Inversion de contrôle

- L'inversion de contrôle peut être illustrée par le principe de Hollywood :

  « Ne nous appelez pas,
  c'est nous qui vous appellerons ».

Inversion de contrôle

- Selon ce principe, l'inversion de contrôle a lieu entre un framework (ou une couche logicielle sous-jacente) et une application.
- Ce n'est plus l'application qui gère les appels au framework, mais ce dernier à l'application.

Exemple : application qui demande des informations sur l'utilisateur

- Selon l'approche classique, le corps du programme pose non seulement les briques d'interaction Humain-Machine (IHM) mais contrôle aussi la séquence d'exécution de celles-ci.

```
print  'Votre Nom :'
read   Nom
traite Nom
print  'Votre âge :'
read   age
traite age
…
```

Exemple : application qui demande des informations sur l'utilisateur

- Avec l'inversion de contrôle, le framework s'assure du contrôle du flot d'exécution souvent sous la forme d'une fonction principale. Le corps principal du programme prend alors en charge seulement le traitement et, dans une moindre mesure, les briques IHM.

```
framework           ← init
q1                  ← question 'Votre nom :'
attache_traitement  q1, mon_code
q2                  ← question 'Votre âge :'
attache_traitement  q2, mon_autre_code
boucle_principale   framework
```

Injection de dépendances

- L'injection de dépendances est un mécanisme qui consiste à créer dynamiquement (injecter) les dépendances entre les différents objets en s'appuyant sur une description (fichier de configuration ou métadonnées) ou par programme. Ainsi les dépendances entre composants logiciels ne sont plus exprimées dans le code de manière statique mais déterminées dynamiquement à l'exécution.

Dépendance

- Une classe A dépend d'une classe B si au moins une des conditions suivantes est vérifiée :
  - A possède un attribut de type B (dépendance par composition) ;
  - A est de type B (dépendance par héritage) ;
  - A dépend d'une autre classe C qui dépend de B (dépendance par transitivité) ;
  - une méthode de A appelle une méthode de B.
  - une méthode de A utilise la classe B (type de paramètre, valeur de retour, variable locale, appel de méthode de la classe B)

Supprimer la dépendance

- Si A dépend de B, on a besoin de B pour instancier A. Mais en pratique, on n'a pas toujours B.
- Pour supprimer la dépendance, un moyen possible consiste à
  - créer une interface I qui contiendra toutes les méthodes que A peut appeler sur B,
  - indiquer que B implante l'interface I,
  - remplacer toutes les références à B par des références à l'interface I dans A.

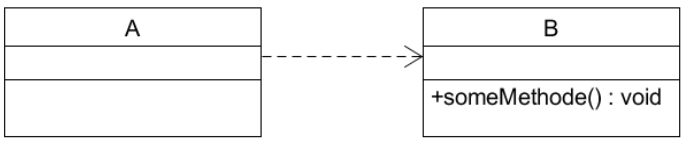
Supprimer la dépendance…

- Un problème qui se pose est de disposer dans A d'un objet implantant I alors que l'on ne sait pas comment l'instancier.
- La solution consiste à créer, par exemple, un objet b de type B et de l'injecter dans un objet de type A. L'injection proprement dite peut se faire :
  - à l'instanciation : on passe l'objet b à l'instanciation de A
  - par modificateur : on passe l'objet b à une méthode de A qui va par exemple modifier un attribut (setter)

Mise en œuvre de l'injection de dép.

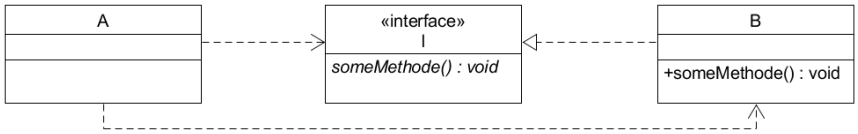
- Créer une interface I déclarant les méthodes de la classe B utilisées par la classe A ;
- Déclarer la classe B comme implantation de cette interface I ;
- Remplacer toute référence à la classe B par des références à l'interface I ;
- Si la classe A instancie des instances de B à son initialisation, alors remplacer l'instanciation par un passage d'une instance de l'interface I au(x) constructeur(s) de A ;
- Si besoin, ajouter une méthode pour spécifier l'instance de l'interface I à utiliser.

Exemple



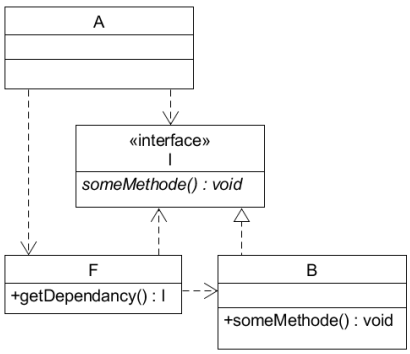

```
public class A {
  private B b;
  public A() { b = new B(); }
  void doWork() {
    b.someMethod();
  }
}
```

Interface(s) ?




```
public class A {
  private I i;
  public A() { i = new B(); }
  void doWork() {
    i.someMethod();
  }
}
```

Fabrique(s) ?




```
public class Factory {
  public I getDependency()
    return new B();
  }
}
```

```
public class A {
  private I i;
  public A() { i = new factory().getDependency(); }
  void doWork() {
    i.someMethod();
  }
}
```

Injection par constructeur

```
public class A {
  private I i;
  public A(I i) { this.i = i; }
  void doWork() { i.someMethod(); }
}
```

```
public class Assemblage {
  private A a;
  public Assemblage() {
    a = new A(new factory().getDependency());
  }
  public doWork() { a.doWork(); }
}
```

Injection d'interface

```
public interface Inject {
  void inject(I i);
}
```

```
public class A
implements Inject {
  private I i;
  public A() { }
  public void inject(I i) {
    this.i = i;
  }
  public doWork() {
    a.doWork();
  }
}
```

```
public class Assemblage {
  private A a;
  public Assemblage() {
    a = new A();
    a.inject(new B());
  }
  public doWork() {
    a.doWork();
  }
}
```

Injection par transformateur

- public class A {
  private I i;
  public A() { }
  public void setDep(I i) {
  this.i = i;
  i.doWork();
  }
  }
- public class Assemblage {
  private A a;
  public Assemblage() {
  a = new A();
  a.setDep (new B());
  }
  }