L’injection de dépendances

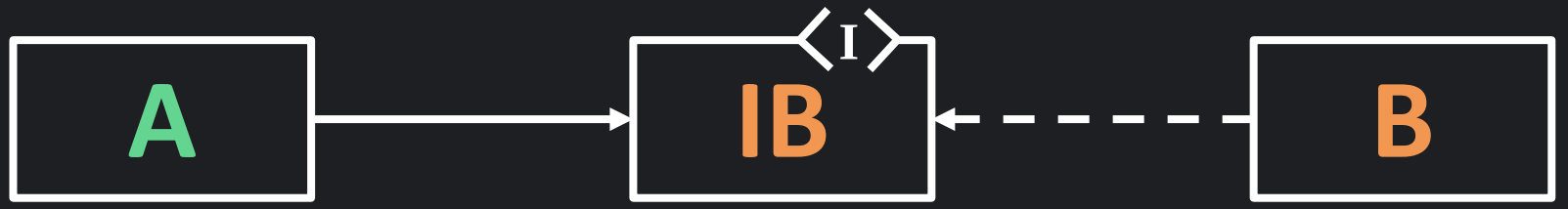
Patron d’entreprise

L’injection de dépendances

C’est quoi ?


A
B


A
B

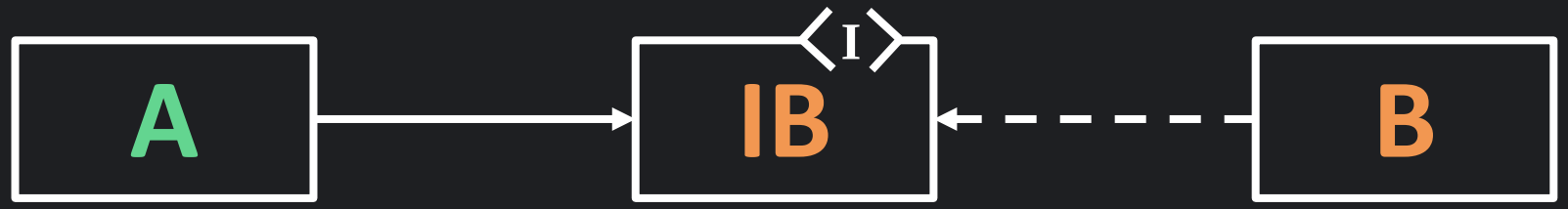

A
I
IB
B


```
Class A {
    B b;
    public A() {
        b = new B()
    }
}
```

```
Main() {
    A a = new A();
}
```


```
Class A {
    B b;
    public A() {
        b = new B()
    }
}
```

```
Main() {
    A a = new A();
}
```




```
Class A {
    IB b;
    public A(IB b) {
        this.b = b
    }
}
```

```
Main() {
    IB b = new B();
    A a = new A(b);
}
```

L'injection de dépendances consiste à fournir les objets dont un objet a besoin (ses dépendances) au lieu qu'il les construire lui-même

L’injection de dépendances

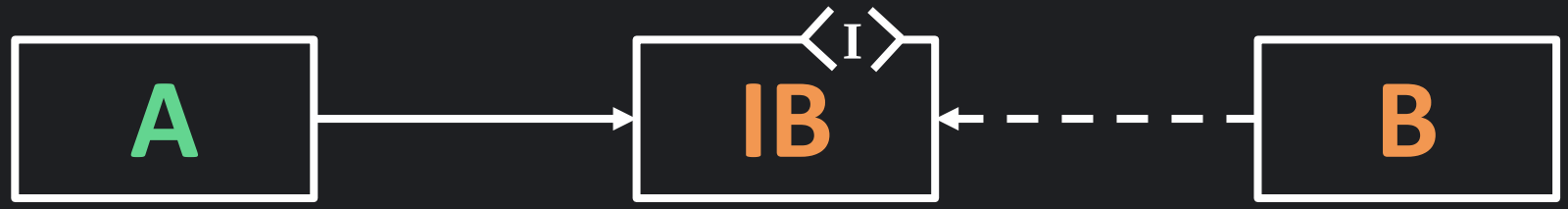
Pourquoi ?

Réduire les dépendances

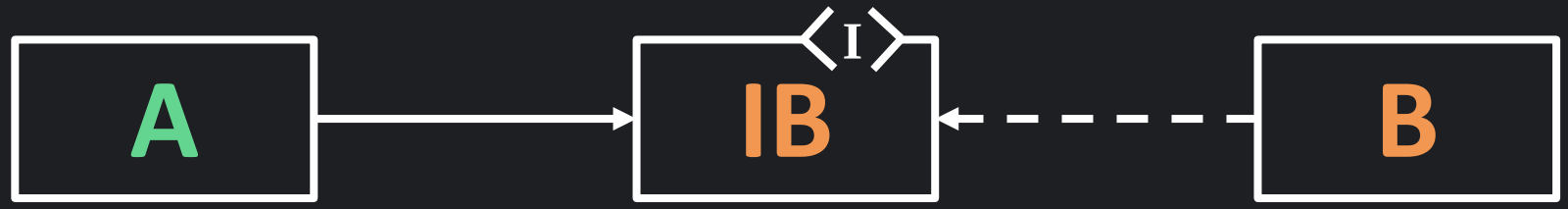
Réduire les dépendances

Si B est modifiée, alors B **ET** A devront être redéployées,
pourtant A n’a pas été modifiée

Réduire les dépendances

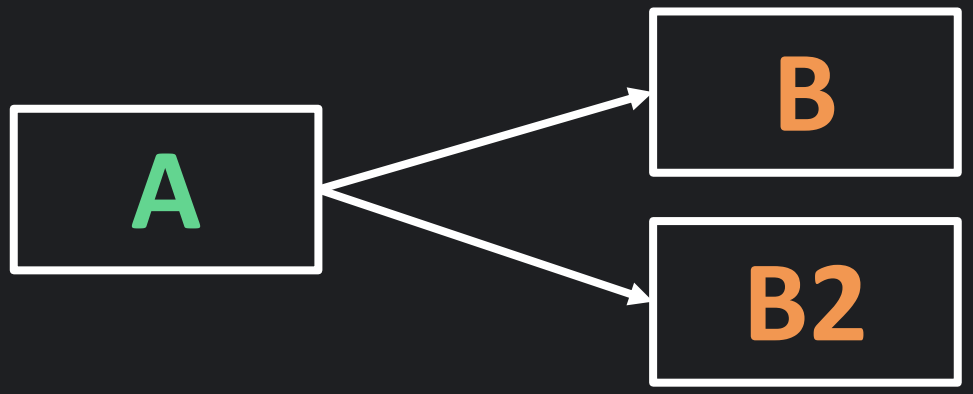


Réduire les dépendances



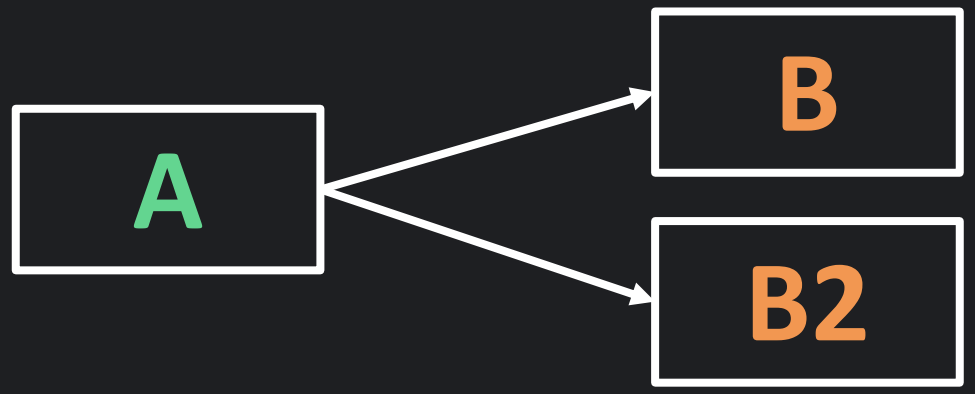
Si B est modifiée alors seulement B est redéployée, A ne dépend que de IB

Code plus flexible



Si je souhaite utiliser une autre classe que B, alors je vais devoir recoder A

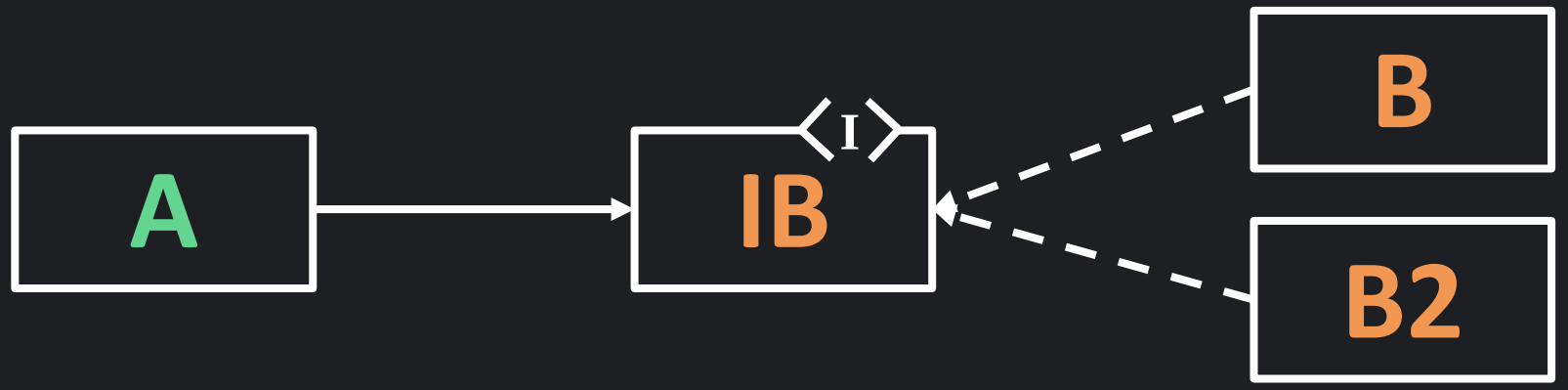
Code plus flexible




```
Class A {
    B b;
    public A() {
        b = new B()
    }
}
```

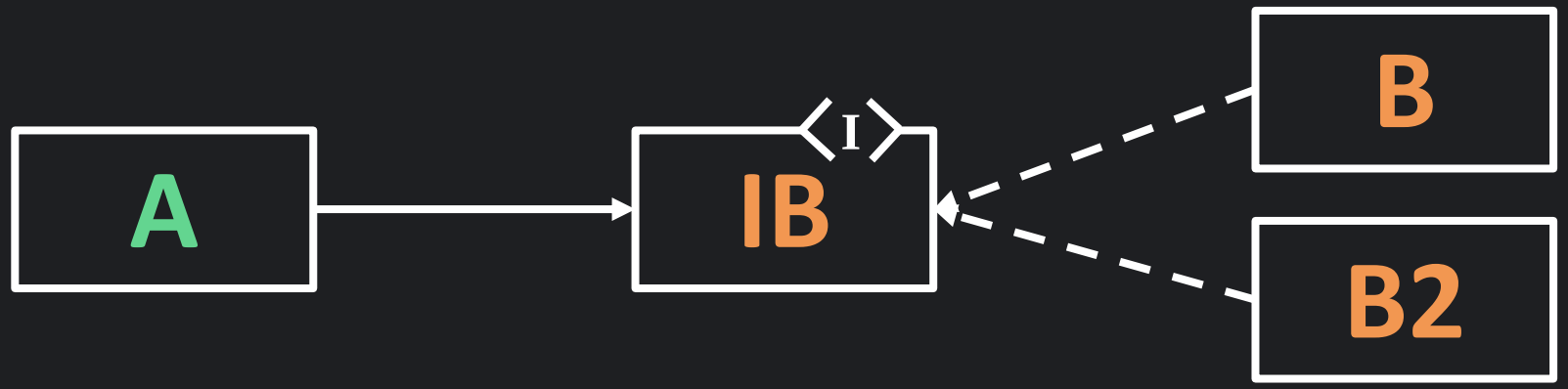
```
Class A {
    B2 b2;
    public A() {
        b2 = new B2()
    }
}
```

Code plus flexible



A dépend de IB, donc je peux utiliser n’importe quelle implémentation de IB sans affecter A

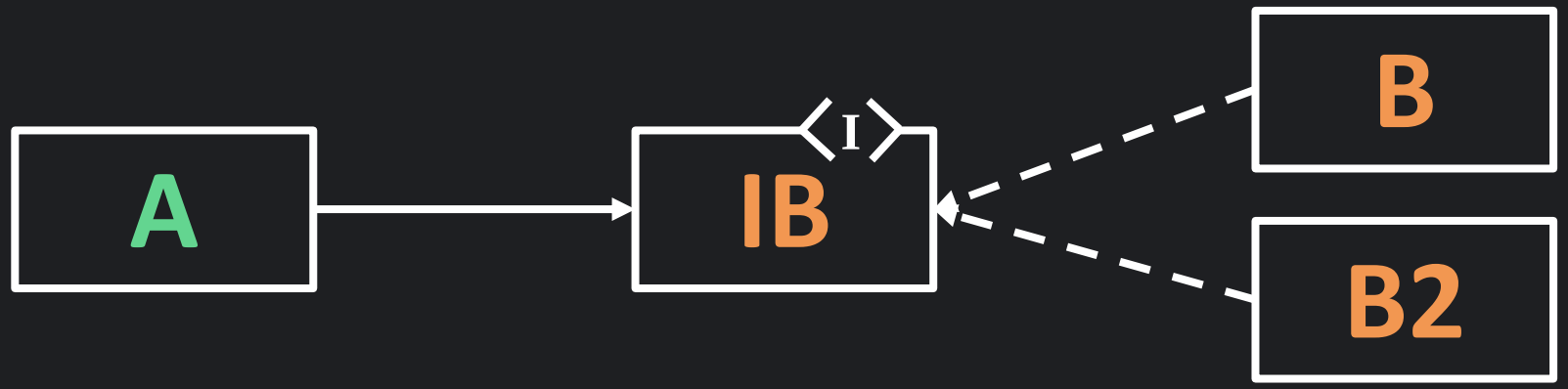
Code plus flexible




```
Class A {
    IB b;
    public A(IB b) {
        this.b = b
    }
}
```

```
Main() {
    IB b = new B();
    A a = new A(b);
}
```

Code plus flexible



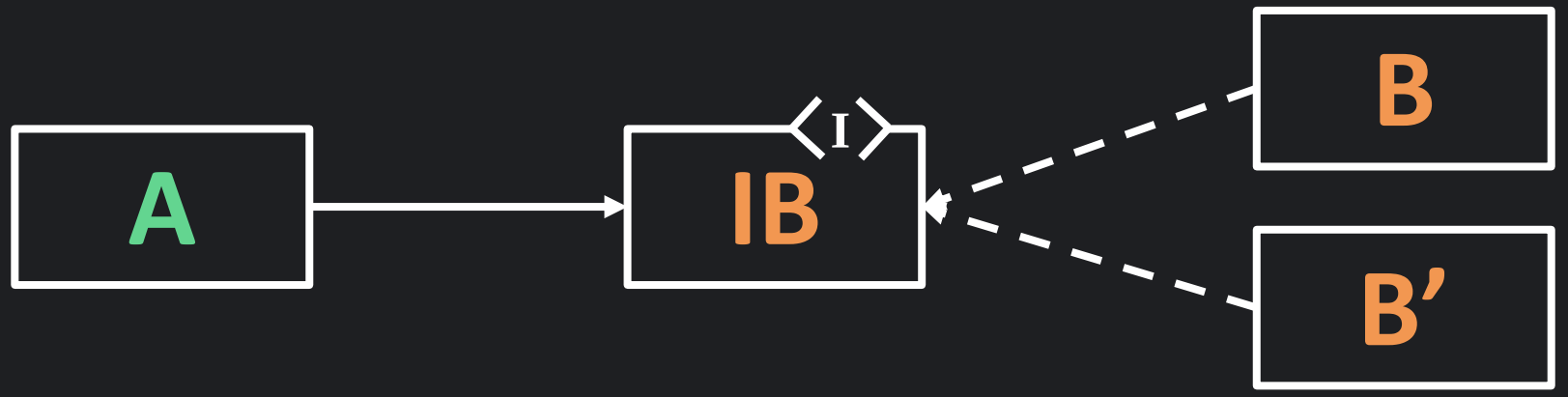

```
Class A {
    IB b;
    public A(IB b) {
        this.b = b
    }
}
```

```
Main() {
    IB b2 = new B2();
    A a = new A(b2);
}
```

Code plus testable

Comme A dépend de B, B doit être codée et testée en premier

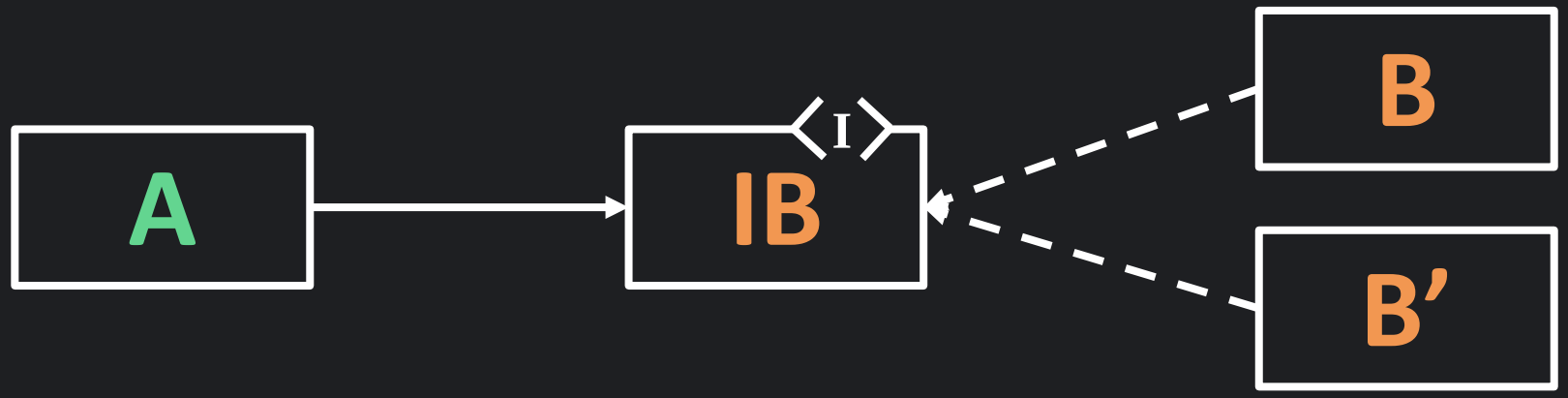
Code plus testable



Créer un bouchon B’ et tester A sans que B soit ni codée ni testée

Code plus testable

```
Public class B {
    public boolean b(){
        /* à coder */
    }
}
```



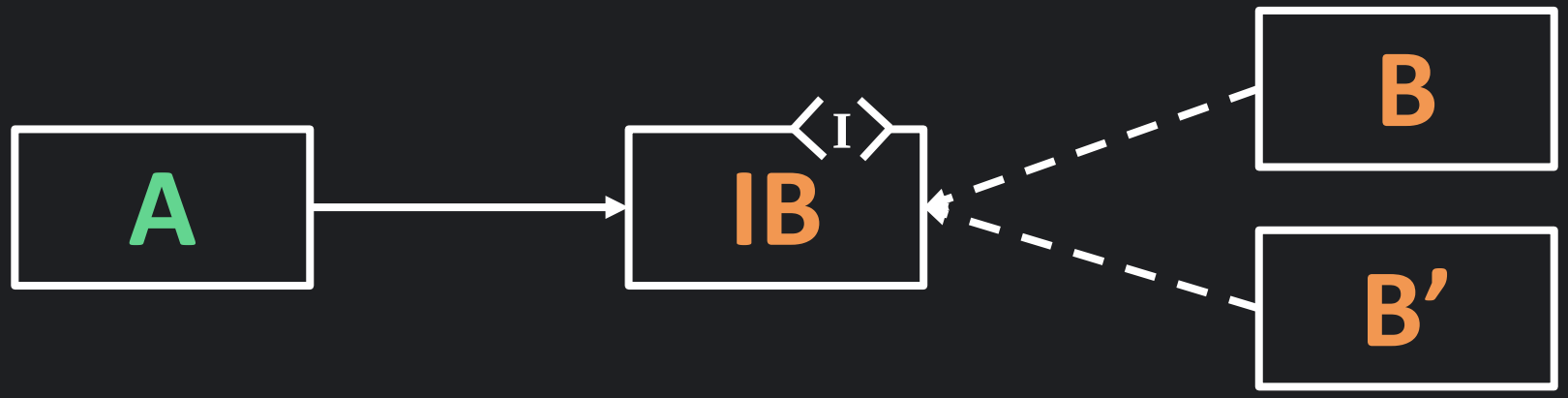

```
Public class A {
    void a() {
        if ( b() ) {
            print(1);
        } else {
            print(2);
        }
    }
}
```

```
Public class B’ {
    public boolean b(){
        return false;
    }
}
```

```
Main() {
    IB b = new B’();
    A a = new A(b);
}
```

Code plus testable

```
Public class B {
    public boolean b(){
        /* à coder */
    }
}
```



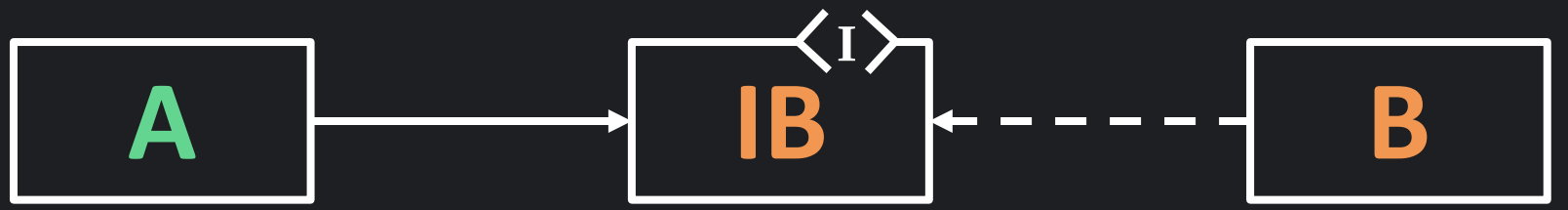

```
Public class A {
    void a() {
        if ( b() ) {
            print(1);
        } else {
            print(2);
        }
    }
}
```

```
Public class B’ {
    public boolean b(){
        return true;
    }
}
```

```
Main() {
    IB b = new B’();
    A a = new A(b);
}
```

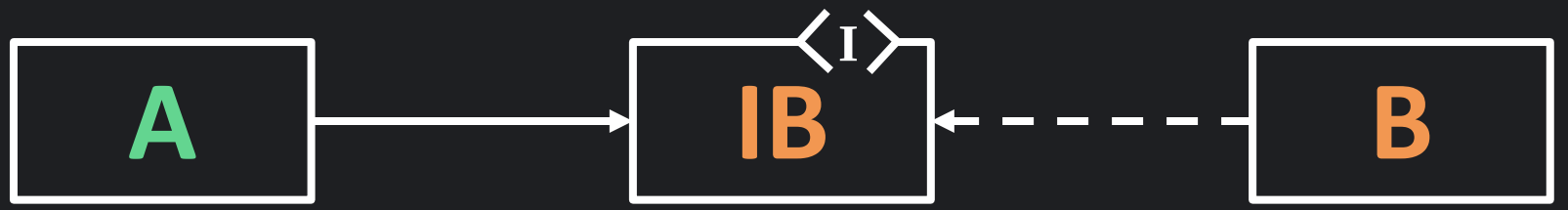
L’injection de dépendances

Comment ?

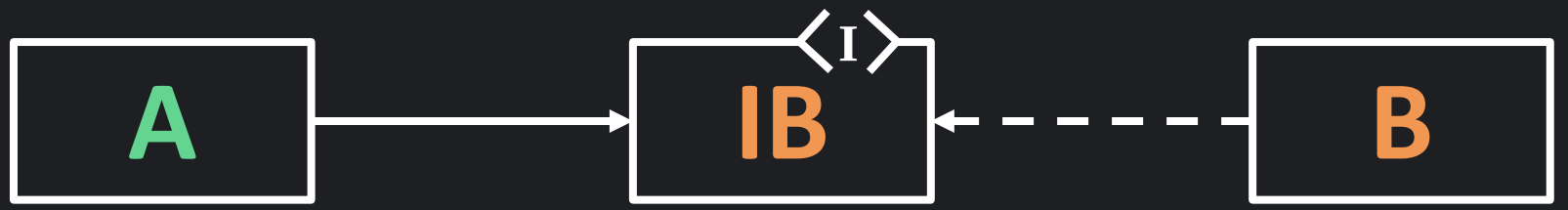



```
Class A {
    private IB b;


}
```

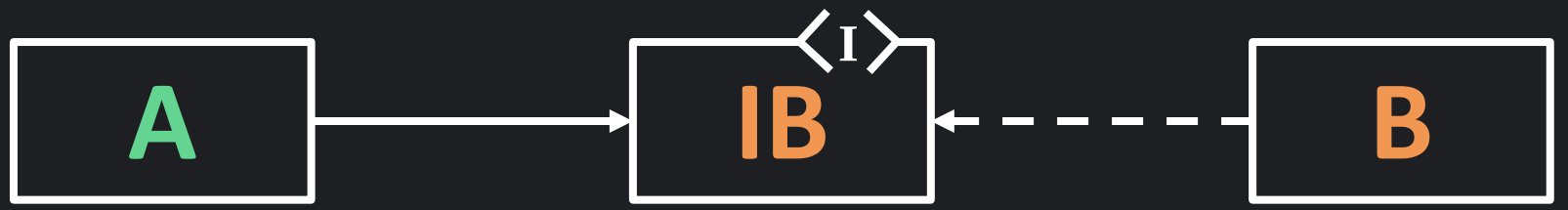



```
Class A {
    private IB b;
    public A(IB b) {
        this.b = b
    }
}
```




```
Class A {
     private IB b;
     public A(IB b) {
          this.b = b
     }
}
```

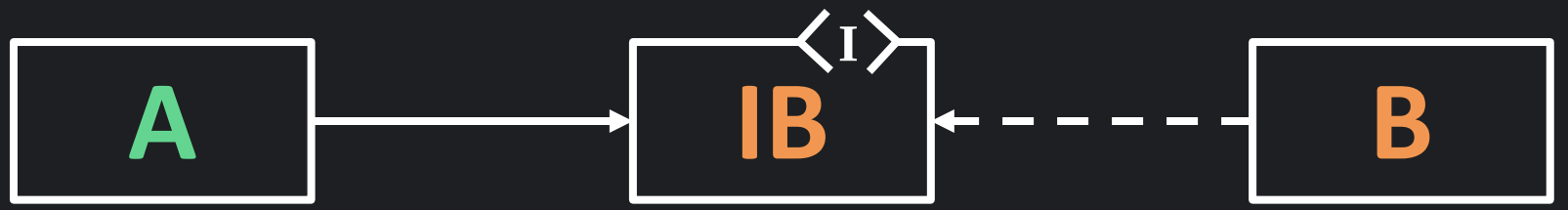
B est injecté dans A
au travers du
constructeur de A



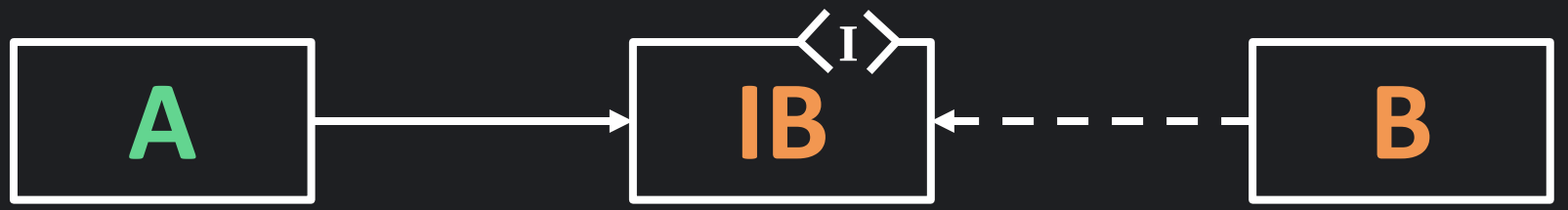

```
Class A {
    private IB b;
    public A(IB b) {
        this.b = b
    }
}
```

```
Main() {
   ① IB b = new B()

}
```

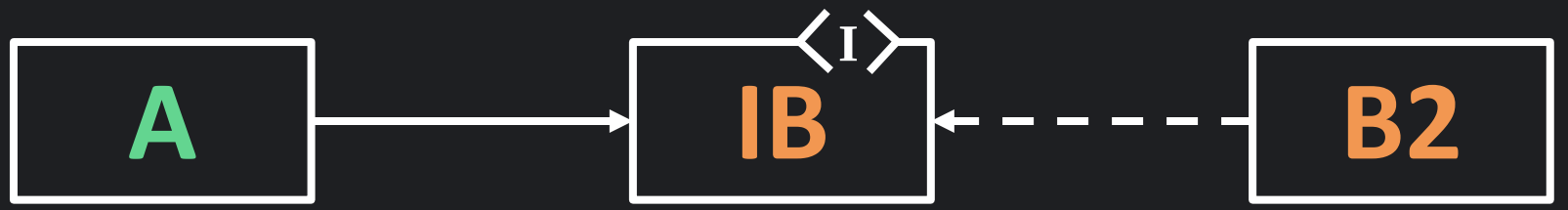



```
Class A {                          Main() {
    private IB b;
    public A(IB b) {
        this.b = b                 }
    }
}
```




```
Class A {
    private IB b;
    public A(IB b) {
        this.b = b
    }
}
```

```
Main() {
    ① IB b = new B()
    ② A a = new A(b)
}
```




```
Class A {
    private IB b;
    public A(IB b) {
        this.b = b
    }
}
```

```
Main() {
   ① IB b = new B2()
   ② A a = new A(b)
}
```

L’injection de dépendances

Conclusion

C’est rendre son code

C’est rendre son code

Libre

C’est rendre son code

Libre

Flexible

C’est rendre son code

Libre

Flexible

Indépendant

C’est rendre son code

Libre

Flexible

Indépendant