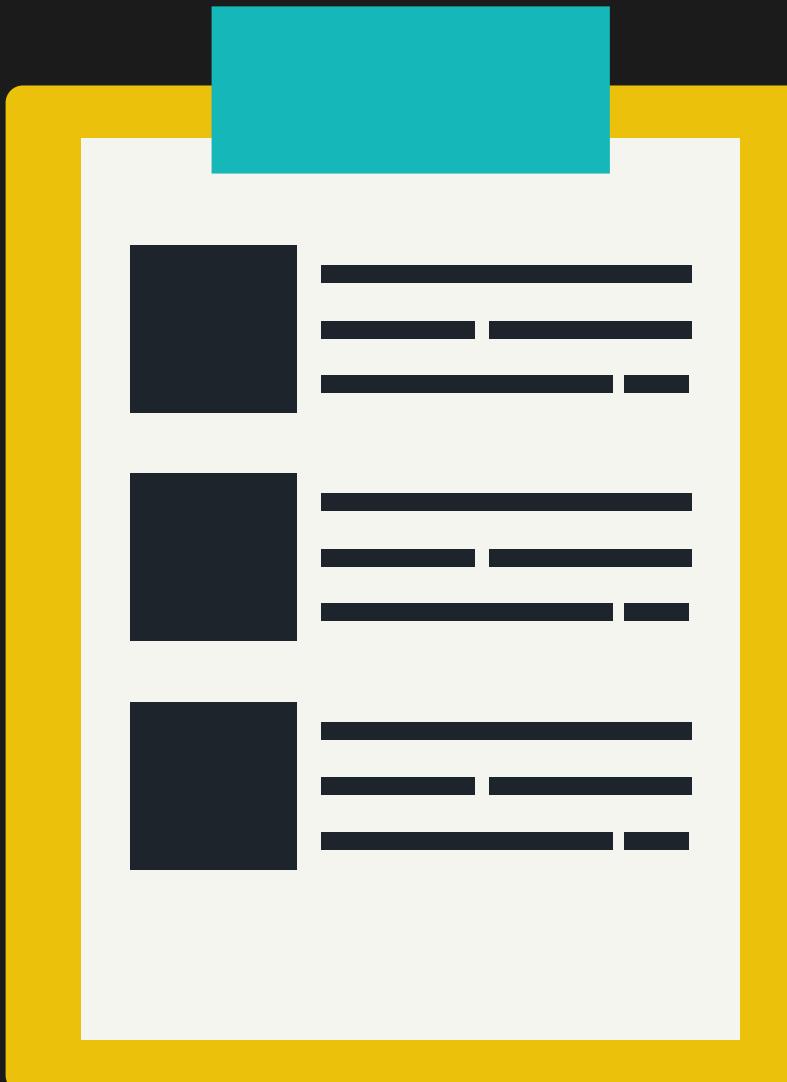


## 01 - INTRODUÇÃO

O que são padrões?  
Porque existem?  
Onde se aplicam?

## 02 - PADRÕES DE PROJETOS CRIACIONAIS

Existem diversos mecanismos para criar um objeto. Ao invés de utilizar diretamente o operador "new", podemos utilizar algum padrão que nos forneça mais flexibilidade no código.

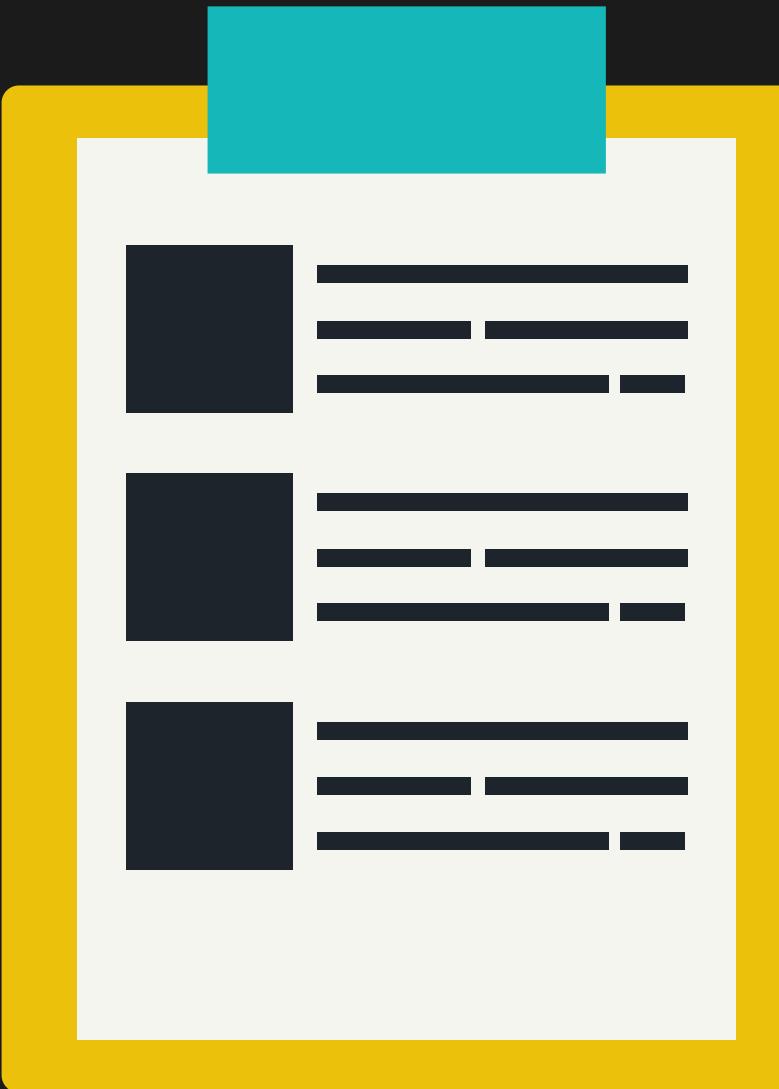


## 03 - PADRÕES DE PROJETOS ESTRUTURAIS

Mostrar como objetos podem se unir em estruturas maiores, porém de forma organizada, facilitando possíveis extensões.

## 04 - PADRÕES DE PROJETOS COMPORTAMENTAIS

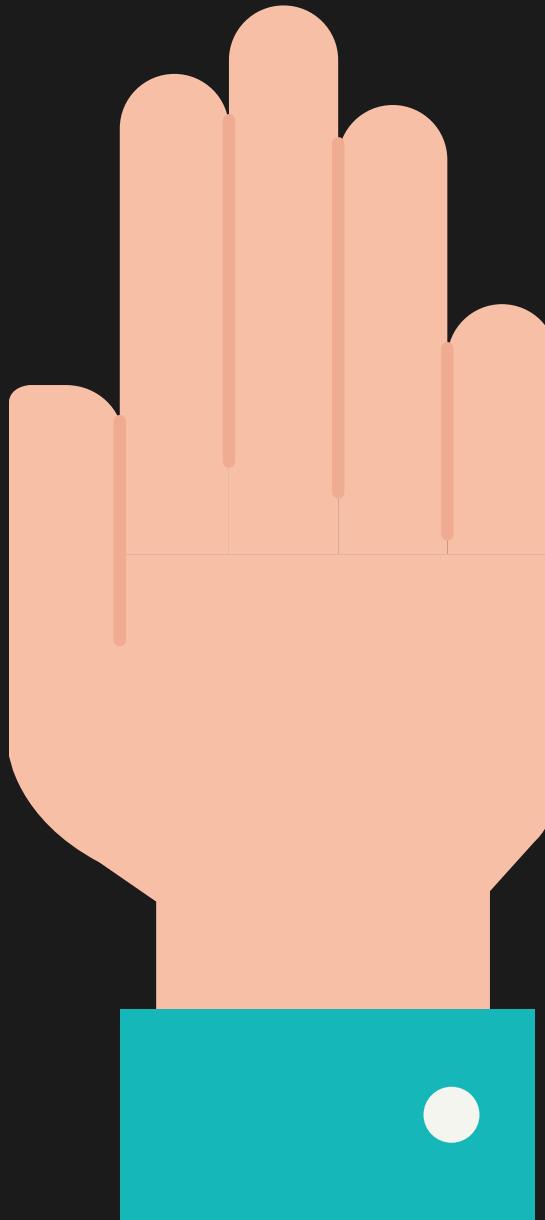
Organizar a forma de comunicação entre os objetos.



## 05 - SOLID

Apresentar cinco princípios para deixar seus códigos mais legíveis, flexíveis e manutêveis.

# RECAPTULANDO AS BOAS PRÁTICAS NO FÓRUM



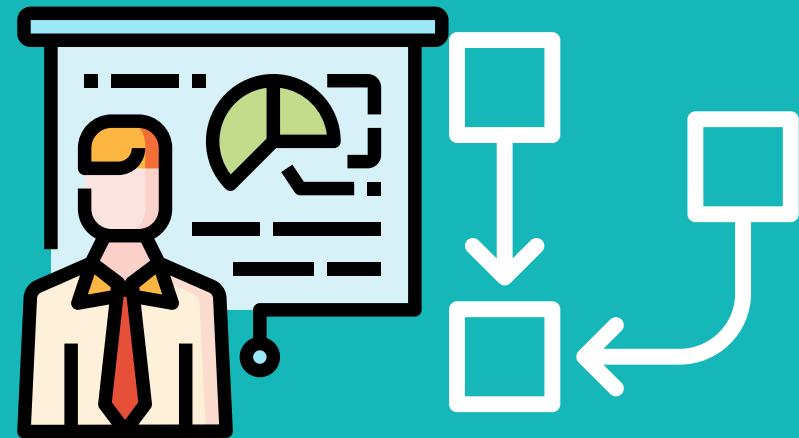
**FORNEÇA TODAS AS INFORMAÇÕES**

**SEJA GENTIL**

**PESQUISE ANTES DE PERGUNTAR**

**AJUDE OUTROS ALUNOS**

# RECURSOS DISPONIBILIZADOS



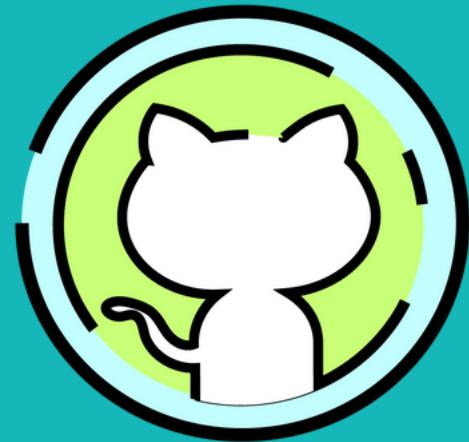
## MATERIAL PARA DOWNLOAD

Todos os slides e diagramas de Classes apresentados no curso estão disponíveis para download.



## FÓRUM DO CURSO

Equipe atenta para buscar tirar as suas dúvidas no fórum da plataforma.



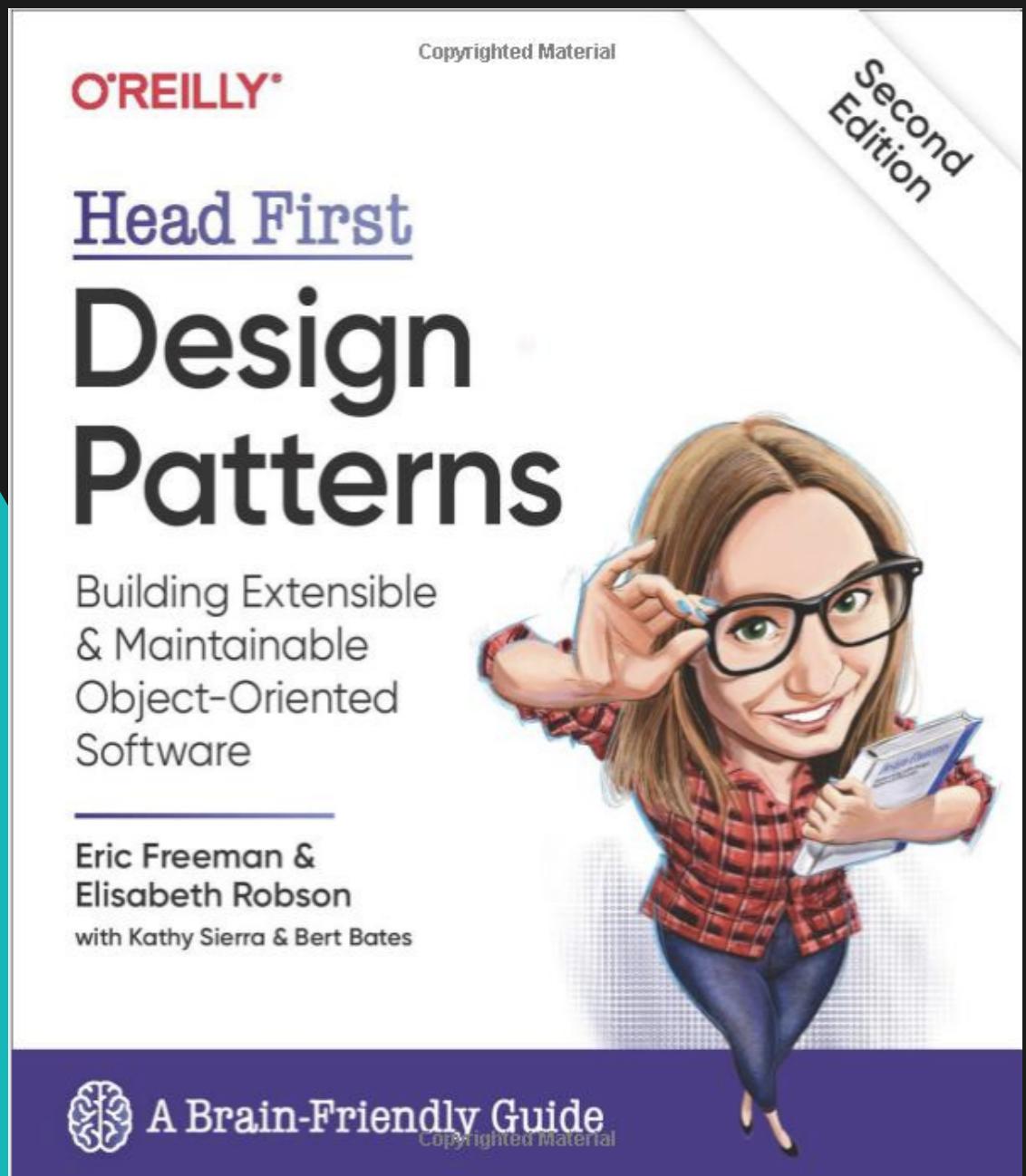
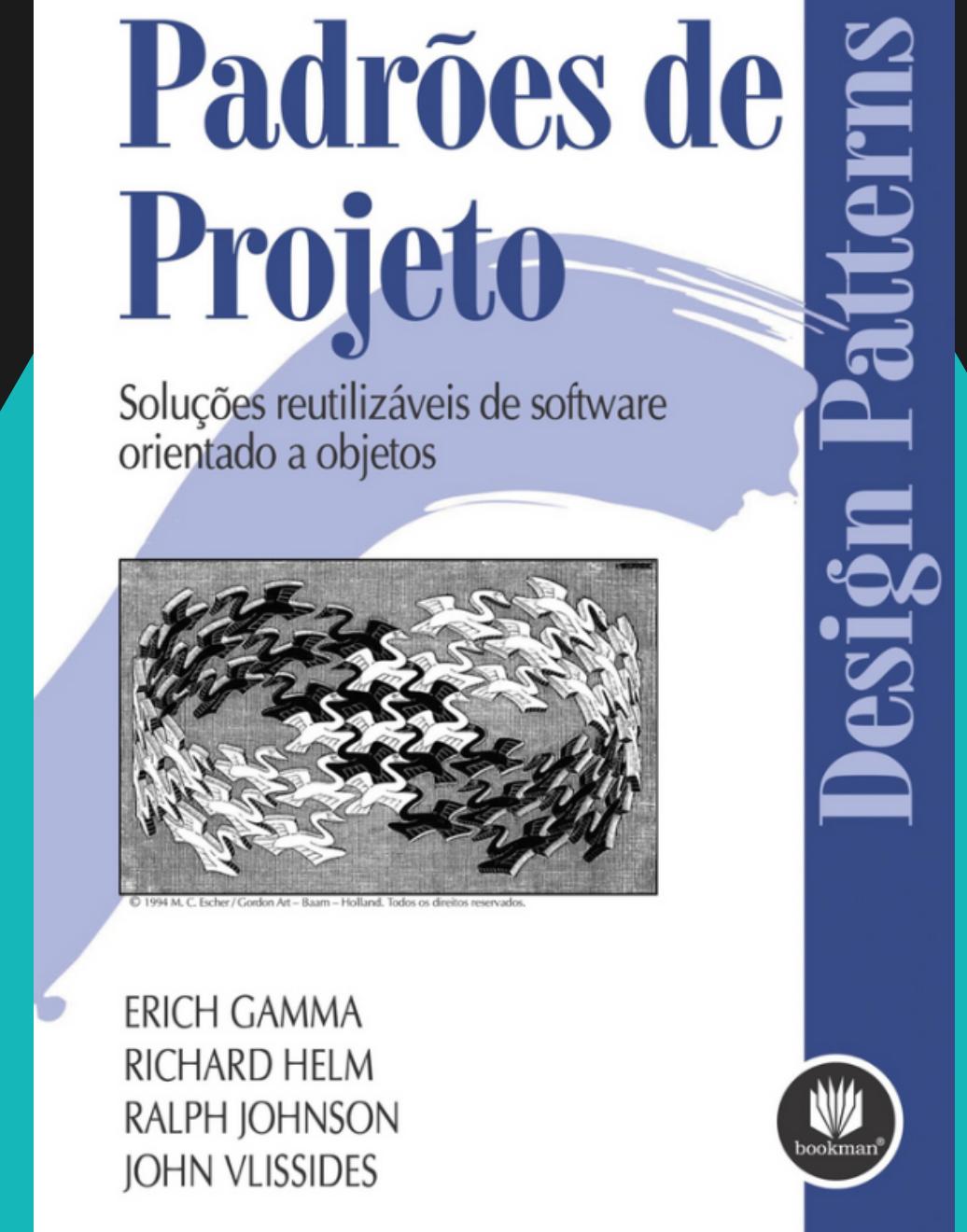
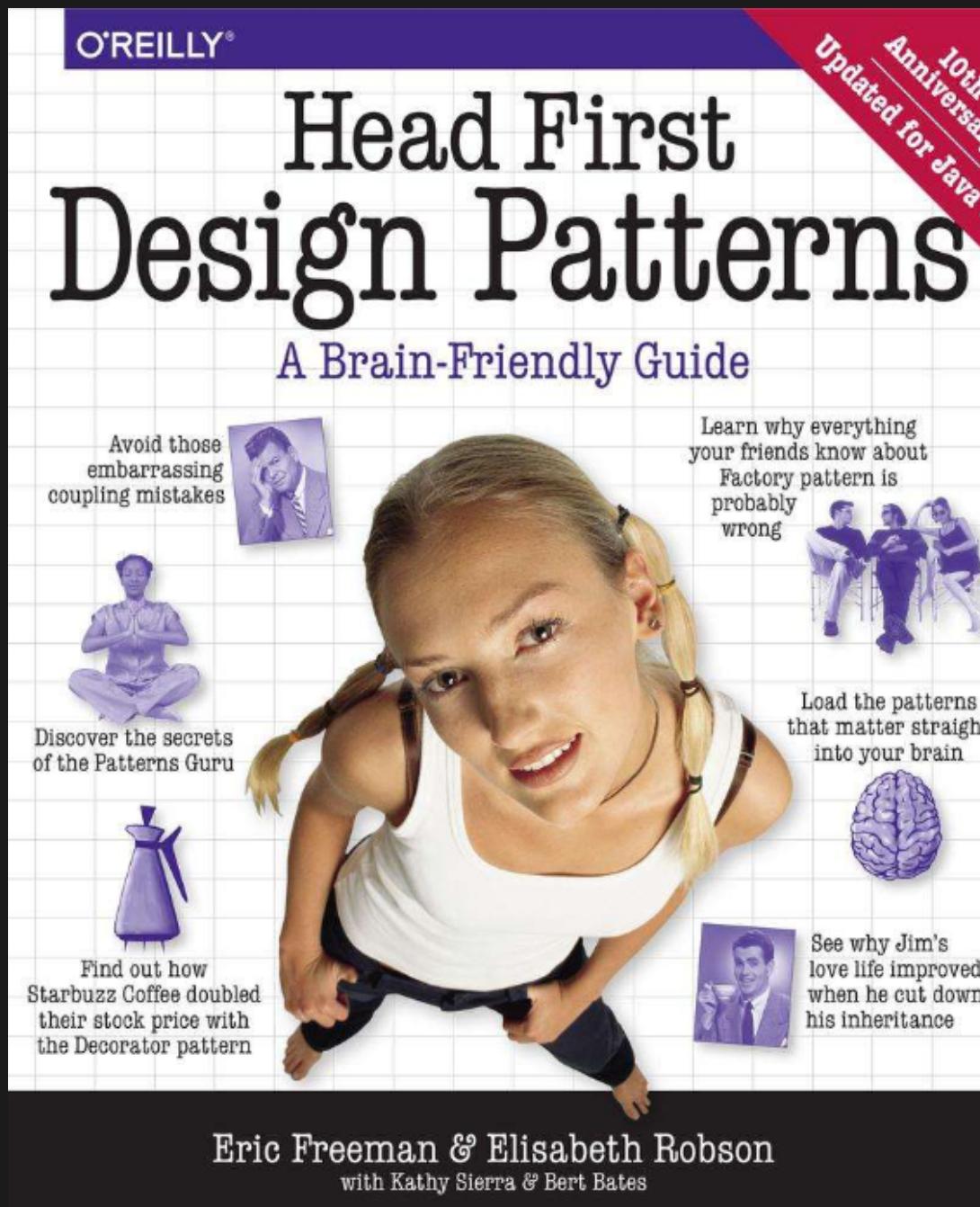
## REPOSITÓRIO GITHUB

Todos os códigos gerados no curso estão disponíveis no Github da Cod3r.

Cada padrão descreve um problema que ocorre freqüentemente em seu ambiente, e então descreve o cerne da solução para aquele problema, de um modo tal que você pode usar esta solução milhões de vezes, sem nunca fazer a mesma coisa repetida

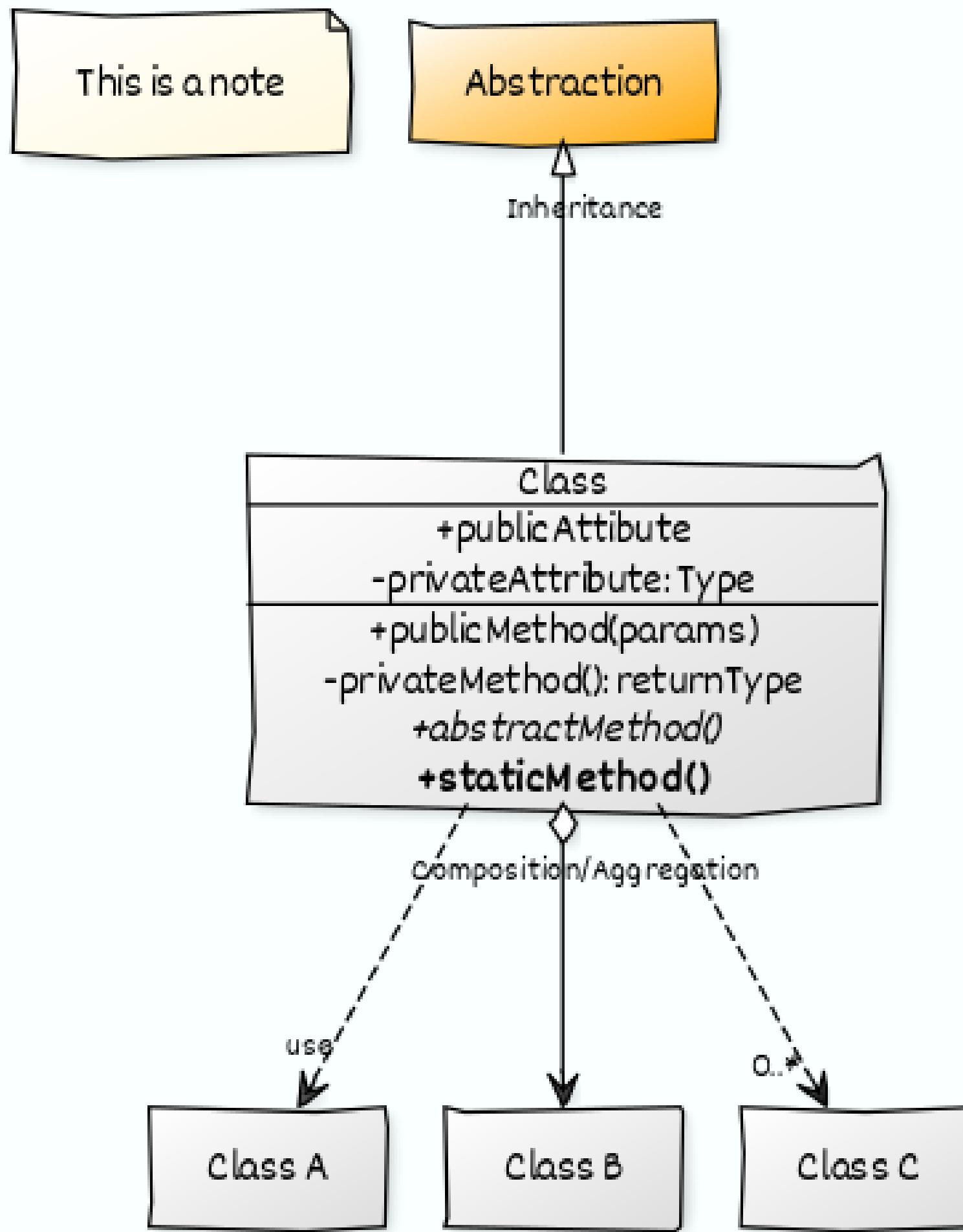
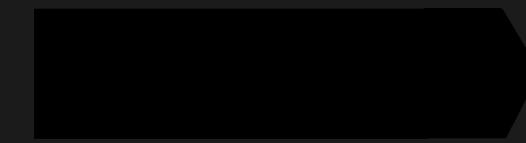
**CHRISTOPHER ALEXANDER**





# BIBLIOGRAFIA

# ENTENDENDO OS DIAGRAMAS



# MÓDULO PADRÕES DE PROJETOS CRIACIONAIS

MUITAS VEZES CRIAR OBJETOS PODE SER  
COMPLICADO E POR ESSE MOTIVO EXISTEM  
PADRÕES PRA AJUDAR!

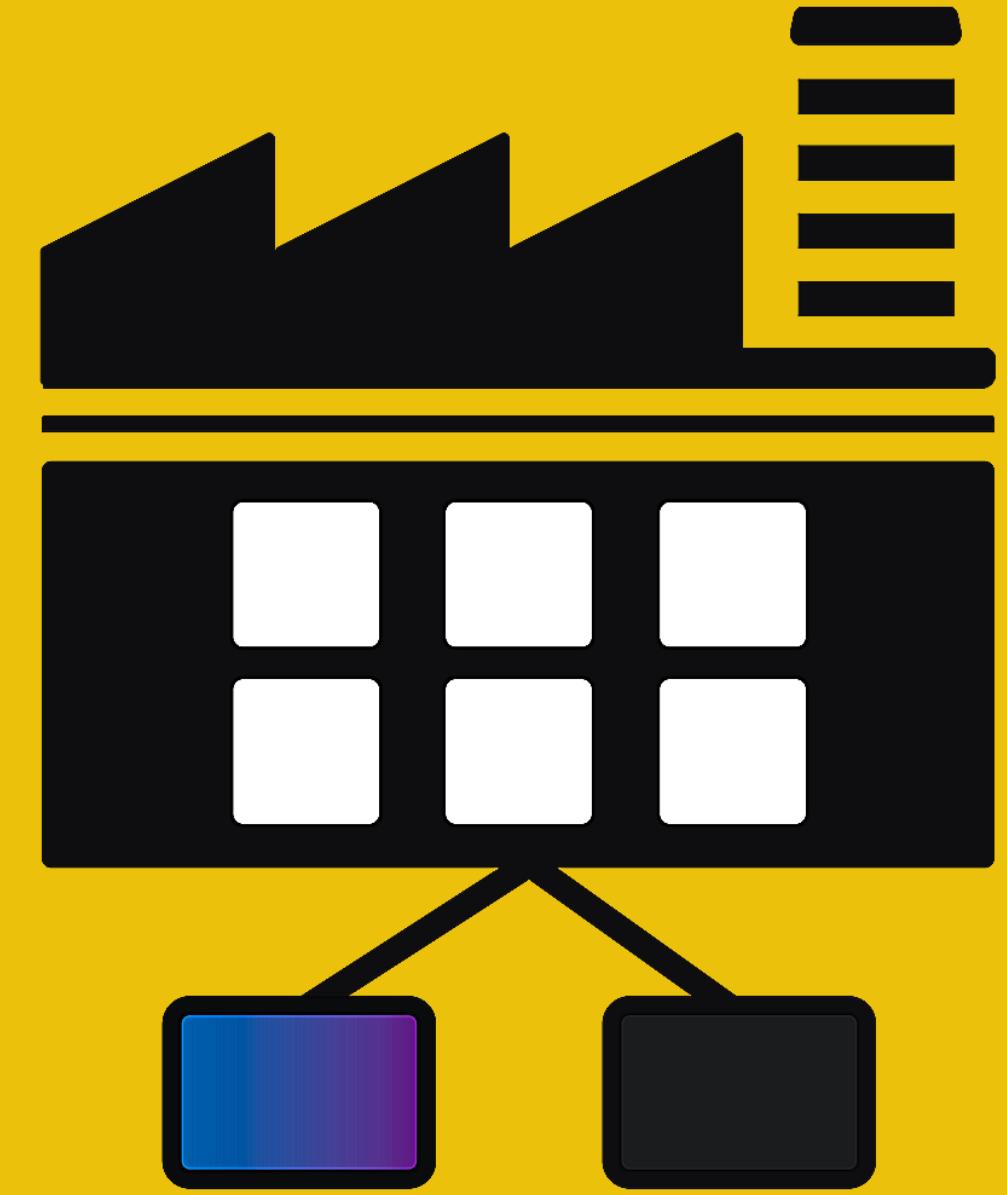


## OBJETIVOS DO MÓDULO

- O QUE SÃO PADRÕES CRIACIONAIS
- ENTENDER SUA IMPORTÂNCIA
- CONHECER OS PRINCIPAIS PADRÕES DESTE TIPO
- ENTENDER SUAS DIFERENÇAS E RELACIONAMENTOS

# AULA FACTORY METHOD

CHEGOU A HORA DE CONHECER O NOSSO  
PRIMEIRO PADRÃO, E UM DOS MAIS  
FAMOSOS!





## OBJETIVOS

- APRESENTAR O PROBLEMA GERAL
- APRESENTAR UMA SOLUÇÃO UTILIZANDO O FACTORY
- APRESENTAR UMA VARIAÇÃO MAIS UTILIZADA
- REFATORAR O NOSSA IMPLEMENTAÇÃO PARA UM MEIO-TERMO
- APRESENTAR UMA UTILIZAÇÃO BEM COMUM NA ESTRUTURA DOS SISTEMAS

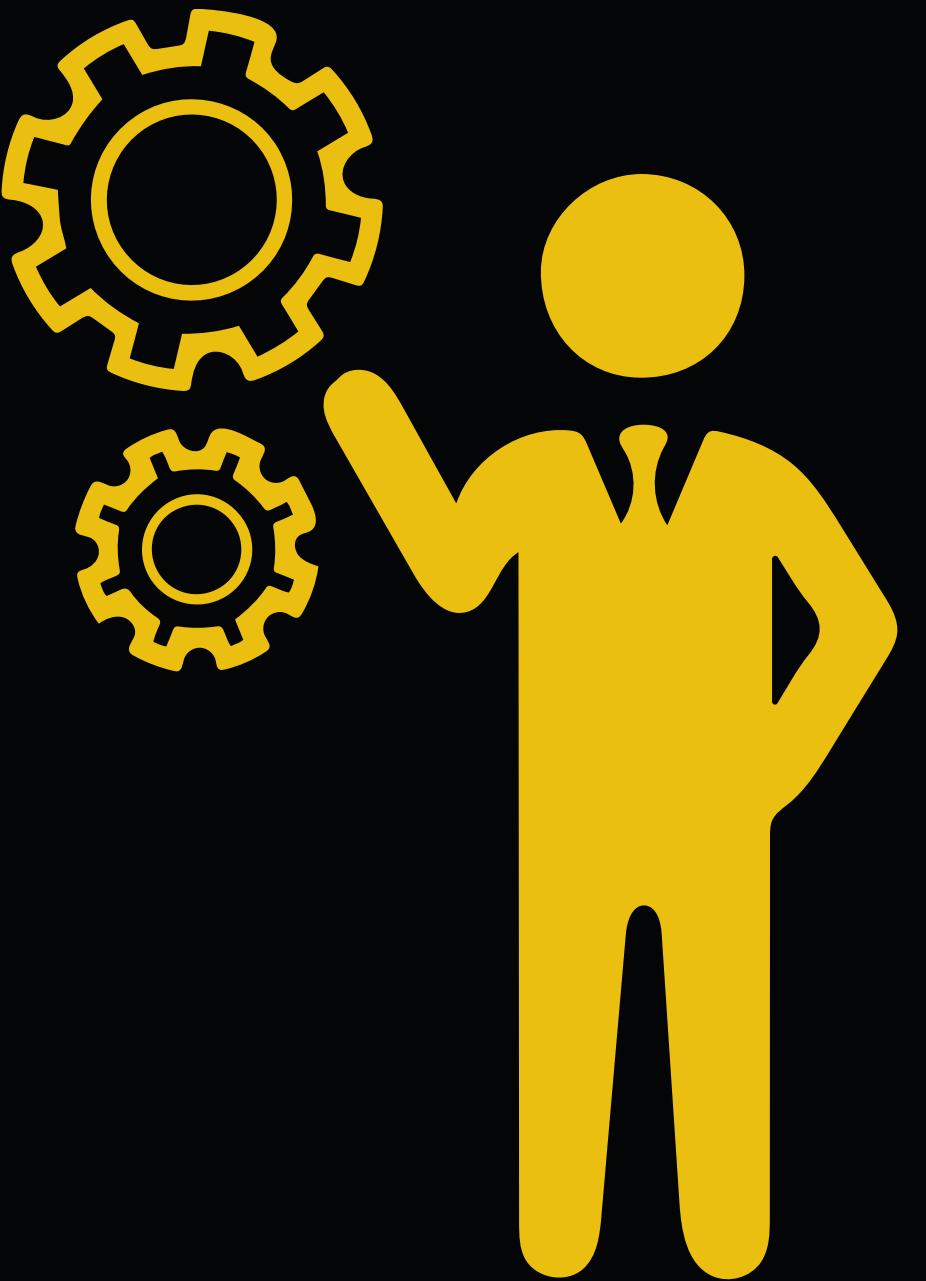
# PROBLEMAS

- COMO POSSO ESCREVER UM CÓDIGO ONDE AS CLASSES INSTANCIADAS POSSAM VARIAR DENTRO DE UMA MESMA INTERFACE?
- COMO DEIXAR O MEU CÓDIGO DESACOPLADO DAS CLASSES CONCRETAS?



# SOLUÇÃO

- EXTRAIR A LÓGICA DE CRIAÇÃO DOS OBJETOS PARA UM FACTORY METHOD
- INVOCAR O FACTORY METHOD PARA RECEBER UMA INSTÂNCIA QUALQUER QUE IMPLEMENTE UMA DETERMINADA INTERFACE





Um padrão que define uma interface para criar um objeto, mas permite às classes decidirem qual classe instanciar. O Factory Method permite a uma classe deferir a instanciação para subclasses.

**GOF**



Sample Code

```
import br.com.cod3r.factory.apple.before.model.IPhone;
import br.com.cod3r.factory.apple.before.model.IPhone11;
import br.com.cod3r.factory.apple.before.model.IPhoneX;

public IPhone orderIPhone() {
    IPhone device = null;

    //...

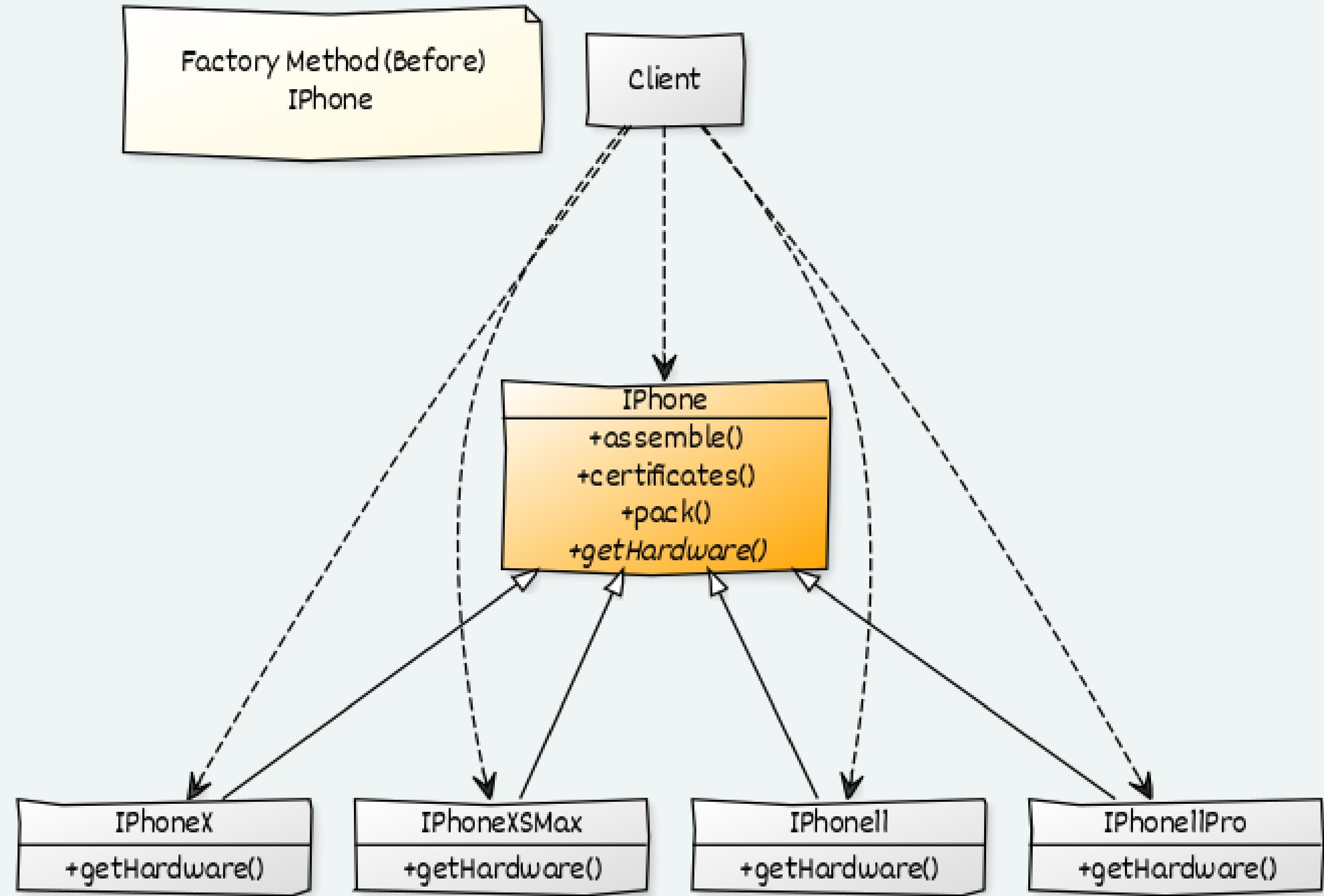
    device = new IPhoneX();

    //...

    device = new IPhone11();

    //...

    return device;
}
```



# factory > apple > after > ☕ Client.java

## Sample Code

```
import br.com.cod3r.factory.apple.after.factory.IPhone11ProFactory;
import br.com.cod3r.factory.apple.after.factory.IPhoneFactory;
import br.com.cod3r.factory.apple.after.factory.IPhoneXFactory;
import br.com.cod3r.factory.apple.after.model.IPhone;

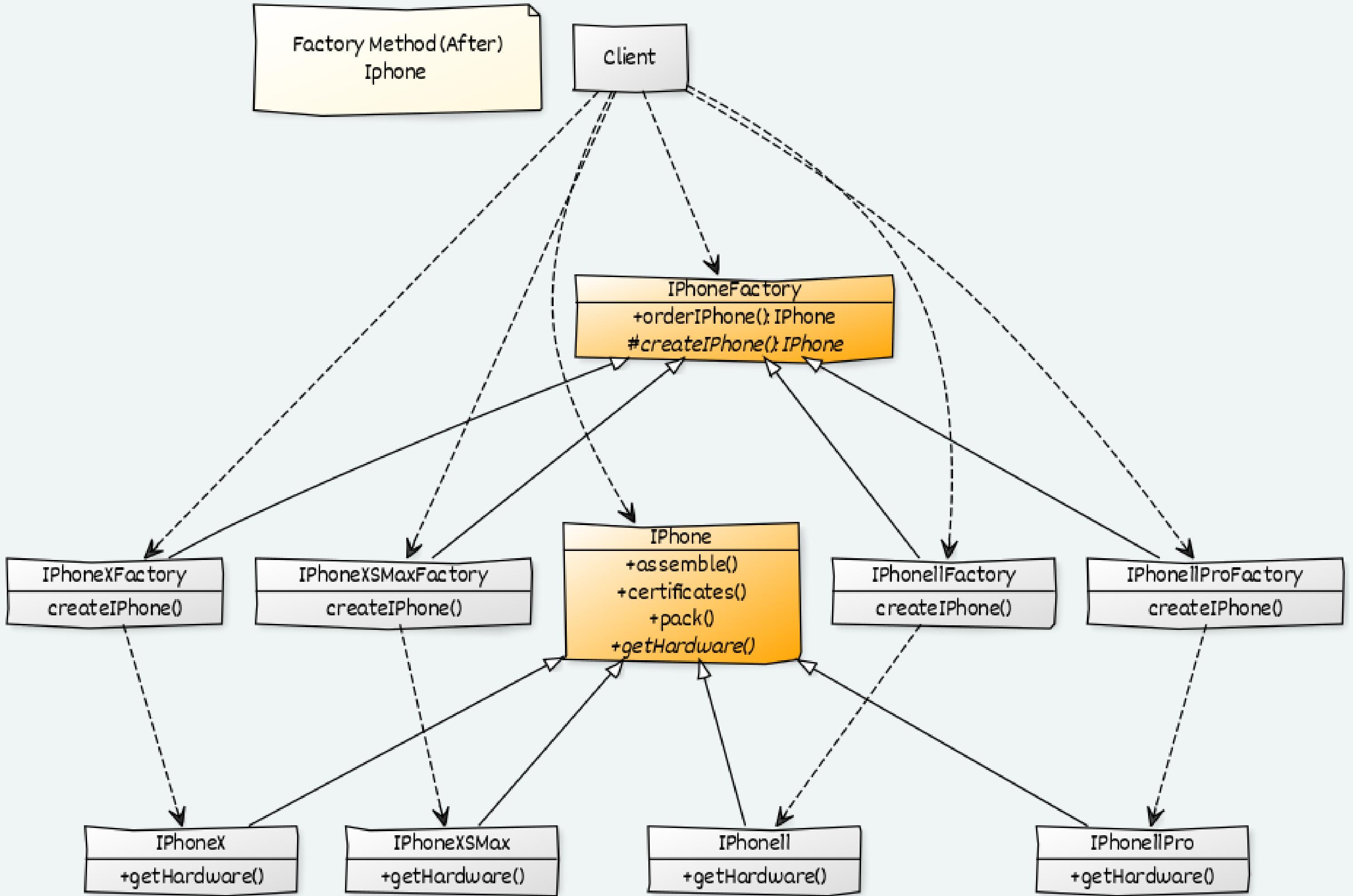
public IPhone orderIPhone() {
    IPhone device = null;
    IPhoneFactory iphoneXFactory = new IPhoneXFactory();
    IPhoneFactory iphone11ProFactory = new IPhone11ProFactory();

    //...

    device = iphoneXFactory.orderIPhone();

    //...

    device = iphone11ProFactory.orderIPhone();
    return device;
}
```



 factory > apple > simple > Client.java Sample Code

```
import br.com.cod3r.factory.apple.simple.factory.IPhoneSimpleFactory;
import br.com.cod3r.factory.apple.simple.model.IPhone;

public IPhone orderIPhone() {
    IPhone device = null;

    //...

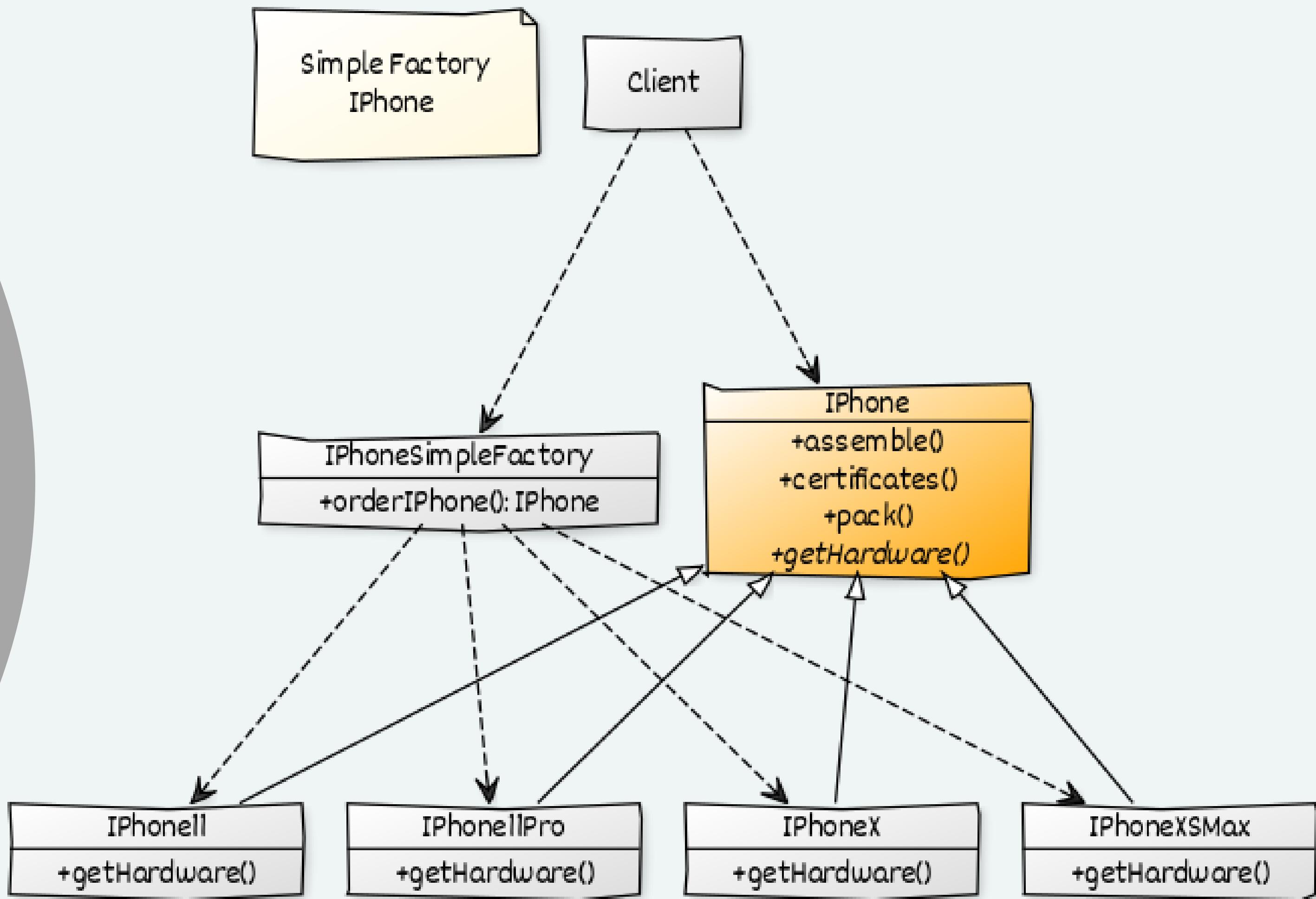
    device = IPhoneSimpleFactory.orderIPhone("X", "standard");

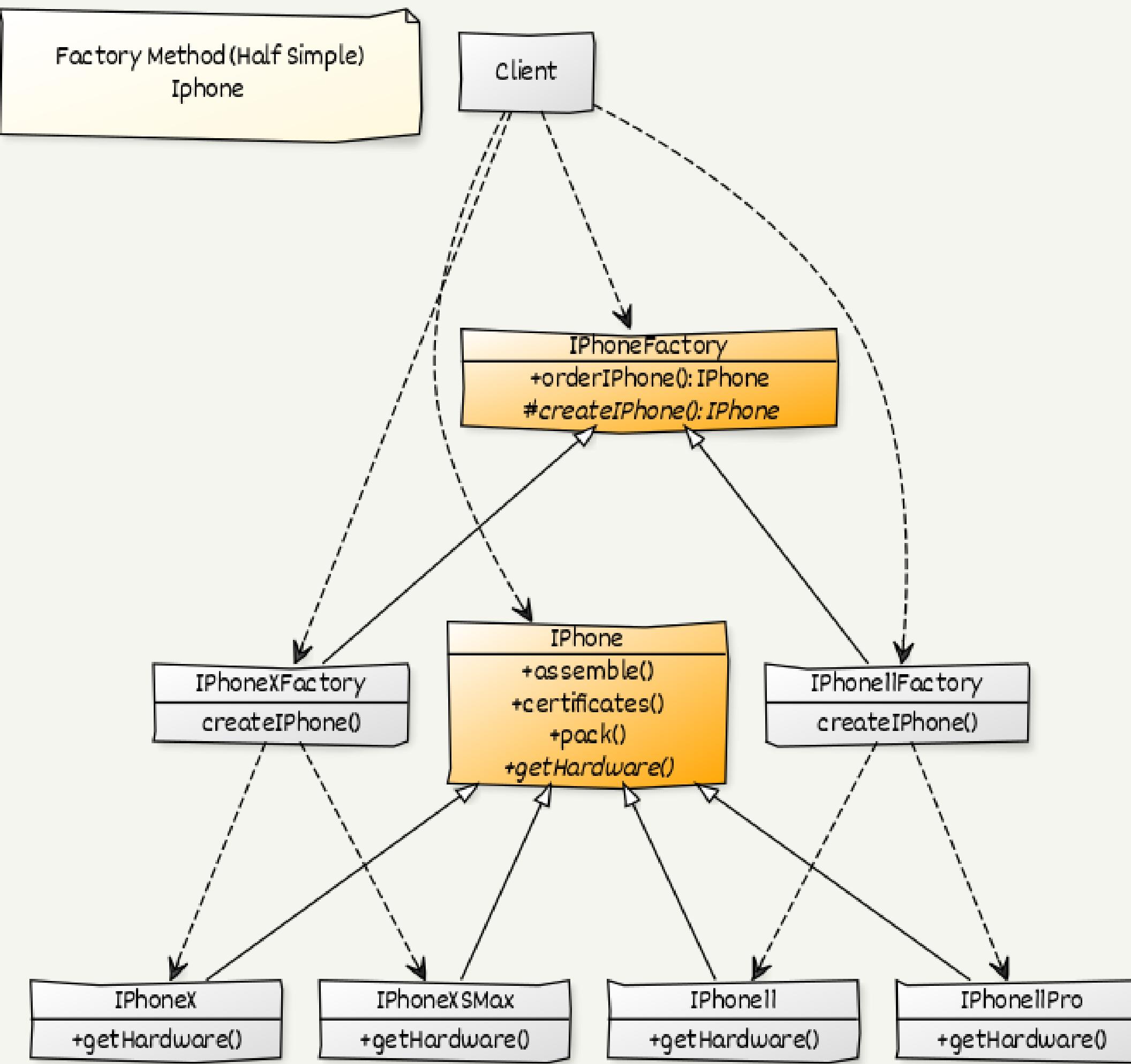
    //...

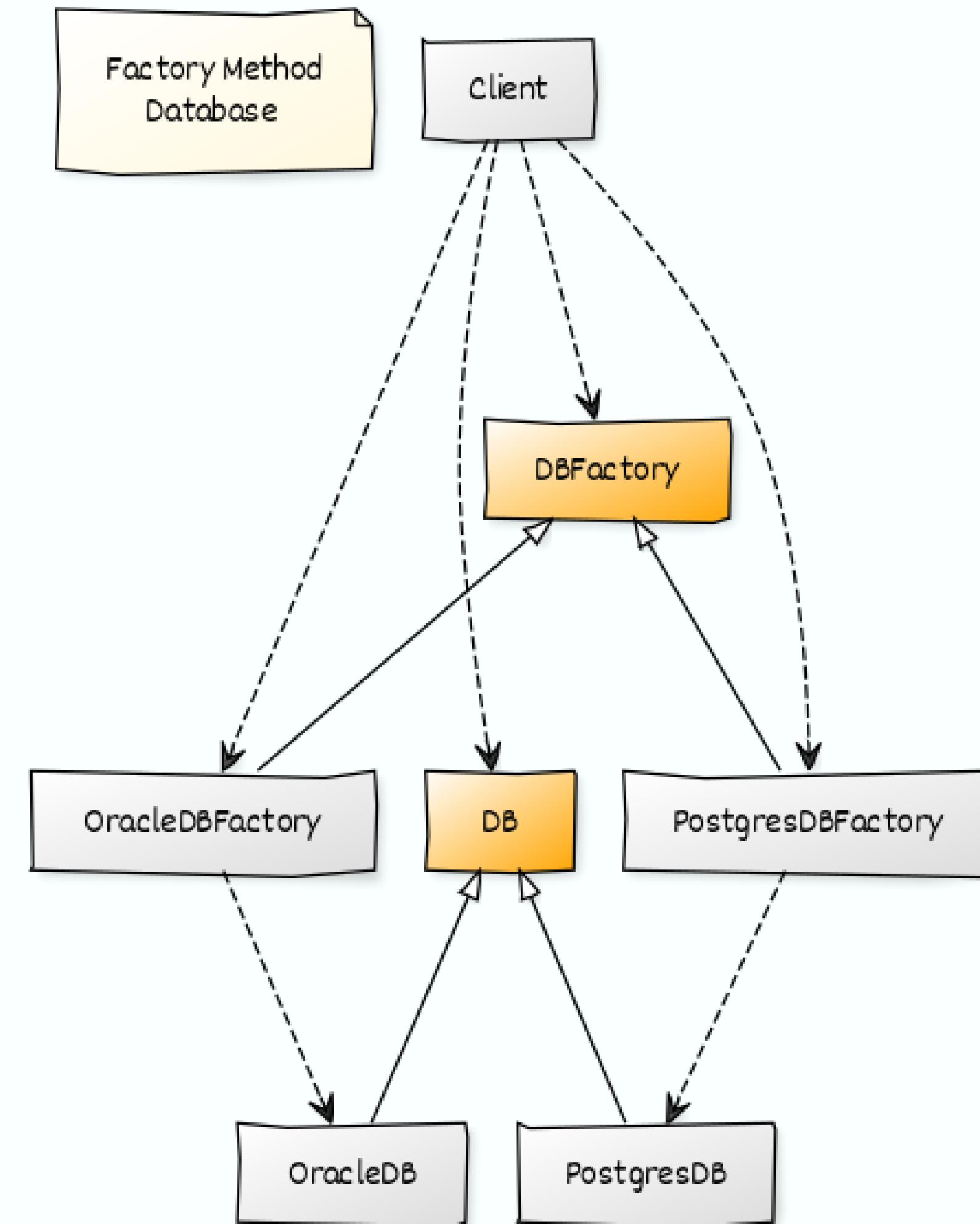
    device = IPhoneSimpleFactory.orderIPhone("11", "highEnd");

    //...

    return device;
}
```

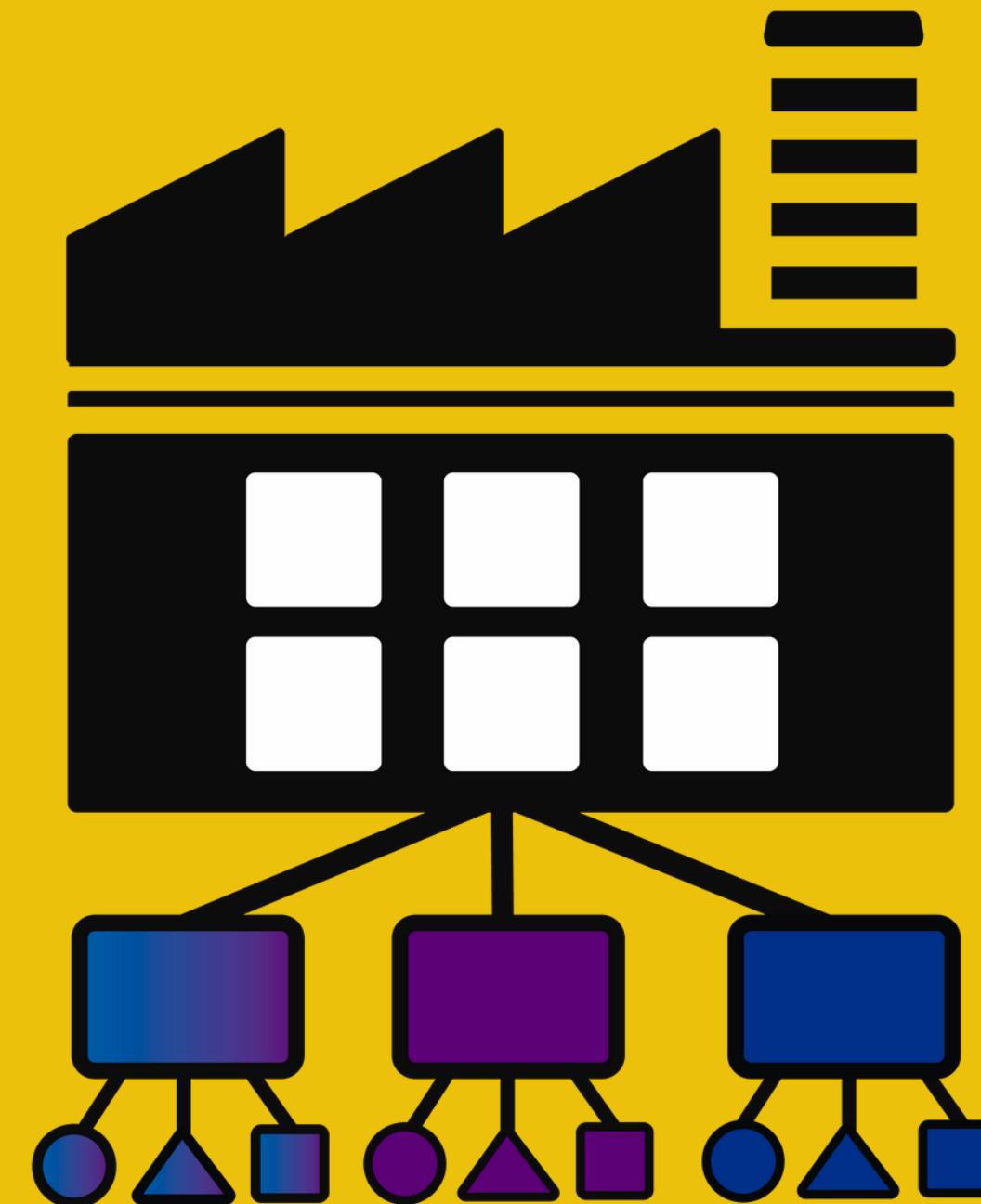






# AULA ABSTRACT FACTORY

ESSE SIM É UM CARA DE FAMÍLIA





## OBJETIVOS

- APRESENTAR O PROBLEMA GERAL
- APRESENTAR UMA SOLUÇÃO UTILIZANDO O ABSTRACT FACTORY
- APRESENTAR COMO ESTE PADRÃO AJUDA A MANTER O CONTEXTO DA FAMÍLIA
- APRESENTAR COMO A ESTRUTURA PODE SER ALTERADA DE ACORDO COM A MODELAGEM DA SOLUÇÃO

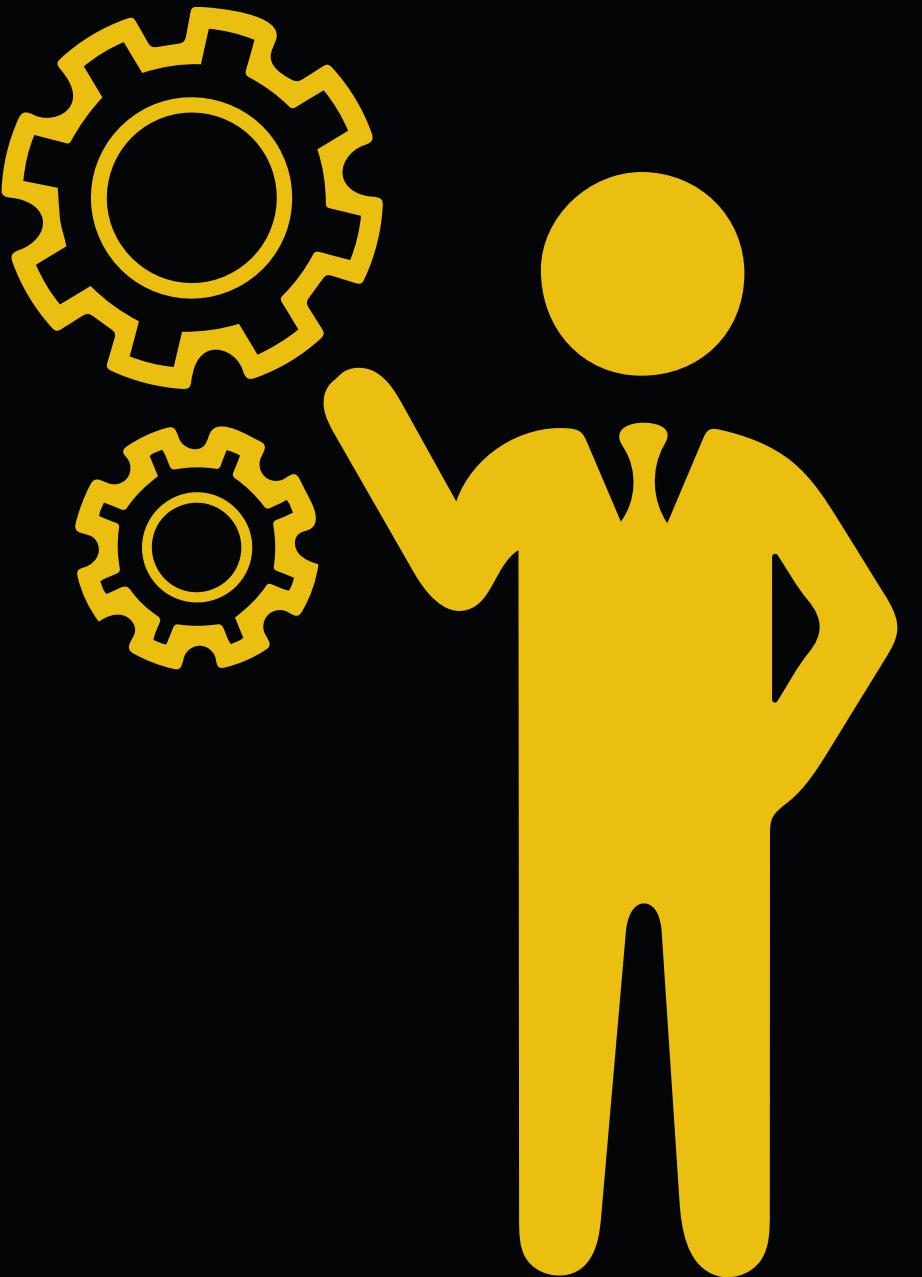
# PROBLEMAS

- COMO POSSO ESCREVER UM CÓDIGO ONDE AS CLASSES INSTANCIADAS POSSAM VARIAR DENTRO DE UMA MESMA INTERFACE?
- COMO GARANTIR QUE UM CONJUNTO DE OBJETOS RELACIONADOS (OU DEPENDENTES) POSSAM SER CRIADOS MANTENDO O CONTEXTO ÚNICO?



# SOLUÇÃO

- EXTRAIR A LÓGICA DE CRIAÇÃO DOS OBJETOS PARA UM ABSTRACT FACTORY
- CRIAR UMA IMPLEMENTAÇÃO DO ABSTRACT FACTORY PARA CADA CONTEXTO, GARANTINDO QUE TODOS OS OBJETOS CRIADOS ESTEJAM RELACIONADOS

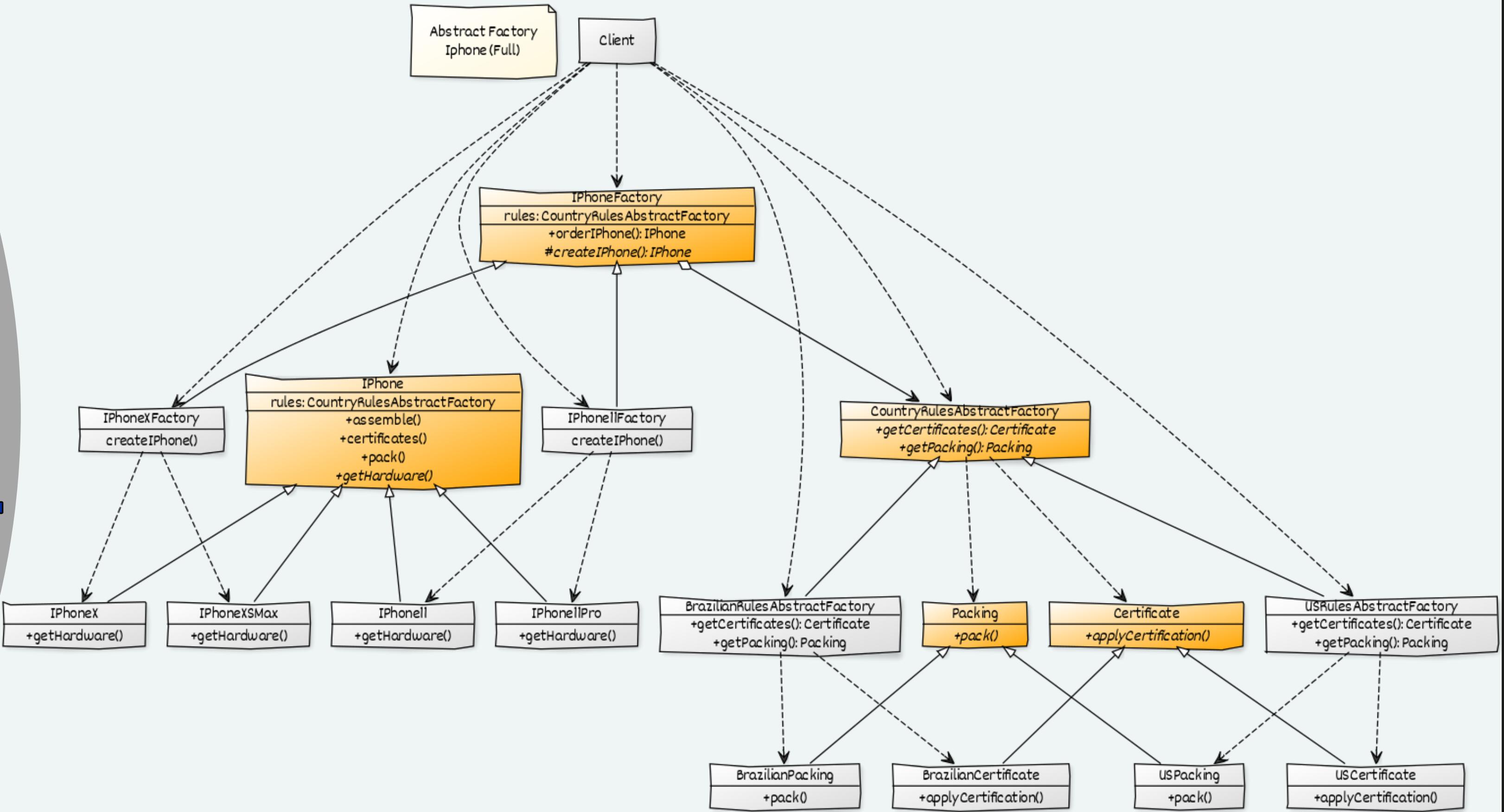


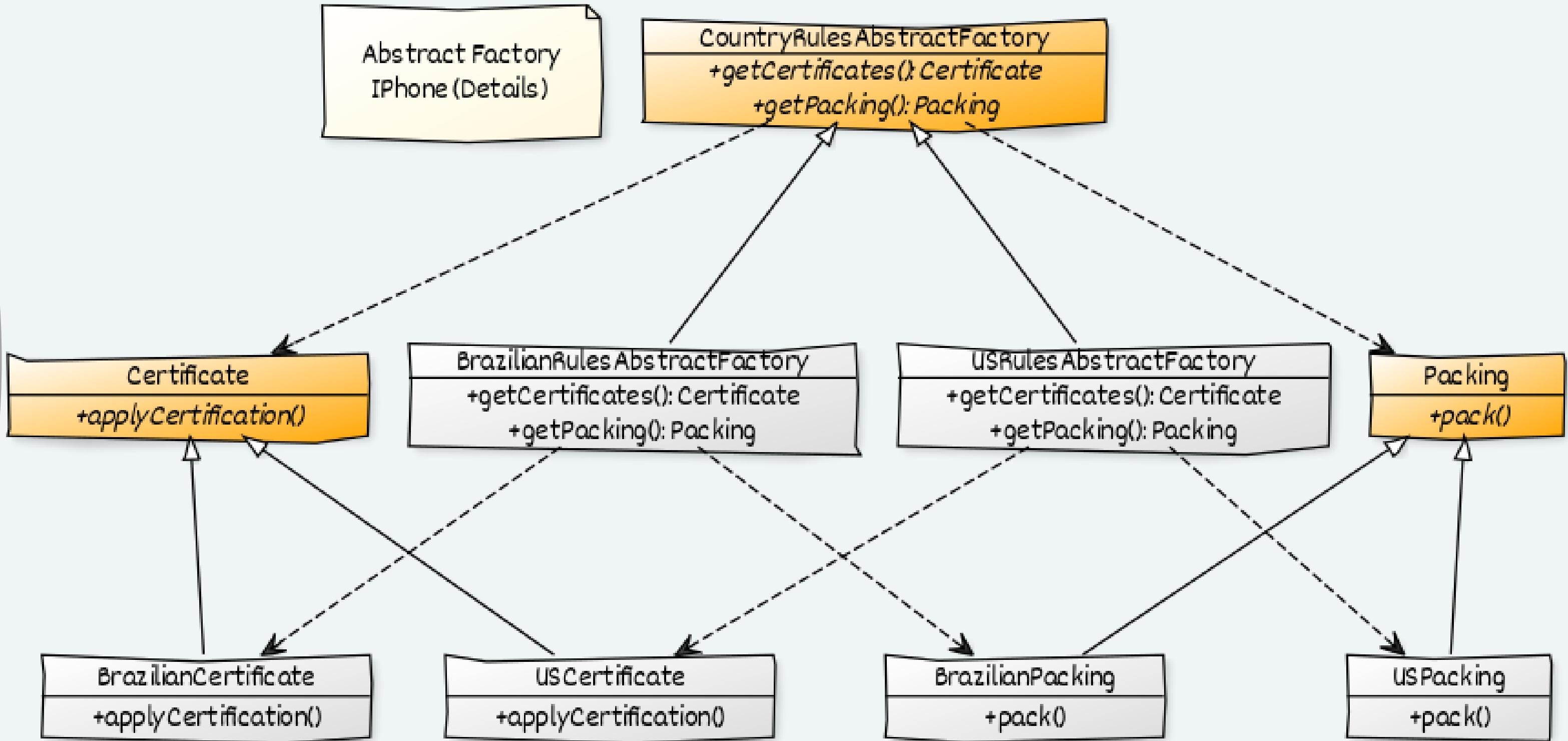
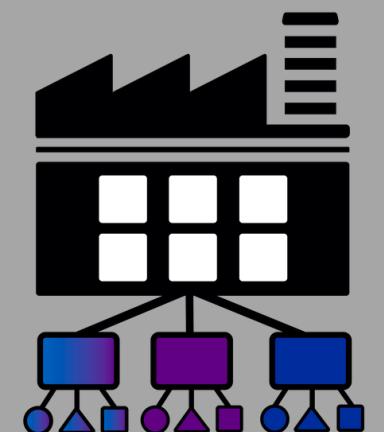


Prover uma interface para criar famílias de objetos relacionados ou dependentes sem especificar suas classes concretas.

**GOF**



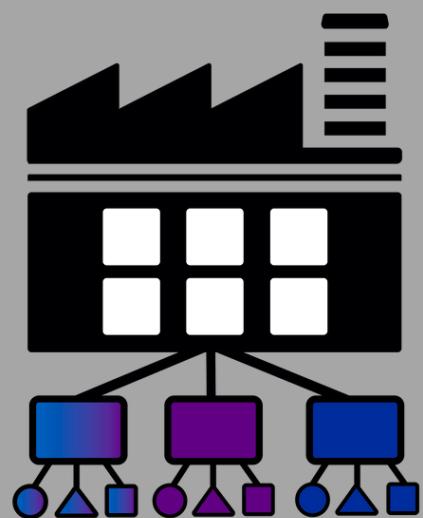
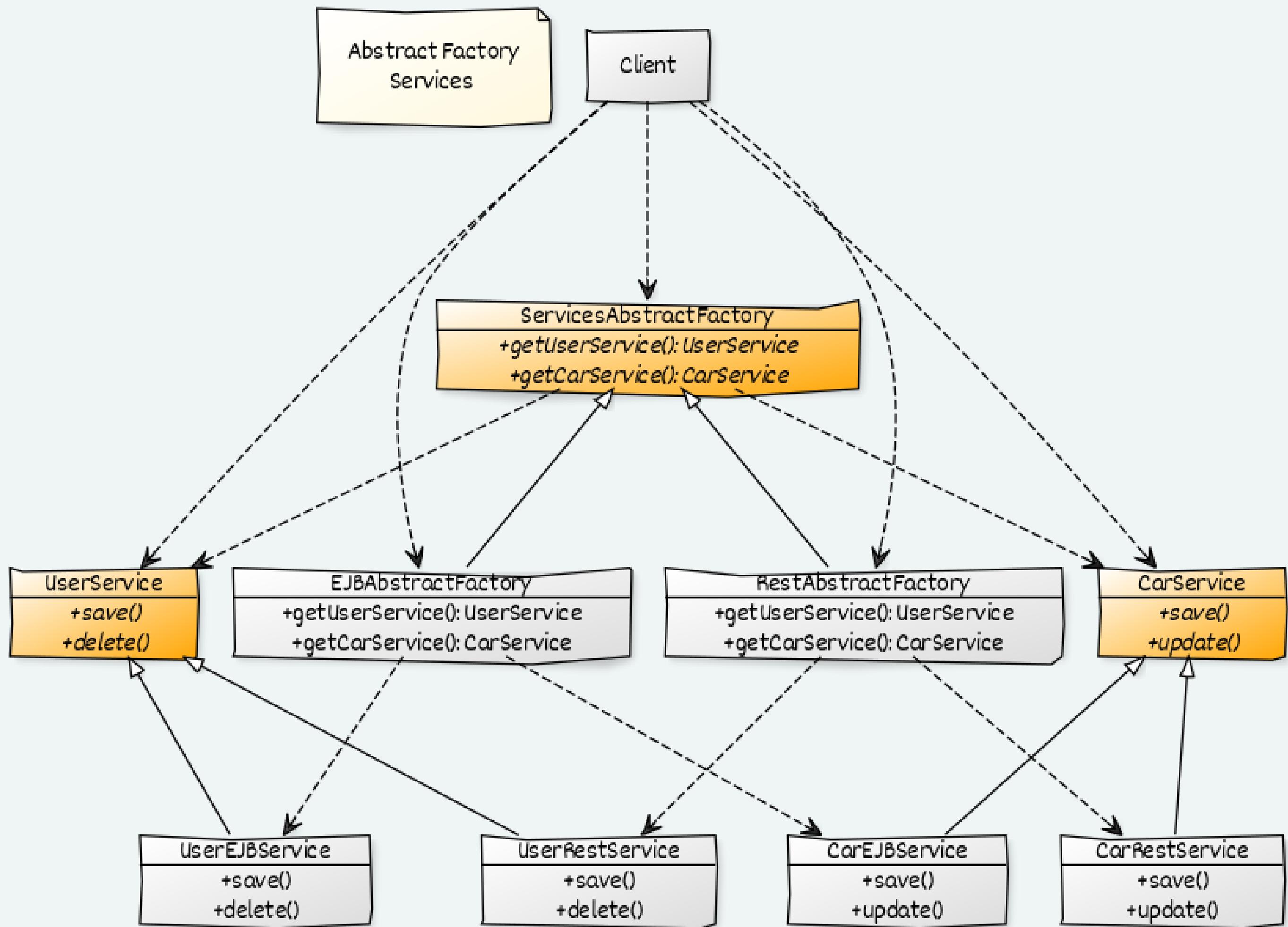


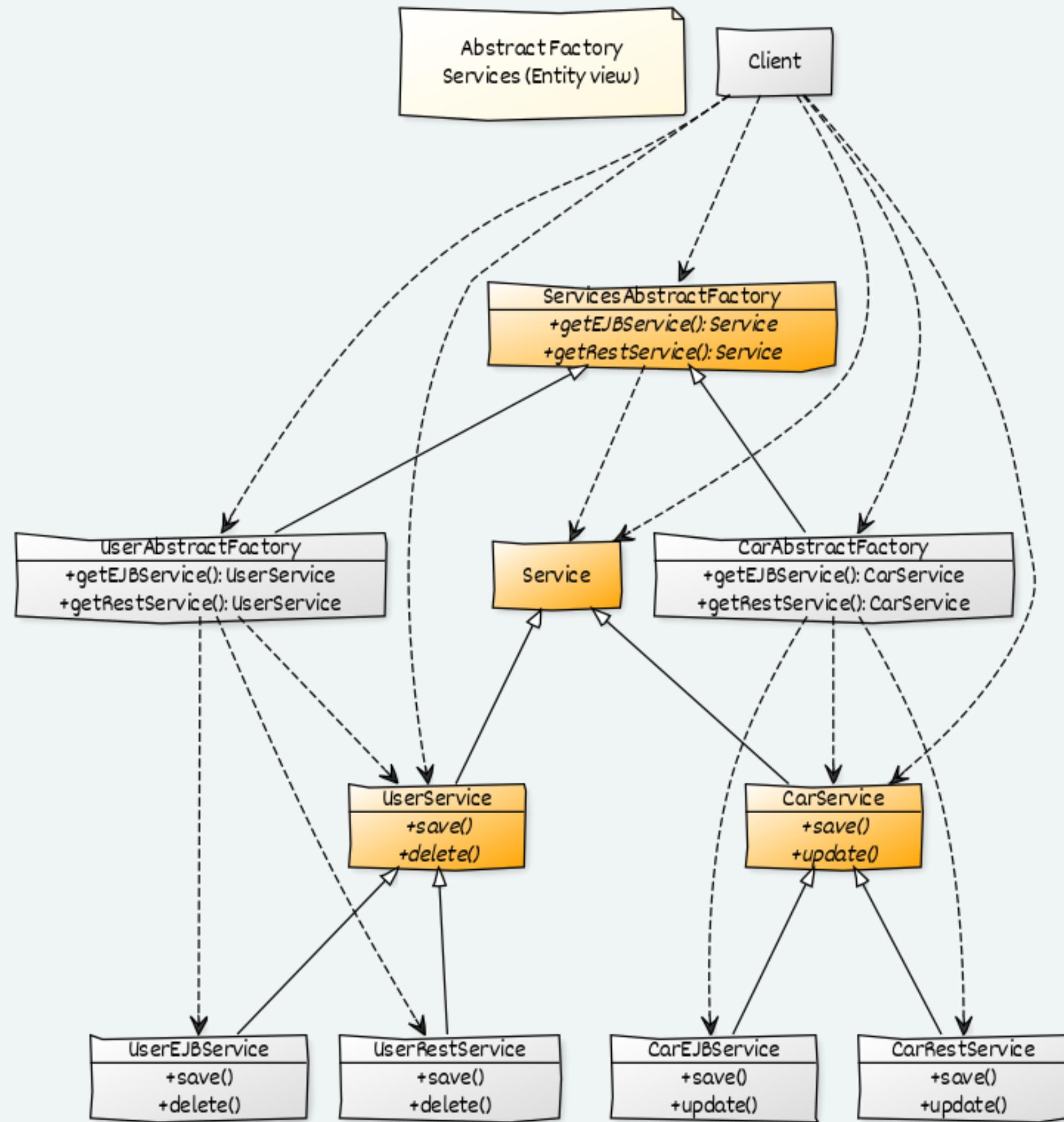
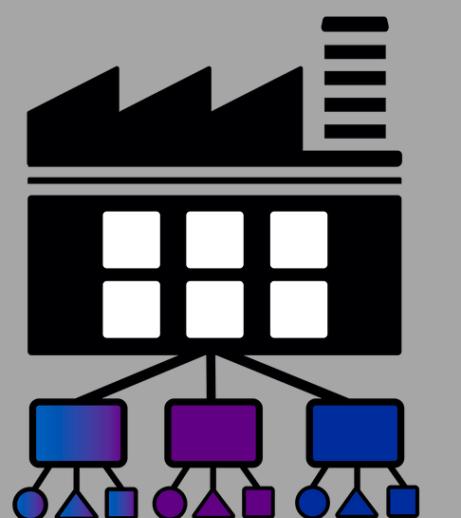


CREATED WITH YUML

Sample Code

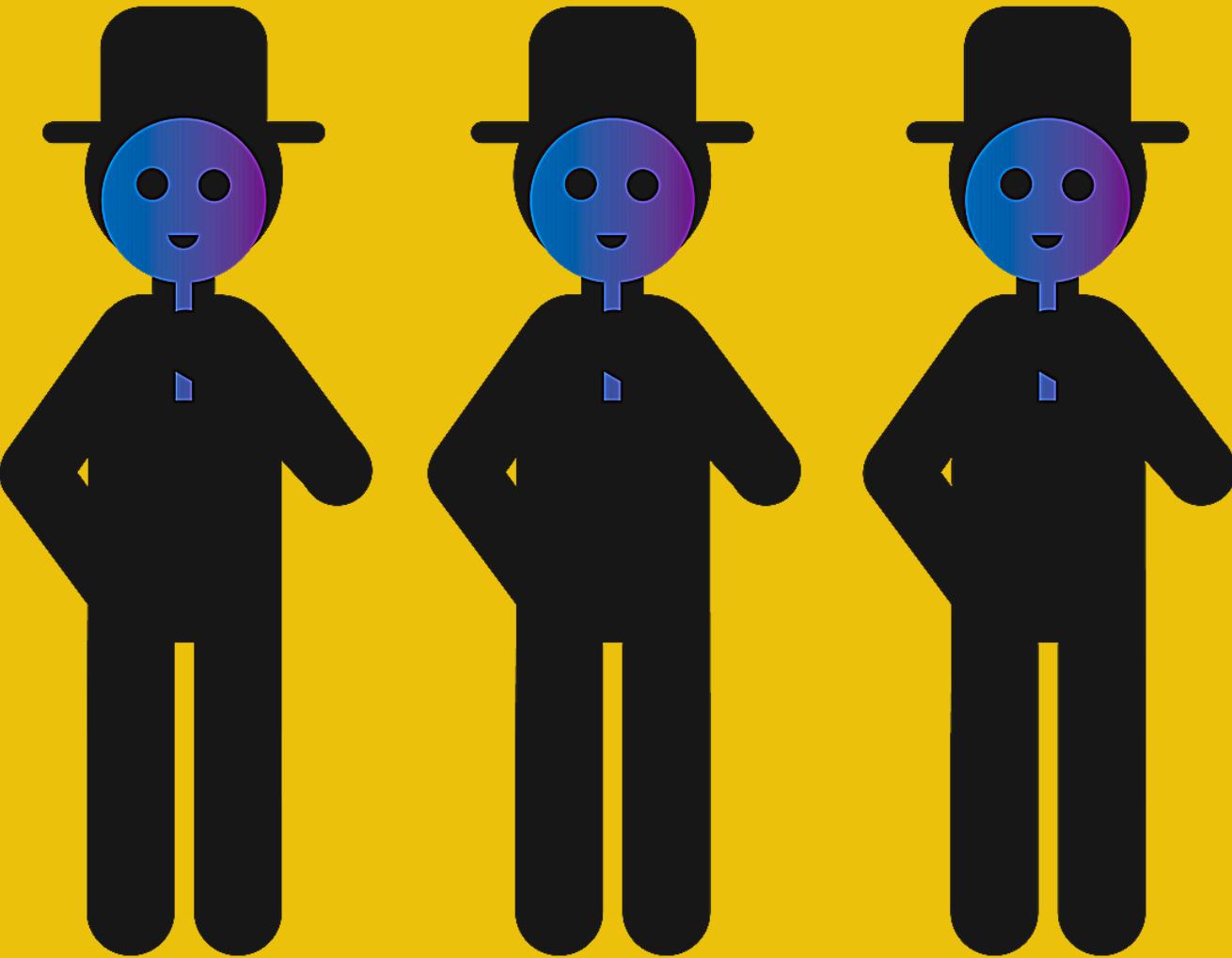
```
public abstract class IPhone {  
    CountryRulesAbstractFactory rules;  
  
    public IPhone(CountryRulesAbstractFactory rules) {  
        this.rules = rules;  
    }  
  
    //...  
  
    public void certificates() {  
        System.out.println("Testing all the certificates");  
        System.out.println(rules.getCertificates().applyCertification());  
    }  
  
    public void pack() {  
        System.out.println("Packing the device");  
        System.out.println(rules.getpacking().pack());  
    }  
}
```





# AULA SINGLETON

UM PADRÃO SIMPLES E POLEMICO.





## OBJETIVOS

- APRESENTAR O PROBLEMA GERAL
- APRESENTAR UMA SOLUÇÃO UTILIZANDO O SINGLETON
- DISCUTIR OS PROBLEMAS DESTE PADRÃO E DAR UMA ALTERNATIVA

# PROBLEMAS

- COMO POSSO GARANTIR QUE UMA CLASSE TENHA APENAS UMA INSTÂNCIA?
- COMO FAZER COM QUE ESTA INSTÂNCIA ÚNICA POSSA SER ACESSÍVEL GLOBALMENTE?



# SOLUÇÃO

- ESCONDER O CONSTRUTOR DESSA CLASSE...
- DEFINIR UM PONTO DE CRIAÇÃO ESTÁTICO...
- ... QUE RETORNE ESTA INSTÂNCIA ÚNICA

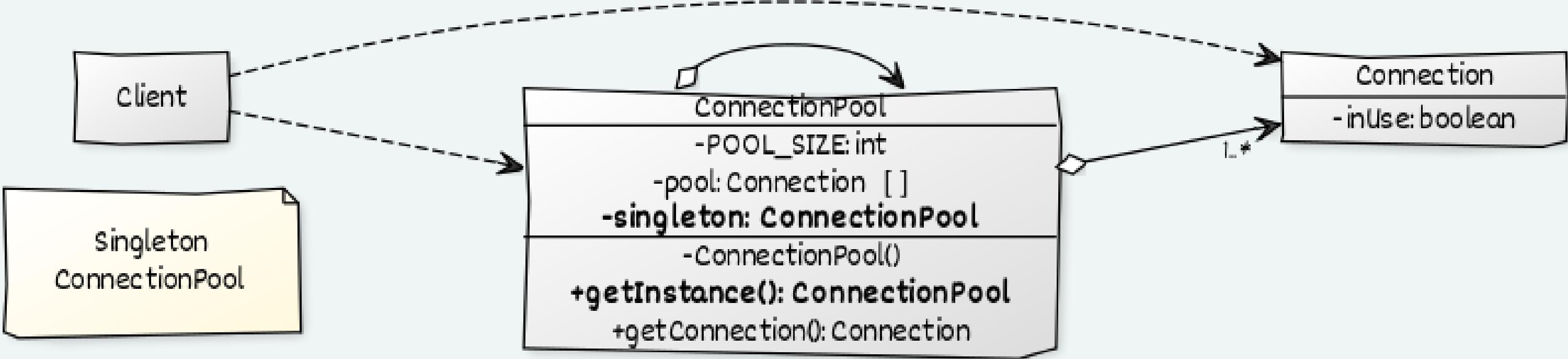
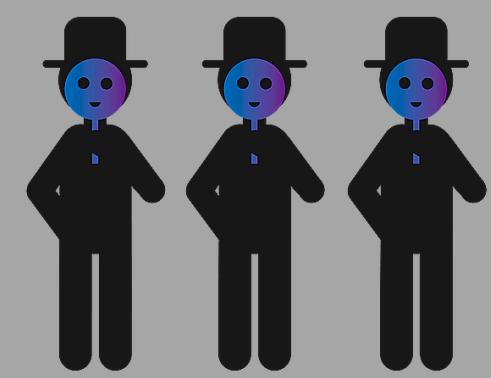




Garantir que uma classe só tenha uma única instância, e prover um ponto de acesso global a ela.

**GOF**

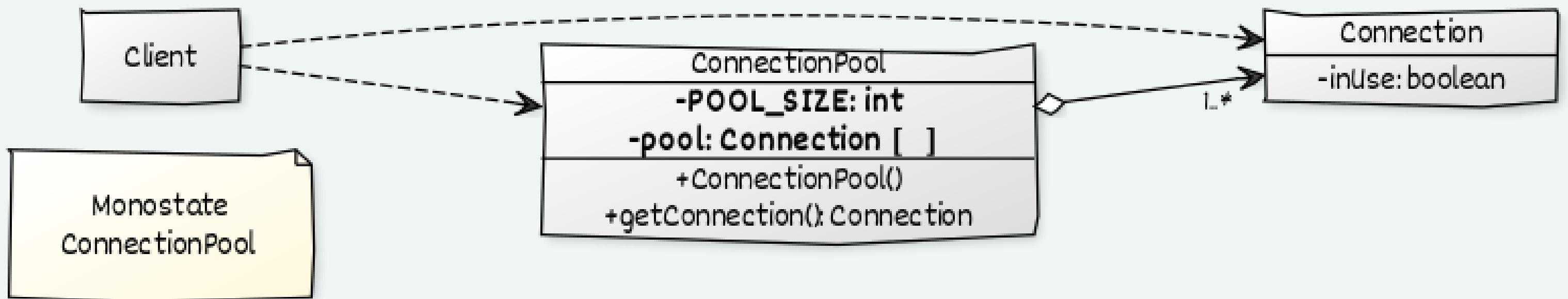




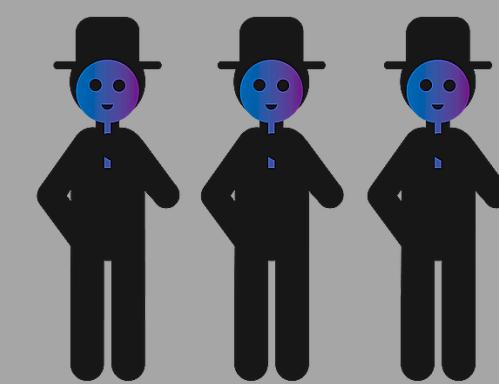
CREATED WITH YUML

Sample Code

```
public class ConnectionPool {  
    private static ConnectionPool singleton = new ConnectionPool();  
    private List<Connection> connectionsPool;  
  
    public static ConnectionPool getInstance() {  
        return singleton;  
    }  
  
    private ConnectionPool() {  
        System.out.println("Creating Connection Pool");  
        connectionsPool = new ArrayList<Connection>();  
        for(int i = 0; i < POOL_SIZE; i++) {  
            connectionsPool.add(new Connection());  
        }  
    }  
    //...  
}
```

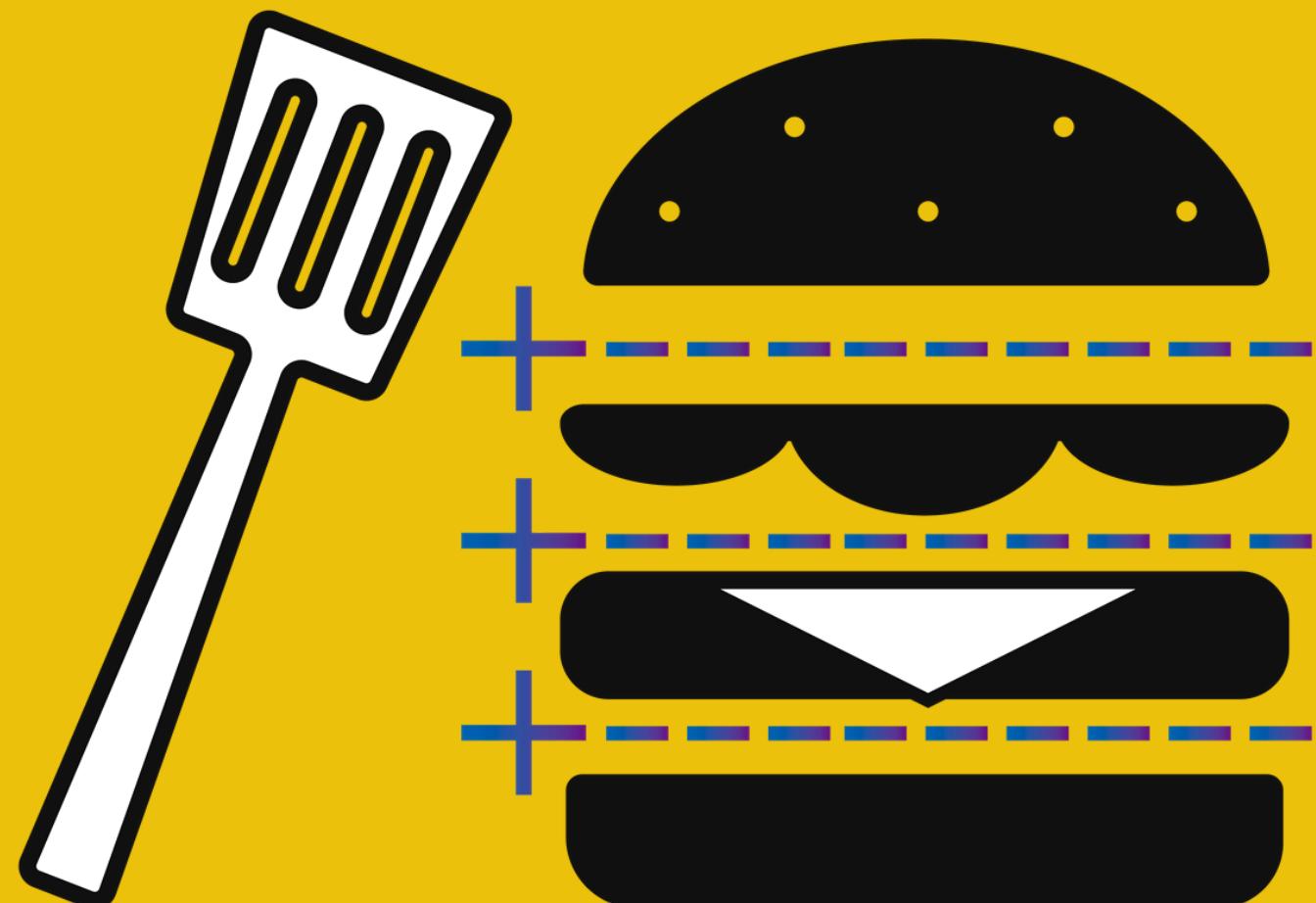


CREATED WITH YUML



# AULA BUILDER

UM PADRÃO CLÁSSICO QUE FOI EVOLUINDO  
COM OUTRAS ABORDAGENS





## OBJETIVOS

- APRESENTAR O PROBLEMA GERAL
- APRESENTAR A SOLUÇÃO CLÁSSICA DO BUILDER
- APRESENTAR UMA SOLUÇÃO ALTERNATIVA COM FLUENT INTERFACE
- REFATORAR O NOSSA IMPLEMENTAÇÃO PARA UMA ABORDAGEM FUNCIONAL

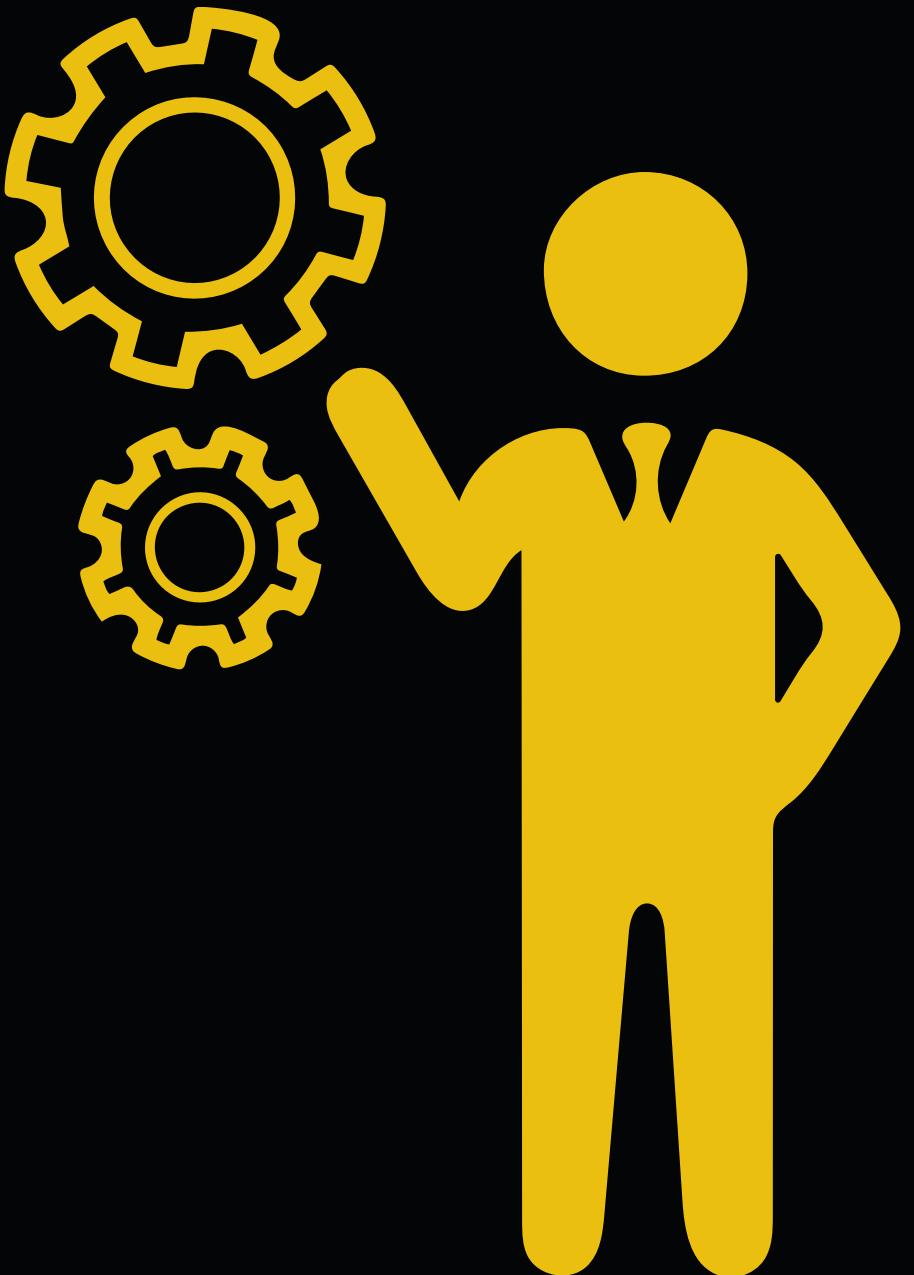
# PROBLEMAS

- COMO UMA CLASSE PODE CRIAR  
DIFERENTES REPRESENTAÇÕES DE UM  
MESMO OBJETO COMPLEXO?



# SOLUÇÃO

- DELEGAR A CRIAÇÃO DO OBJETO PARA UM BUILDER AO INVÉS DE INSTANCIAR O OBJETO CONCRETO DIRETAMENTE
- DIVIDIR A CRIAÇÃO DO OBJETO EM PARTES...
- ENCAPSULAR A CRIAÇÃO E MONTAGEM DESSAS PARTES EM UM BUILDER SEPARADO

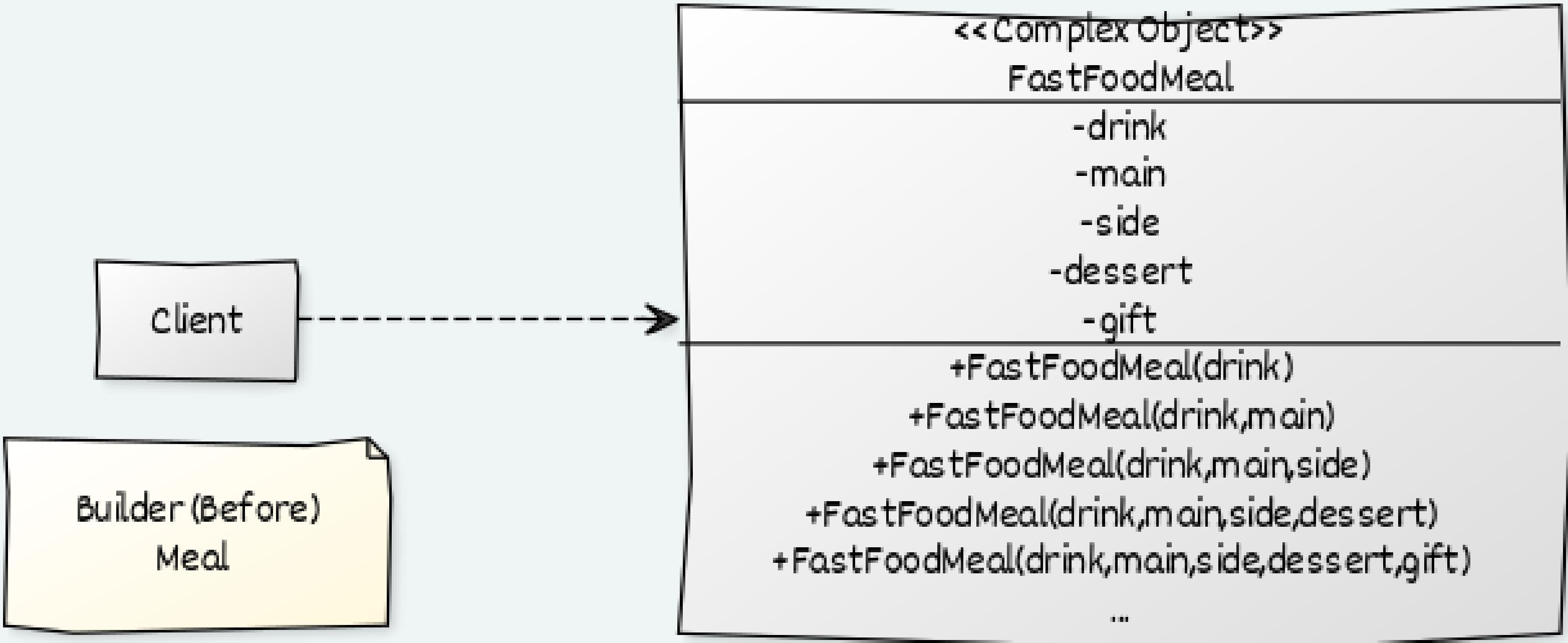




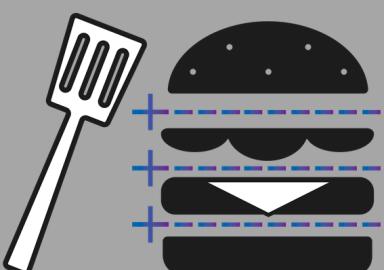
Separar a construção de um objeto complexo de sua representação para que o mesmo processo de construção possa criar representações diferentes.

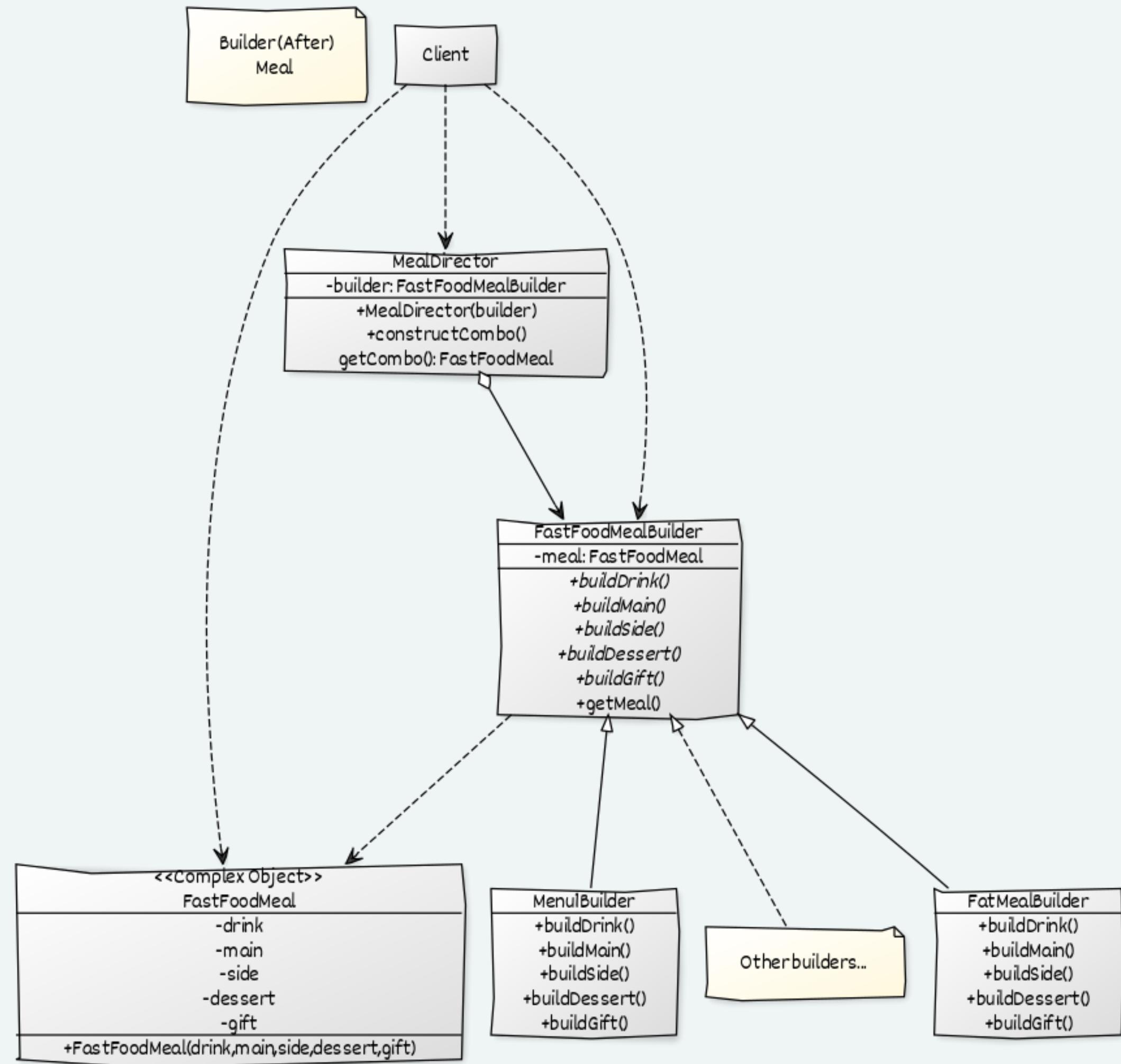
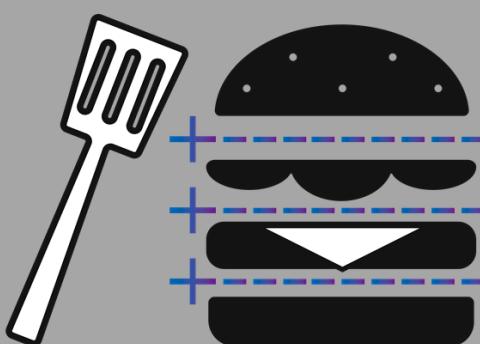
**GOF**

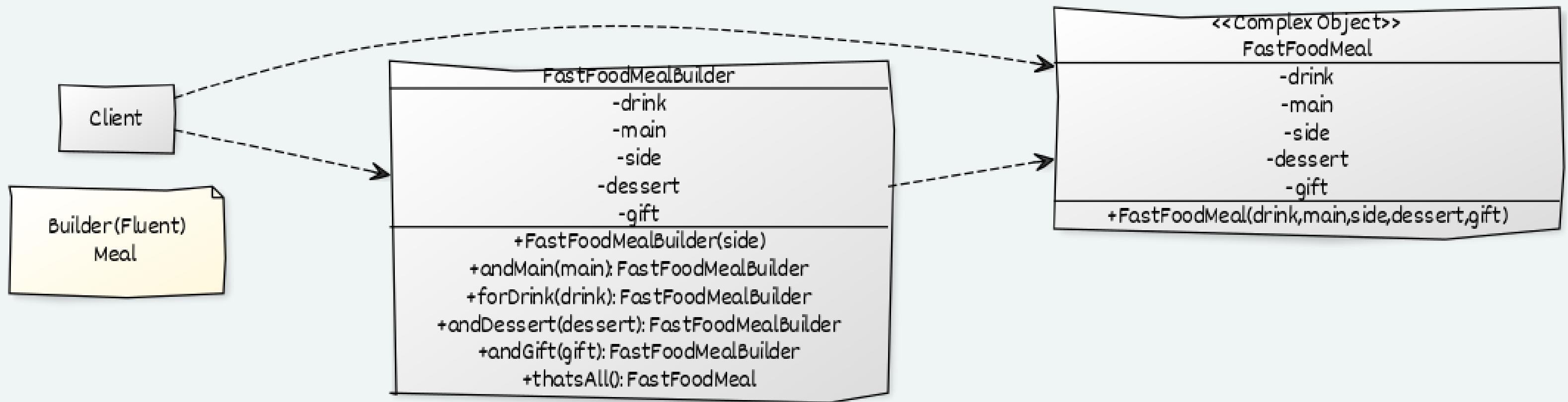




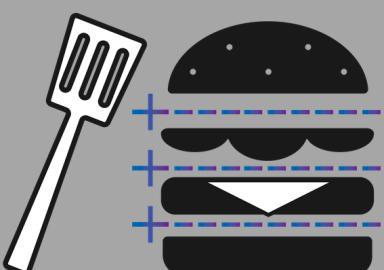
CREATED WITH YUML

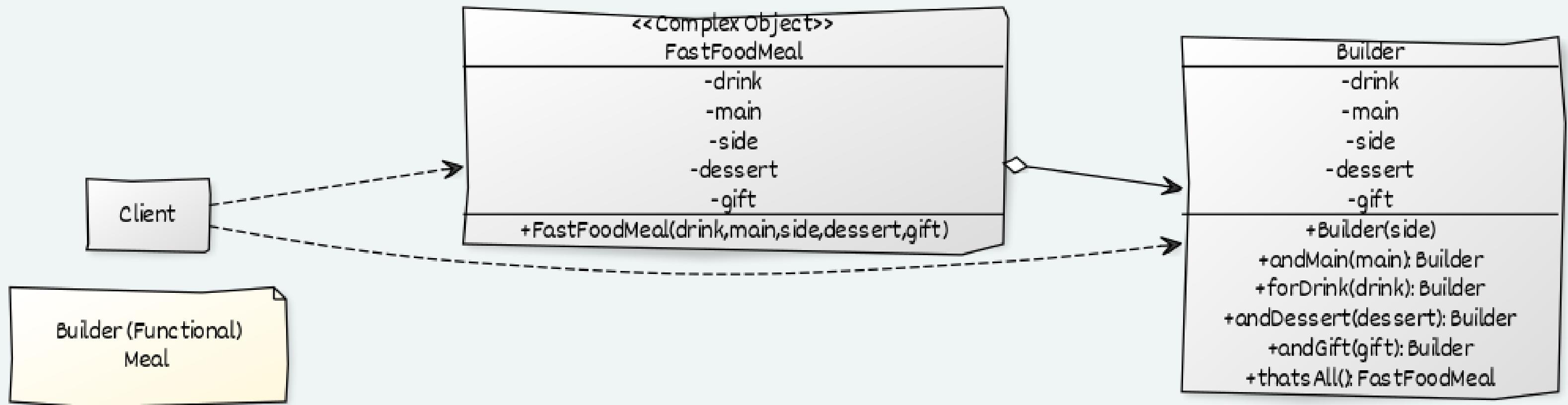






CREATED WITH YUML



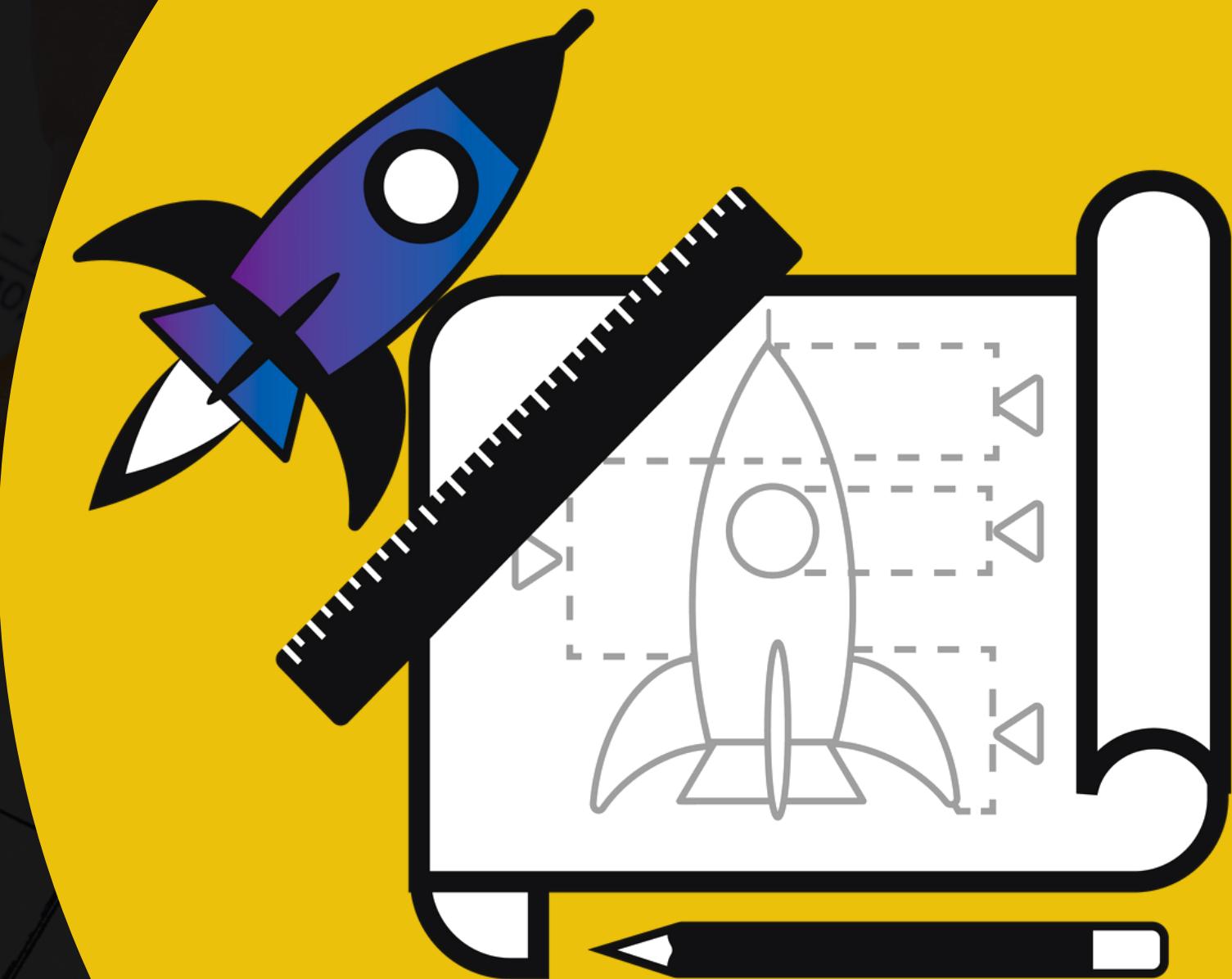


Sample Code

```
public class Client {  
  
    public static void main(String[] args) {  
        //Before  
        justFries = new FastFoodMeal(null, null, "Fries");  
  
        //After  
        justFries = order("Just Fries", new JustFriesBuilder());  
  
        //Fluent  
        justFries = new FastFoodMealBuilder("Fries").thatsAll();  
  
        //Functional  
        justFries = new FastFoodMeal.Builder("Fries").thatsAll();  
    }  
}
```

# AULA PROTOTYPE

RESUMINDO... CLONE!





## OBJETIVOS

- APRESENTAR O PROBLEMA GERAL
- APRESENTAR UMA SOLUÇÃO UTILIZANDO O PROTOTYPE
- APRESENTAR OS CUIDADOS QUE DEVE SE TER COM ESTA OPERAÇÃO
- APRESENTAR UMA IMPLEMENTAÇÃO QUE INTEGRA O BUILDER COM O PROTOTYPE PARA PROGRAMAÇÃO FUNCIONAL

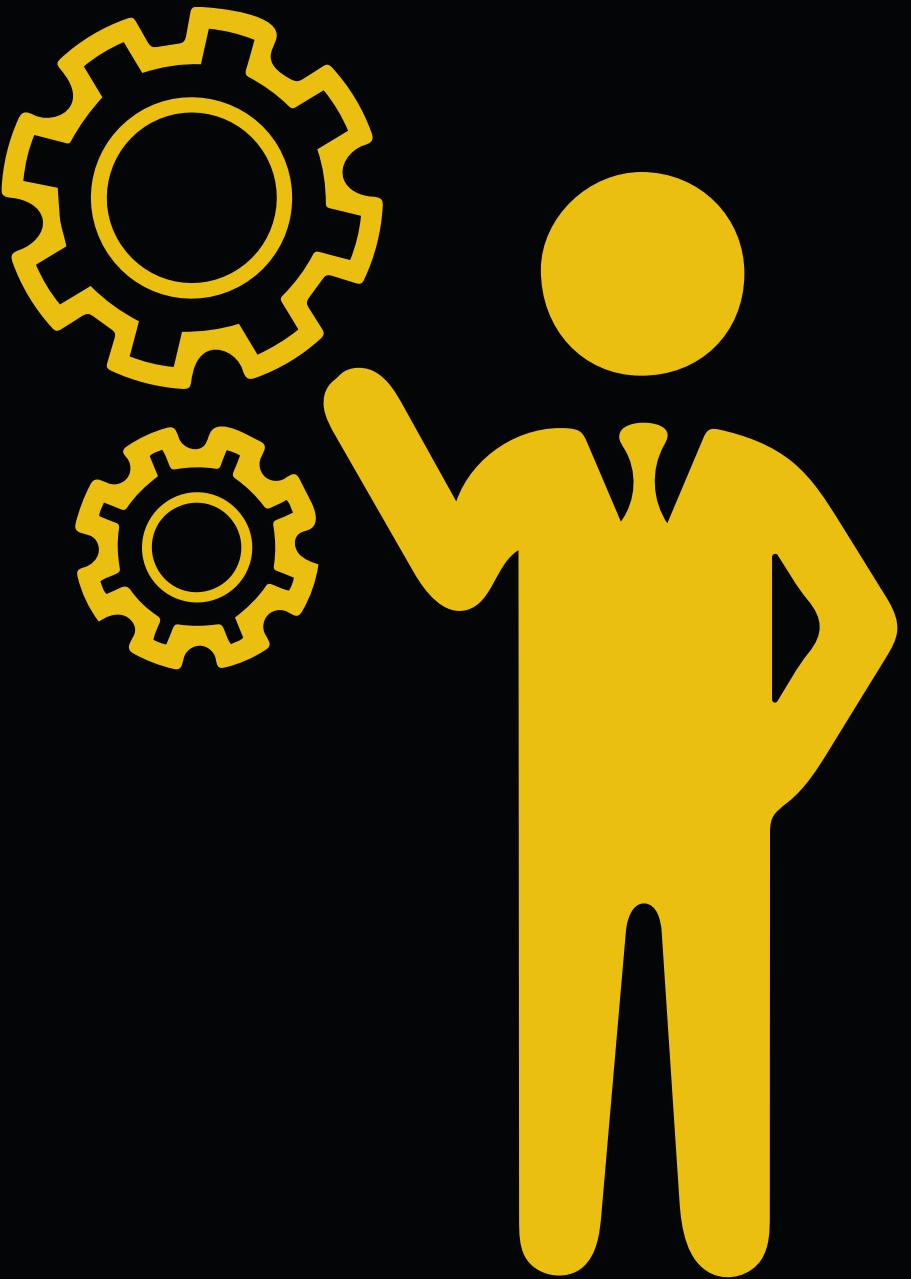
# PROBLEMAS

- COMO POSSO CRIAR UM OBJETO NOVO  
APROVEITANDO O ESTADO PREVIAMENTE  
EXISTENTE DE OUTRO OBJETO?



# SOLUÇÃO

- DEFINIR UM PROTOTYPE QUE RETORNE A CÓPIA DE SI MESMO

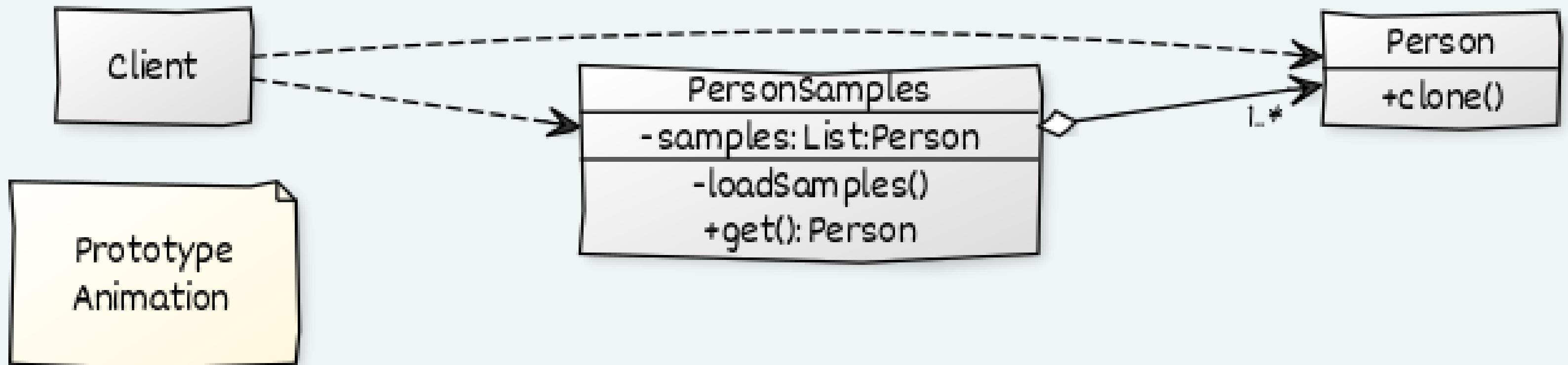




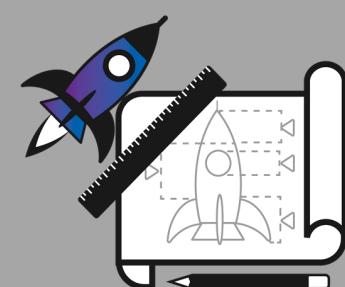
Especificar os tipos de objetos a serem criados usando uma instância como protótipo e criar novos objetos ao copiar este protótipo.

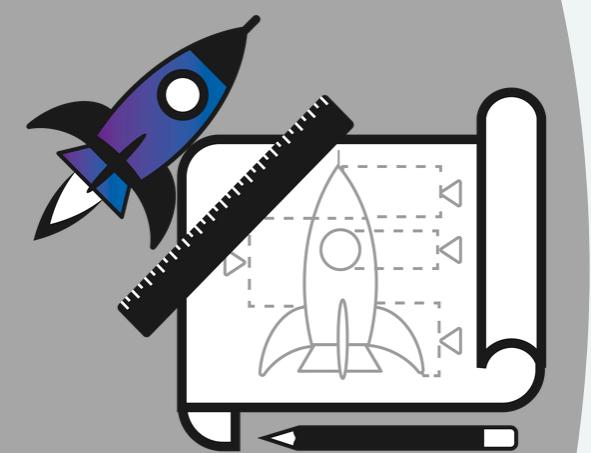
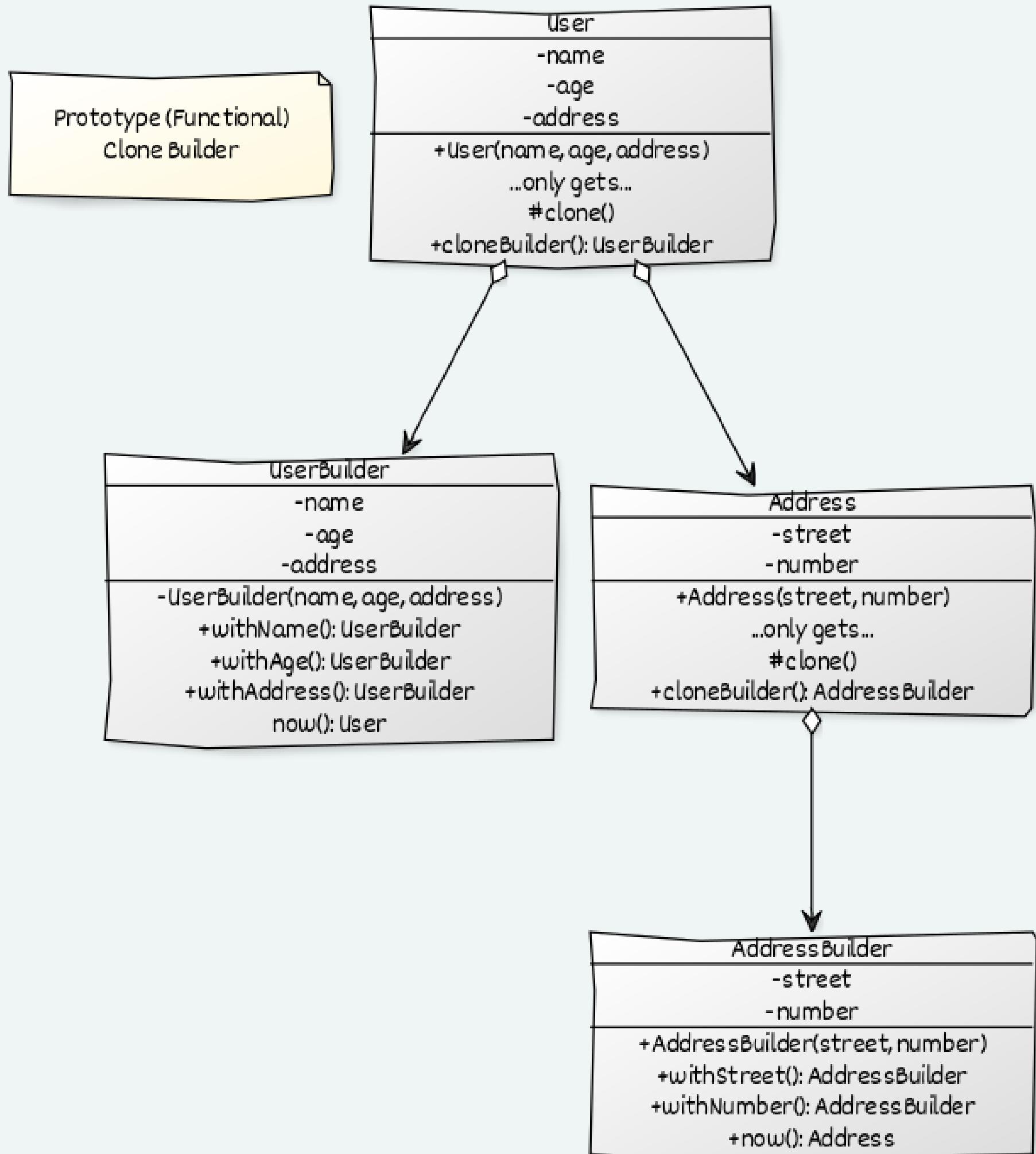
**GOF**





CREATED WITH YUML





# MÓDULO PADRÕES DE PROJETOS ESTRUTURAIS

COMO SERIA POSSÍVEL ESTRUTURAR DIVERSOS  
OBJETOS E CLASSES DE FORMA EXTENSÍVEL E  
FLEXÍVEL?

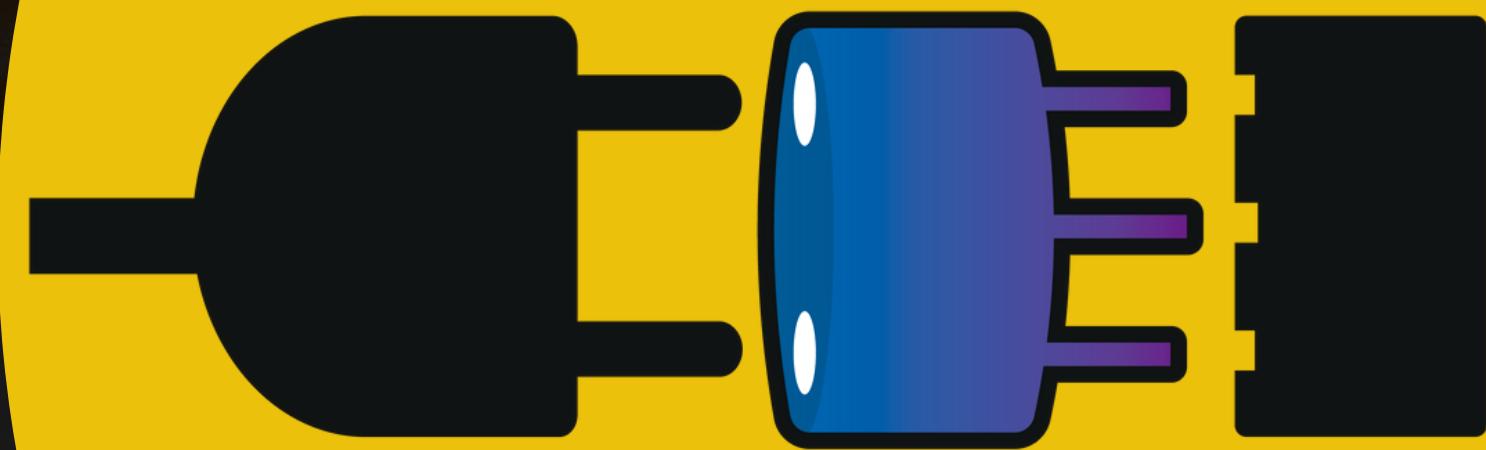


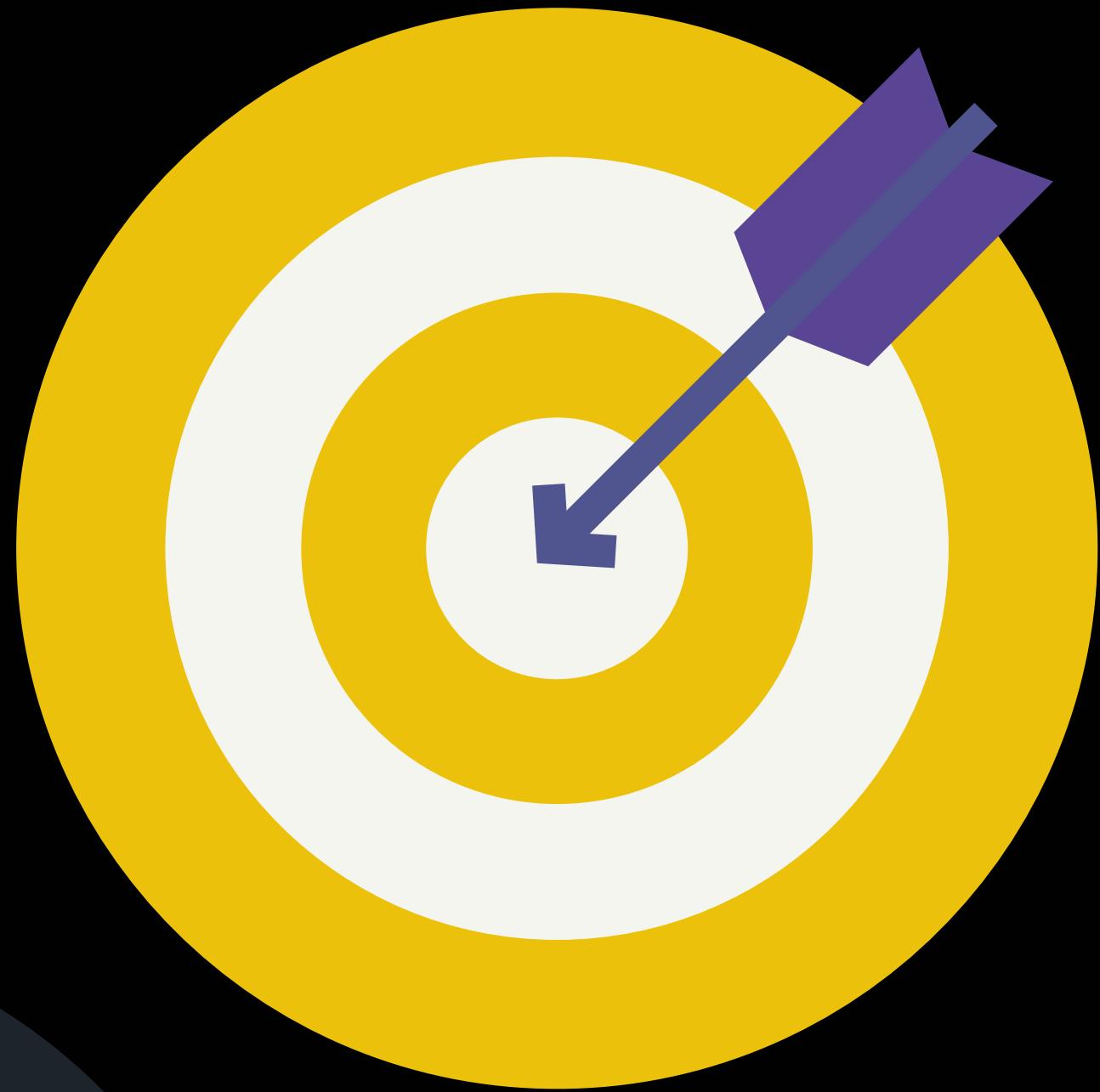
## OBJETIVOS DO MÓDULO

- O QUE SÃO PADRÕES ESTRUTURAIS
- ENTENDER SUA IMPORTÂNCIA
- CONHECER OS PRINCIPAIS PADRÕES DESTE TIPO
- ENTENDER SUAS DIFERENÇAS E RELACIONAMENTOS

# AULA ADAPTER

ADAPTANDO BEM, TODO MUNDO SE INTEGRA





## OBJETIVOS

- APRESENTAR O PROBLEMA GERAL
- APRESENTAR UMA SOLUÇÃO UTILIZANDO O ADAPTER
- APRESENTAR A DIFERENÇA ENTRE ADAPTERS DE OBJETO E CLASSE
- APRESENTAR COMO ESSE PADRÃO ESTÁ SENDO UTILIZADO EM ARQUITETURAS DE SISTEMAS MODERNOS

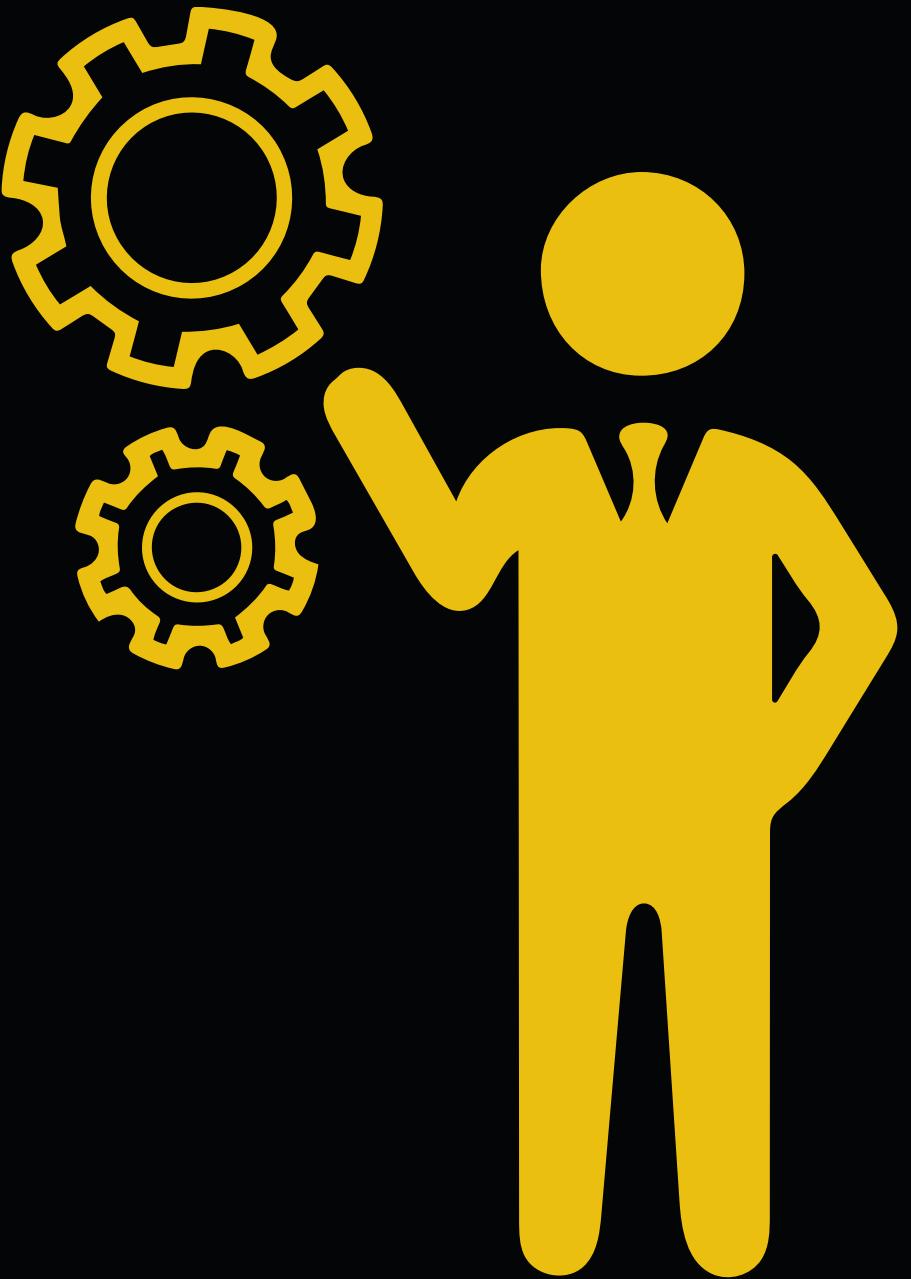
# PROBLEMAS

- COMO UMA CLASSE PODE SER REUSADA MESMO QUE NÃO TENHA A INTERFACE REQUISITADA PELO CLIENTE?
- COMO CLASSES DE INTERFACES INCOMPATÍVEIS PODEM TRABALHAR JUNTAS?



# SOLUÇÃO

- DEFINIR UMA CLASSE ADAPTER QUE CONVERTA A INTERFACE DE UMA CLASSE EM OUTRA QUE O CLIENTE NECESSITE

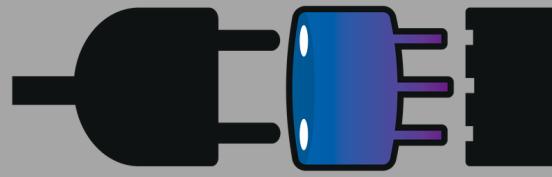




Converter a interface de uma classe em outra interface esperada pelos clientes. Adapter permite a comunicação entre classes que não poderiam trabalhar juntas devido à incompatibilidade de suas interfaces.

**GOF**





Adapter (Before)  
HDMI

