



Voyage en Legacy avec le Kata Trip Service

Isabelle BLASQUEZ
@iblasquez

Un Kata sur du code legacy

Contexte métier : Réseau Social pour voyageurs

Règles métier:

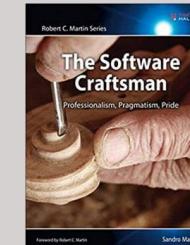
- Il faut être connecté(e) pour accéder au contenu du site.
- Il faut faire partie des ami(e)s d'un utilisateur pour voir ses voyages.

Objectif : Tester *puis* refactorer le code legacy

Idée originale de Sandro Mancuso :

<https://github.com/sandromancuso/trip-service-kata>

(existe pour différents langages)

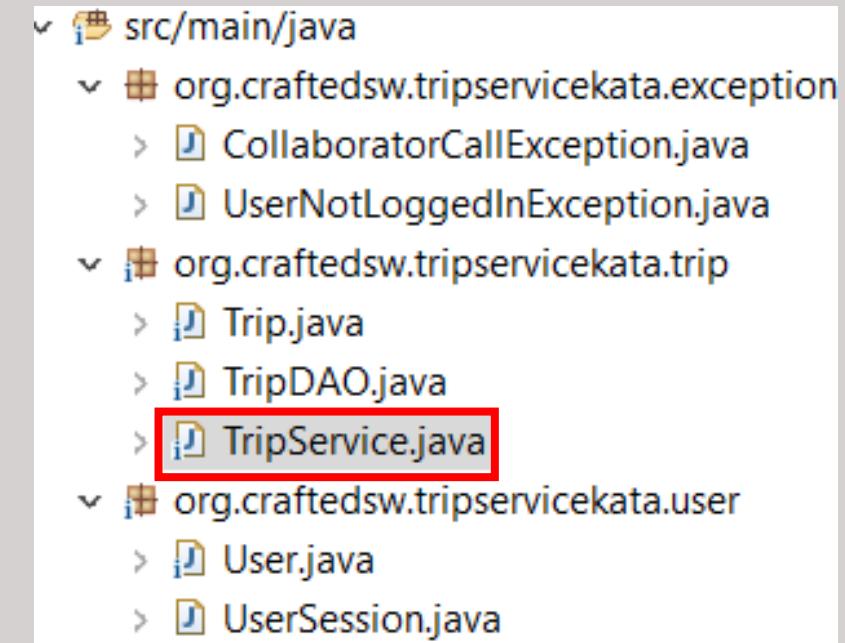


Code Legacy

```
public class TripService {
```

```
    public List<Trip> getTripsByUser(User user) throws UserNotLoggedInException {
        List<Trip> tripList = new ArrayList<Trip>();
        User loggedUser = UserSession.getInstance().getLoggedUser();
        boolean isFriend = false;
        if (loggedUser != null) {
            for (User friend : user.getFriends()) {
                if (friend.equals(loggedUser)) {
                    isFriend = true;
                    break;
                }
            }
            if (isFriend) {
                tripList = TripDAO.findTripsByUser(user);
            }
        }
        return tripList;
    } else {
        throw new UserNotLoggedInException();
    }
}
```

Pour la phase de couverture par les tests
Se focaliser uniquement sur la classe **TripService**





Première Partie (Couverture du code legacy par des tests)

Mettre en place
un harnais de tests
pour garantir
le *bon* comportement
du produit

**Pas de refactoring,
sans test !**

Mettre en place un harnais de tests



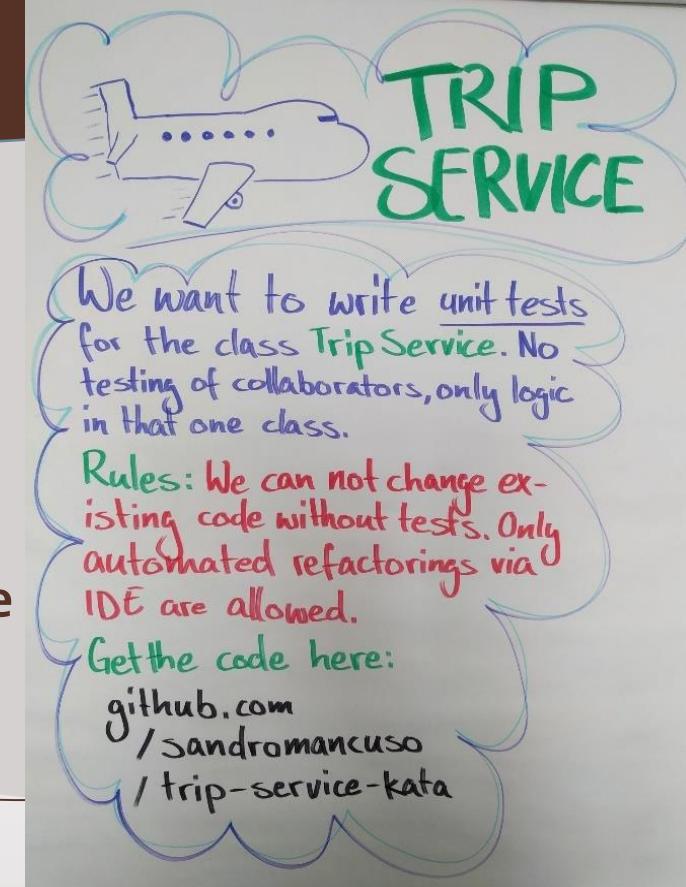
100% du code legacy actuel couvert
c'est-à-dire

100% de couverture de code par les tests
sur la méthode `getTripsByUser` de la classe `TripService`

Règles

Pas de refactoring sur un code de production non couvert par un test !

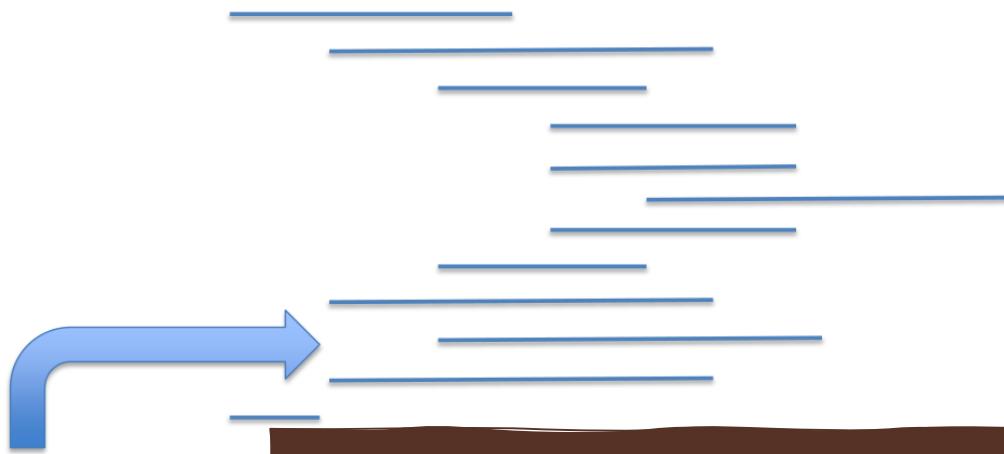
Exception : refactoring automatisés proposés par votre IDE
possible sur le code de production pendant la phase de couverture par les tests



Par où commencer ? ...

Petit conseil de Sandro pour écrire le premier test

Working with Legacy Code Tips



Start testing from
shortest to
deepest branch

Bonne pratique :

*Commencer par tester le comportement obtenu
lors du plus court chemin d'exécution et remo*

⇒ branche de code la moins profonde

c-a-d branche la plus à gauche qui nécessite moins de préparation pour y accéder

Par où commencer ? ... Sur quelle partie du code focaliser le premier test ?



```
public class TripService {
```

```
    public List<Trip> getTripsByUser(User user) throws UserNotLoggedInException {
        List<Trip> tripList = new ArrayList<Trip>();
        User loggedUser = UserSession.getInstance().getLoggedUser();
        boolean isFriend = false;
        if (loggedUser != null) {
            for (User friend : user.getFriends()) {
                if (friend.equals(loggedUser)) {
                    isFriend = true;
                    break;
                }
            }
            if (isFriend) {
                tripList = TripDAO.findTripsByUser(user);
            }
        }
        return tripList;
    } else {
        throw new UserNotLoggedInException();
    }
}
```

⇒ branche de code la moins profonde
chemin d'exécution le plus court

⇒ Exception levée
si l'utilisateur N'est PAS connecté



Test n°1

Comportement
à couvrir :

Exception levée
quand
l'utilisateur n'est connecté

```
public class TripServiceTest {
```

```
@Test (expected = UserNotLoggedInException.class)
public void should_throw_an_exception_when_user_is_not_logged_in(){
```

```
TripService tripService = new TripService();
```

```
tripService.getTripsByUser(null);
```

null car peu importe la valeur du **User** passée en paramètre de cette méthode, elle ne sera jamais utilisée dans le code qui permet de déterminer le comportement du test ...

Pour ce test, seul importe que la valeur de `userLogged` dans `getTripService` soit bien à `null` pour simuler un utilisateur non connecté 😊

Attention : `getTripsByUser` manipule bien deux utilisateurs distincts : `userLogged` ≠ `user`

(local) (par un d'entrée)

→ Mais à l'exécution ...



Utilisateur « curieux » connecté ou non

Utilisateur potentiellement un *ami* passé en paramètre



≡ Failure Trace

```
java.lang.Exception: Unexpected exception, expected<org.craftedsw.tripservicekata.exception.UserNotLoggedInException> but was<org.craftedsw.tripservicekata.exception.CollaboratorCallException>
Caused by: org.craftedsw.tripservicekata.exception.CollaboratorCallException: UserSession.getLoggedUser() should not be called in an unit test
at org.craftedsw.tripservicekata.user.UserSession.getLoggedUser(UserSession.java:17)
at org.craftedsw.tripservicekata.trip.TripService.getTripsByUser(TripService.java:14)
at org.craftedsw.tripservicekata.trip.TripServiceTest.should_show_an_exception_when_user_is_not_logged_in(TripServiceTest.java:11)
... 15 more
```



Une idée pour que l'exception `CollaboratorCallException` ne soit pas levée lors de l'exécution de ce test?

```
public class TripService {  
  
    public List<Trip> getTripsByUser(User user) throws UserNotLoggedInException {  
        List<Trip> tripList = new ArrayList<Trip>();  
        User loggedUser = UserSession.getInstance().getLoggedUser();  
        boolean isFriend = false;  
        if (loggedUser != null) {  
            for (User friend : user.getFriends()) {  
                if (friend.equals(loggedUser)) {  
                    isFriend = true;  
                    break;  
                }  
            }  
            if (isFriend) {  
                tripList = TripDAO.findTripsByUser(user);  
            }  
        }  
        return tripList;  
    } else {  
        throw new UserNotLoggedInException();  
    }  
}
```

```
public class UserSession {  
    //...  
  
    public User getLoggedUser() {  
        throw new  
CollaboratorCallException("UserSession.  
getLoggedUser() should not be called in  
an unit test");  
    }  
}
```

On voudrait avoir `null` pour `loggedUser`
mais une exception est levée
(métaphore d'accès à un service tiers
sur lequel on n'a pas la main...)

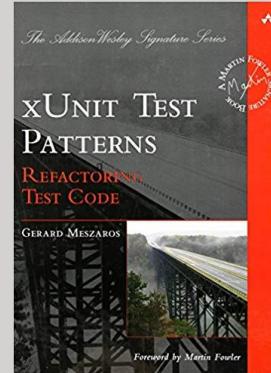
Rappel règle du kata :
on ne doit pas toucher au code de
production tant que ce dernier n'est
pas couvert par mes tests

Des idées pour isoler la dépendance ?



Mettre en place une doublure de tests (*mock*)

- **Stub** provide canned answers to calls made during the test, usually not responding at all to anything outside what's programmed in for the test.
- Doublure implémentée via [mockito](#)



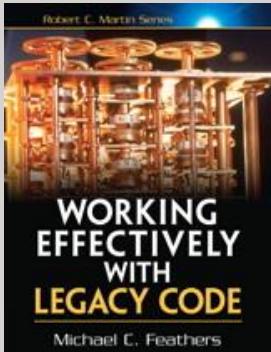
Mais Mockito ne permet de mocker ni méthode statique, ni singleton ;-)



Mettre en place une couture (Seam)

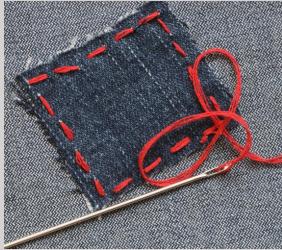


A **seam** is a place where you can alter behavior in your program without editing in that place.



Zoom sur la couture (Seam)

A **seam** is a place where you can alter behavior in your program without editing in that place. Every seam has an **enabling point**, a place where you can make the decision to use one behavior or another.



La couture, en ajoutant une fine couche (*thin layer*), va permettre d'**isoler une dépendance** et de contrôler son comportement pendant la phase de **couverture de tests** pour avoir le contrôle total du SUT

La mise en place d'un Seam va permettre au code legacy de rester compatible pour la production tout en l'ouvrant pour les tests...

(c-a-d que le code de production ne sera ni modifié, ni cassé 😊)



Mais la couture n'est pas une fin en soi...

Sa présence n'a pour but que de vous permettre de tester votre code. Pendant la phase de **refactoring**, il faudra faire en sorte de **casser cette dépendance** (et donc **surement** revisiter les tests pour supprimer la couture à ce moment là)



Mettre en place
la couture (Seam)

pour isoler la dépendance
au Singleton

Etapes pour mettre en place le Seam Isoler la dépendance et forcer son comportement



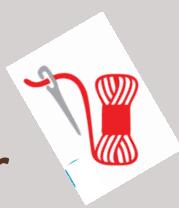
1. Dans le code de production (`TripService`)



- ✓ Isoler le code qui appelle la dépendance (à l'aide de l'Extract Method de l'IDE)
⇒ L'extraire dans une méthode `getLoggedInUser` de type `protected`

2. Dans le code test, procéder à la couture avec l'aide d'une polymorphisme

- ✓ Créer une « *thin layer* » avec une nouvelle classe privée `TestableTripService` qui hérite de `TripService`
- ✓ Redéfinir la méthode `getLoggedInUser` dans `TestableTripService` et bouchonner son comportement (`null`)
- ✓ Instancier une classe `TestableTripService` pour que le code passe désormais par la *thin layer*



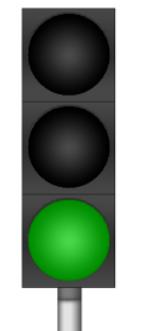
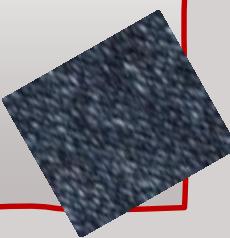
Isoler la dépendance au Singleton

Et faire passer le test !

```
public class TripServiceTest {  
  
    @Test(expected = UserNotLoggedInException.class)  
    public void should_throw_an_exception_when_user_is_not_logged_in(){  
        TripService tripService = new TestableTripService();  
        tripService.getTripsByUser(null);  
    }  
}
```

```
private class TestableTripService extends TripService {  
    @Override  
    protected User getLoggedInUser() {  
        return null;  
    }  
}
```

```
public class TripService {  
  
    public List<Trip> getTripsByUser(User user)  
        throws UserNotLoggedInException {  
        //...  
        User loggedUser = getLoggedInUser();  
        //...  
    }  
  
    protected User getLoggedInUser() {  
        return  
            UserSession.getInstance().getLoggedUser();  
    }  
}
```



Organisation de l'espace de travail et petit coup d'œil sur la couverture de code

The screenshot shows an IDE interface with two code editors side-by-side. The left editor contains `TripServiceTest.java` and the right editor contains `TripService.java`. The `TripService.java` code is annotated with colored highlights:

- Green:** `List<Trip> tripList = new ArrayList<Trip>();`, `User loggedUser = getLoggedInUser();`, `boolean isFriend = false;`, `if (loggedUser != null) {`, `for (User friend : user.getFriends()) {`, `if (friend.equals(loggedUser)) {`, `isFriend = true;`, `tripDAO.findTripsByUser(user);`, `return tripList;`, `throw new UserNotLoggedInException();`, `UserSession.getInstance().getLoggedUser();`.
- Yellow:** `if (loggedUser != null) {`, `for (User friend : user.getFriends()) {`, `if (friend.equals(loggedUser)) {`, `isFriend = true;`, `break;`, `}`, `}`, `if (isFriend) {`.
- Red:** `public List<Trip> getTripsByUser(User user) throws UserNotLoggedInException {`, `if (loggedUser == null) {`, `else {`, `}`, `return tripList;`, `}`, `else {`, `throw new UserNotLoggedInException();`, `}`, `}`, `else {`, `return UserSession.getInstance().getLoggedUser();`.

```
File Edit Source Refactor Navigate Search Project Run Window Help
Quick Access

TripServiceTest.java
1 package org.craftedsw.tripservicekata.trip;
2
3 import org.craftedsw.tripservicekata.exception.UserNotLoggedInException;
4 import org.craftedsw.tripservicekata.user.User;
5 import org.junit.Test;
6
7 public class TripServiceTest {
8
9     @Test(expected = UserNotLoggedInException.class)
10    public void should_throw_an_exception_when_user_is_not_logged_in() {
11        TripService tripService = new TestableTripService();
12        tripService.getTripsByUser(null);
13    }
14
15    private class TestableTripService extends TripService {
16        @Override
17        protected User getLoggedInUser() {
18            return null;
19        }
20    }
21
22 }

TripService.java
1 package org.craftedsw.tripservicekata.trip;
2
3 import java.util.ArrayList;
4 import java.util.List;
5
6 import org.craftedsw.tripservicekata.exception.UserNotLoggedInException;
7 import org.craftedsw.tripservicekata.user.User;
8 import org.craftedsw.tripservicekata.user.UserSession;
9
10 public class TripService {
11
12     public List<Trip> getTripsByUser(User user) throws UserNotLoggedInException {
13         List<Trip> tripList = new ArrayList<Trip>();
14         User loggedUser = getLoggedInUser();
15         boolean isFriend = false;
16         if (loggedUser != null) {
17             for (User friend : user.getFriends()) {
18                 if (friend.equals(loggedUser)) {
19                     isFriend = true;
20                     break;
21                 }
22             }
23             if (isFriend) {
24                 tripList = tripDAO.findTripsByUser(user);
25             }
26             return tripList;
27         } else {
28             throw new UserNotLoggedInException();
29         }
30     }
31
32     protected User getLoggedInUser() {
33         return UserSession.getInstance().getLoggedUser();
34     }
35
36 }

! BLASQUEZ
```



→ Un peu de refactoring sur le code de test est permis et même fortement conseillé



Sentez-vous
une mauvaise odeur
dans ce code ?



Améliorer la lisibilité des tests

Faire apparaître
des termes métiers relatif au
concept d'utilisateur :

*pour rendre explicite le rôle
que joue chaque utilisateur
dans le comportement
du programme*

Rappel :

*2 utilisateurs peuvent influencer
le comportement du programme*



Utilisateur « curieux »
connecté ou non



utilisateur potentiellement un *ami*
de l'utilisateur curieux

```
public class TripServiceTest {  
  
    @Test(expected = UserNotLoggedInException.class)  
    public void should_throw_an_exception_when_user_is_not_logged_in(){  
        TripService tripService = new TestableTripService();  
        tripService.getTripsByUser(null);  
    }  
  
    private class TestableTripService extends TripService {  
        @Override  
        protected User getLoggedInUser() {  
            return null;  
        }  
    }  
}
```



Refactoring : Améliorer la lisibilité des tests

Zoom sur null de getLoggedInUser()

```
public class TripServiceTest {  
  
    @Test(expected = UserNotLoggedInException.class)  
    public void should_throw_an_exception_when_user_i  
        TripService tripService = new TestableTripService();  
        tripService.getTripsByUser(null);  
    }  
  
    private class TestableTripService extends TripSe  
        @Override  
        protected User getLoggedInUser() {  
            return null;  
        }  
    }  
}
```

Se rapporte à  Utilisateur « curieux » non connecté (pour ce test)

Dans ce test, l'utilisateur n'est pas connecté.

Cet état de l'utilisateur a un impact sur le comportement du programme donc sur le *verdict du test* (le *Assert* du test).

Cet état devrait donc être explicitement spécifier dans le test, et plus précisément au moment où on définit le contexte du test c-a-d dans la partie *Arrange*

Rappel Bonne Pratique :
Un test bien écrit
respecte le pattern AAA !



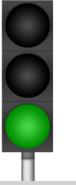
Etapes de refactoring pour faire apparaître Explicitement que l'utilisateur n'est pas connecté

```
public class TripServiceTest {  
    private User loggedInUser;  
  
    @Test(expected = UserNotLoggedInException.class)  
    public void should_throw_an_exception_when_user_is_not_logged_in(){  
        loggedInUser = null;  
        TripService tripService = new TestableTripService();  
        tripService.getTripsByUser(null);  
    }  
  
    private class TestableTripService extends TripService {  
        @Override  
        protected User getLoggedInUser() {  
            return loggedInUser;  
        }  
    }  
}
```

1.b Déclarer **loggedInUser** pour qu'il soit accessible à ces deux méthodes



Utilisateur « *curieux* »
non connecté



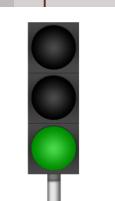
1.a Dans l'étape **Arrange**, spécifier explicitement que l'utilisateur n'est pas connecté
loggedInUser= null

(en s'inspirant de la terminologie métier existante utilisée par la méthode d'où vient le **null**)

2. renvoyer un **loggedInUser**

(c-a-d interroger cette méthode pour savoir

si l'utilisateur est connecté (un vrai objet **User**) ou pas (**null**)



Reste plus qu'à expliciter le null en « s'imprégnant » du langage du métier ...

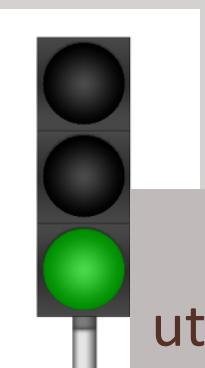


```
public class TripServiceTest {  
    private static final User GUEST = null;  
  
    private User loggedInUser;  
  
    @Test(expected = UserNotLoggedInException.class)  
    public void should_throw_an_exception_when_user_is_not_logged_in(){  
        loggedInUser = GUEST;  
        TripService tripService = new TestableTripService();  
        tripService.getTripsByUser(null);  
    }  
  
    private class TestableTripService extends TripService {  
        @Override  
        protected User getLoggedInUser()  
            return loggedInUser;  
    }  
}
```

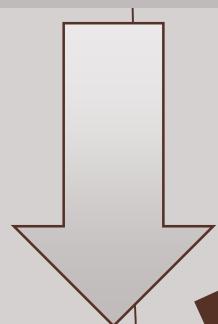
null reste mystérieux : un *Magic Number* ?

Remplaçons le par une constante,
mais comment la nommer ?

Dans une appli web, un utilisateur
qui navigue sans être connecté
est habituellement appelé : **invité**



Qu'à cela ne tienne,
utilisons GUEST dans le code



Bonne Pratique :
Terme métier
repris dans le code



Refactoring : Améliorer la lisibilité des tests

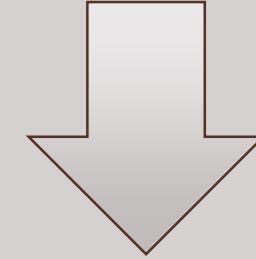
Zoom sur null de getTripsByUser()

```
public class TripServiceTest {  
  
    private static final User GUEST = null;  
    private User loggedInUser;  
  
    @Test(expected = UserNotLoggedInException.class)  
    public void should_throw_an_exception_when_user_is_not_logged_in() {  
        TripService tripService = new TestableTripService();  
        loggedInUser = GUEST;  
  
        tripService.getTripsByUser(null);  
    }  
  
    private class TestableTripService extends TripService {  
  
        @Override  
        protected User getLoggedInUser() {  
            return loggedInUser;  
        }  
    }  
}
```

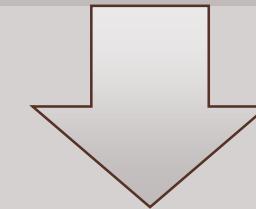


utilisateur potentiellement un *ami*
de l'utilisateur curieux

Dans ce test, l'utilisateur « curieux » n'est pas connecté.



Peu importe qui est l'utilisateur *ami*
car dans ce test, il ne sera jamais consulté
et donc n'impactera en aucun cas le *verdict* du test

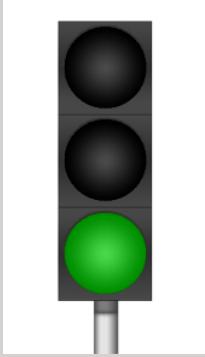


C'est pour cela qu'un **null** suffisait :
pas la peine de perdre son temps à créer un
utilisateur non utilisé

Reste plus qu'à expliciter le null en montrant l'intention métier ...



```
public class TripServiceTest {  
    private static final User UNUSED_USER = null;  
    private static final User GUEST = null;  
    private User loggedInUser;  
  
    @Test(expected = UserNotLoggedInException.class)  
    public void should_throw_an_exception_when_user_is_not_logged_in() {  
        TripService tripService = new TestableTripService();  
        loggedInUser = GUEST;  
  
        tripService.getTripsByUser(UNUSED_USER);  
    }  
  
    private class TestableTripService extends TripService {  
  
        @Override  
        protected User getLoggedInUser() {  
            return loggedInUser;  
        }  
    }  
}
```



Qu'à cela ne tienne,
utilisons **UNUSED_USER** dans le code !



null suffit :
pas la peine de perdre son temps à créer un
utilisateur non utilisé

Petit coup d'œil sur la couverture de code

```
TripServiceTest.java ✘
1 package org.craftedsw.tripservicekata.trip;
2
3 import org.junit.Test;
4 import org.craftedsw.tripservicekata.exception.UserNotLoggedInException;
5 import org.craftedsw.tripservicekata.user.User;
6
7 public class TripServiceTest {
8
9     private static final User GUEST = null;
10    private static final User UNUSED_USER = null;
11    private User loggedInUser;
12
13    @Test(expected = UserNotLoggedInException.class)
14    public void should_throw_an_exception_when_user_is_not_logged_in() {
15        TripService tripService = new TestableTripService();
16        loggedInUser = GUEST;
17        tripService.getTripsByUser(UNUSED_USER);
18    }
19
20    private class TestableTripService extends TripService {
21
22        @Override
23        protected User getLoggedInUser() {
24            return loggedInUser;
25        }
26
27    }
28
29 }
30
```

```
TripService.java ✘
1 package org.craftedsw.tripservicekata.trip;
2
3 import java.util.ArrayList;
4 import java.util.List;
5
6 import org.craftedsw.tripservicekata.exception.UserNotLoggedInException;
7 import org.craftedsw.tripservicekata.user.User;
8 import org.craftedsw.tripservicekata.user.UserSession;
9
10 public class TripService {
11
12     public List<Trip> getTripsByUser(User user) throws UserNotLoggedInException {
13         List<Trip> tripList = new ArrayList<Trip>();
14         User loggedUser = getLoggedInUser();
15         boolean isFriend = false;
16         if (loggedUser != null) {
17             for (User friend : user.getFriends()) {
18                 if (friend.equals(loggedUser)) {
19                     isFriend = true;
20                     break;
21                 }
22             }
23             if (isFriend) {
24                 tripList = TripDAO.findTripsByUser(user);
25             }
26             return tripList;
27         } else {
28             throw new UserNotLoggedInException();
29         }
30     }
31
32     protected User getLoggedInUser() {
33         return UserSession.getInstance().getLoggedInUser();
34     }
35
36 }
37
```

Par où continuer ? ... Sur quelle partie du code focaliser le deuxième test ?



```
public class TripService {
```

```
    public List<Trip> getTripsByUser(User user) throws UserNotLoggedInException {
```

```
        List<Trip> tripList = new ArrayList<Trip>();
```

```
        User loggedUser = UserSession.getInstance().getLoggedUser();  
        boolean isFriend = false;
```

```
        if (loggedUser != null) {
```

```
            for (User friend : user.getFriends()) {
```

```
                if (friend.equals(loggedUser)) {
```

```
                    isFriend = true;
```

```
                    break;
```

```
                }
```

```
                if (isFriend) {
```

```
                    tripList = TripDAO.f
```

```
                }
```

```
                return tripList;
```

```
            } else {
```

```
                throw new UserNotLoggedInException();
```

```
}
```

2^{ème} branche la moins profonde
(2^{ème} chemin d'exécution le plus court)

Quand l'utilisateur curieux et
l'utilisateur ami NE sont PAS amis

⇒ AUCUN voyage

⇒ 1^{er} test : DONE !!!



Test n°2

Comportement
à couvrir :

Aucun voyage
quand
les utilisateurs NE sont PAS amis

```

public class TripServiceTest {

    private static final User GUEST = null;
    private static final User UNUSED_USER = null;
    private static final User LOGGED_IN_USER = new User();
    private static final User ANOTHER_USER = new User();
    private static final Trip TO_ITALY = new Trip();
}

```

Cette fois-ci, l'**utilisateur curieux** est connecté (**LOGGED_IN_USER**);

```

private User loggedInUser;

@Test(expected = UserNotLoggedInException.class)
public void should_throw_an_exception_when_user_is_not_logged_in() {
}

```

```

@Test
public void should_not_return_any_trips_when_users_are_not_friends() {
    TripService tripService = new TestableTripService();
}

```

loggedInUser = **LOGGED_IN_USER**;

```

User friend = new User();
friend.addFriend(ANOTHER_USER);
friend.addTrip(TO_ITALY);

```

Yagni! (Simple) : contexte minimum
1 seul ami et 1 seul voyage suffisent pour écrire ce test

Nécessaire de **créer un vrai utilisateur ami avec une “histoire”** (d'ami(s) et de voyage(s))
pas ami avec l'utilisateur connecté,
mais ami avec un autre utilisateur (**ANOTHER_USER**)

List<Trip> friendTrips=tripService.getTripsByUser(friend);

assertThat(friendTrips).hasSize(0);

Assertion : aucun voyage !

}

private class TestableTripService extends TripService { //... }



Petit coup d'œil sur la couverture de code

```
TripServiceTest.java ✘
9 import static org.assertj.core.api.Assertions.assertThat;
10
11 public class TripServiceTest {
12
13     private static final User GUEST = null;
14     private static final User UNUSED_USER = null;
15     private static final User LOGGED_IN_USER = new User();
16     private User loggedInUser;
17
18     @Test(expected = UserNotLoggedInException.class)
19     public void should_throw_an_exception_when_user_is_not_logged_in() {
20         TripService tripService = new TestableTripService();
21         loggedInUser = GUEST;
22         tripService.getTripsByUser(UNUSED_USER);
23     }
24
25     @Test
26     public void should_not_return_any_trips_when_users_are_not_friends() {
27         TripService tripService = new TestableTripService();
28         loggedInUser = LOGGED_IN_USER;
29         User friend = new User();
30
31         List<Trip> friendTrips = tripService.getTripsByUser(friend);
32
33         assertThat(friendTrips).hasSize(0);
34     }
35
36     private class TestableTripService extends TripService {
37
38         @Override
39         protected User getLoggedInUser() {
40             return loggedInUser;
41         }
42
43     }
44 }
```

```
TripService.java ✘
1 package org.craftedsw.tripservicekata.trip;
2
3 import java.util.ArrayList;
4 import java.util.List;
5
6 import org.craftedsw.tripservicekata.exception.UserNotLoggedInException;
7 import org.craftedsw.tripservicekata.user.User;
8 import org.craftedsw.tripservicekata.user.UserSession;
9
10 public class TripService {
11
12     public List<Trip> getTripsByUser(User user) throws UserNotLoggedInException {
13         List<Trip> tripList = new ArrayList<Trip>();
14         User loggedUser = getLoggedInUser();
15         boolean isFriend = false;
16         if (loggedUser != null) {
17             for (User friend : user.getFriends()) {
18                 if (friend.equals(loggedUser)) {
19                     isFriend = true;
20                     break;
21                 }
22             }
23             if (isFriend) {
24                 tripList = TripDAO.findTripsByUser(user);
25             }
26             return tripList;
27         } else {
28             throw new UserNotLoggedInException();
29         }
30     }
31
32     protected User getLoggedInUser() {
33         return UserSession.getInstance().getLoggedUser();
34     }
35
36 }
37
```

Par où continuer ? ... Sur quelle partie du code focaliser le deuxième test ?



```
public class TripService {
```

```
    public List<Trip> getTripsBvUser(User user) throws UserNotLoggedInException {
```

```
        List<Trip> tripList = new ArrayList<Trip>();
```

```
        User loggedUser = UserSession.getInstance().getLoggedUser();
```

```
        boolean isFriend = false;
```

```
        if (loggedUser != null) {
```

```
            for (User friend : user.getFriends()) {
```

```
                if (friend.equals(loggedUser)) {
```

```
                    isFriend = true;
```

```
                    break;
```

```
                }
```

```
            }  
            if (isFriend) {
```

```
                tripList = TripDAO.findTripsByUser(user);
```

```
            }
```

```
            return tripList;
```

⇒ 2ème test : DONE !!!

```
}  
else {
```

```
    throw new UserNotLoggedInException();
```

⇒ 1er test : DONE !!!

```
}
```

```
}
```

branche la moins profonde
restante à couvrir
Liste de voyages
Quand l'utilisateur curieux et
l'utilisateur ami SONT amis



Test n°3

Comportement
à couvrir :

Aucun voyage
quand
les utilisateurs NE sont PAS amis

En s'inspirant des tests précédents ...

```
@Test
```

```
public void should_return_trip_when_users_are_friends() {  
    TripService tripService = new TestableTripService();
```

```
    loggedInUser = LOGGED_IN_USER; l'utilisateur curieux est connecté
```

```
    User friend = new User();  
    friend.addFriend(ANOTHER_USER);  
    friend.addFriend(loggedInUser);  
    friend.addTrip(TO_ITALY);  
    friend.addTrip(TO_SPAIN);
```

```
    List<Trip> friendTrips = tripService.getTripsByUser(friend);
```

```
    assertThat(friendTrips).hasSize(2); Une liste de 2 voyages est récupérée
```

```
    assertThat(friendTrips).contains(TO_ITALY, TO_SPAIN); Qui contient bien les voyages de l'ami
```

Package Explorer JUnit Finished after 0,073 seconds

Runs: 3/3 Errors: 1 Failures: 0

org.craftedsw.tripservicekata.trip.TripServiceTest [Runner: JUnit 4] (0,001 s)

- should_return_trip_when_users_are_friends (0,000 s) *CollaboratorCallException: TripDAO should not be invoked on an unit test.*
- should_throw_an_exception_when_user_is_not_loggedd_in (0,000 s)
- should_not_return_any_trips_when_users_are_not_friends (0,000 s)

Failure Trace

```
java.lang.Exception: CollaboratorCallException: TripDAO should not be invoked on an unit test.  
at org.craftedsw.tripservicekata.trip.TripDAO.findTripsByUser(TripDAO.java:11)  
at org.craftedsw.tripservicekata.trip.TripService.getTripsByUser(TripService.java:24)  
at org.craftedsw.tripservicekata.trip.TripServiceTest.should_return_trip_when_users_are_friends(TripServiceTest.java:59)
```



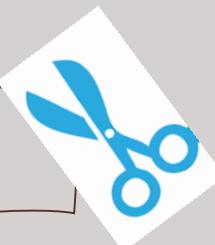
Mettre en place
la couture (Seam)

pour isoler la dépendance
à la couche d'accès aux données
(TripDAO)

Etapes pour mettre en place le Seam et isoler la dépendance à la couche d'accès aux données



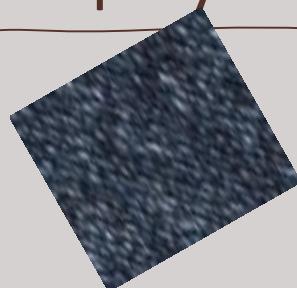
1. Dans le code de production (`TripService`)



- ✓ Isoler le code qui appelle la dépendance (à l'aide de l'Extract Method de l'IDE)
⇒ dans une méthode `tripsBy` de type `protected`

2. Dans le code test, procéder à la couture avec l'aide d'une polymorphisme

- ✓ La « *thin layer* » `TestableTripService` existe déjà ...
- ✓ Redéfinir la méthode `tripsBy`
et bouchonner son comportement (utiliser le `trips` de `User`)



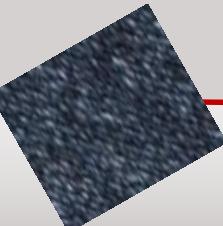
- ✓ puisque c'est bien la « *thin layer* »
qui est déjà instanciée dans l'étape *Arrange* du test.



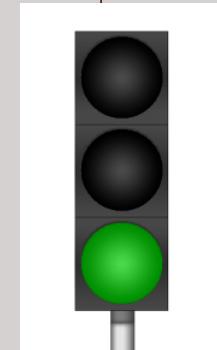
Isoler la dépendance au Singleton

Et faire passer le test !

```
public class TripServiceTest {  
    ...  
  
    public void should_return_trip_when_users_are_friends() {  
        // idem précédemment,  
        // TestableTripService étant déjà instancié  
    }  
  
    private class TestableTripService extends TripService {  
        @Override  
        protected User getLoggedInUser() {  
            return null;  
        }  
  
        @Override  
        protected List<Trip> tripsBy(User user) {  
            return user.trips();  
        }  
    }  
}
```



```
public class TripService {  
  
    public List<Trip> getTripsByUser(User user)  
        throws UserNotLoggedInException {  
        //...  
        tripList = tripsBy(user);  
        //...  
    }  
  
    protected List<Trip> tripsBy(User user) {  
        return TripDAO.findTripsByUser(user);  
    }  
}
```



Petit coup d'œil sur la couverture de code

File Edit Source Refactor Navigate Search Project Run Window Help

Quick Access

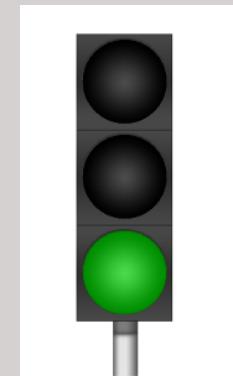
TripServiceTest.java

```
44
45 @Test
46 public void should_return_trip_when_users_are_friends() {
47     TripService tripService = new TestableTripService();
48
49     loggedInUser = LOGGED_IN_USER;
50
51     User friend = new User();
52     friend.addFriend(ANOTHER_USER);
53     friend.addFriend(loggedInUser);
54     friend.addTrip(TO_ITALY);
55     friend.addTrip(TO_SPAIN);
56
57     List<Trip> friendTrips = tripService.getTripsByUser(friend);
58
59     assertThat(friendTrips).hasSize(2);
60     assertThat(friendTrips).contains(TO_ITALY, TO_SPAIN);
61 }
62
63 private class TestableTripService extends TripService {
64     @Override
65     protected User getLoggedInUser() {
66         return loggedInUser;
67     }
68
69     @Override
70     protected List<Trip> tripsBy(User user) {
71         return user.trips();
72     }
73 }
74 }
```

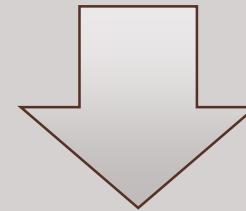
TripService.java

```
3+ import java.util.ArrayList;
9
10 public class TripService {
11
12     public List<Trip> getTripsByUser(User user) throws UserNotLoggedInException {
13         List<Trip> tripList = new ArrayList<Trip>();
14         User loggedUser = getLoggedInUser();
15         boolean isFriend = false;
16
17         for (User friend : user.getFriends()) {
18             if (friend.equals(loggedUser)) {
19                 isFriend = true;
20                 break;
21             }
22         }
23         if (isFriend) {
24             tripList = tripsBy(user);
25         }
26         return tripList;
27     } else {
28         throw new UserNotLoggedInException();
29     }
30 }
31
32     protected User getLoggedInUser() {
33         return UserSession.getInstance().getLoggedUser();
34     }
35
36     protected List<Trip> tripsBy(User user) {
37         return TripDAO.findTripsByUser(user);
38     }
39 }
```





Legacy couvert à 100%



Refactoring du code de test

fortement conseillé

avant de passer

au refactoring du code de production



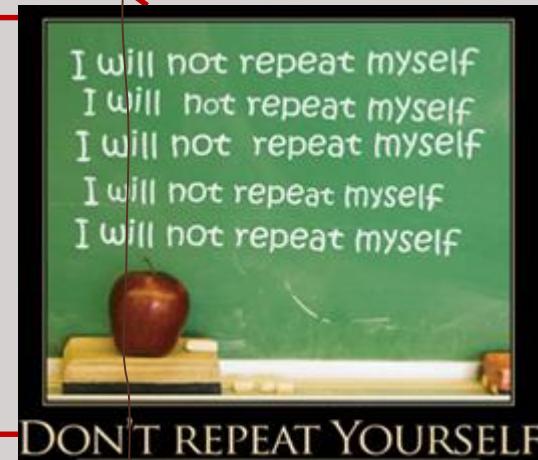
Ne sentez-vous pas comme une mauvaise odeur dans les étapes
Arrange des tests ?

```
public class TripServiceTest {  
    //...  
  
    @Test(expected = UserNotLoggedInException.class)  
    public void should_throw_an_exception_when_user_is_not_logged_in() {  
        TripService tripService = new TestableTripService();  
        loggedInUser = GUEST;  
        //...  
    }  
}
```

```
@Test  
public void should_not_return_any_trips_when_users_are_not_friends() {  
    TripService tripService = new TestableTripService();  
    loggedInUser = LOGGED_IN_USER;  
    User friend = new User();  
    friend.addFriend(ANOTHER_USER);  
    friend.addTrip(TO_ITALY);  
    //...  
}
```

```
@Test  
public void should_return_trip_when_users_are_friends() {  
    TripService tripService = new TestableTripService();  
    loggedInUser = LOGGED_IN_USER;  
    User friend = new User();  
    friend.addFriend(ANOTHER_USER);  
    friend.addFriend(loggedInUser);  
    friend.addTrip(TO_ITALY);  
    friend.addTrip(TO_SPAIN);  
    //...  
}
```

```
private class TestableTripService extends TripService {}//...  
}
```



Bonne pratique :

A partir de 3 fois, on a de la duplication,
il est temps d'agir ...

```

public class TripServiceTest {
    //...

    @Before
    public void setUp() {
        tripService = new TestableTripService();
        loggedInUser = LOGGED_IN_USER; ←
    }

    @Test(expected = UserNotLoggedInException.class)
    public void should_throw_an_exception_when_user_is_not_logged_in() {
        loggedInUser = GUEST; ←
        //...
    }

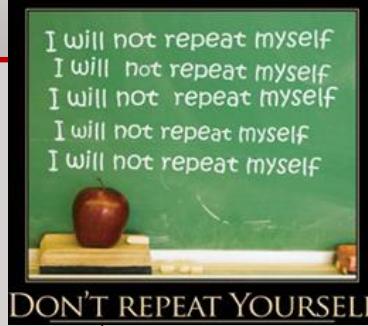
    @Test
    public void should_not_return_any_trips_when_users_are_not_friends() {
        User friend = new User();
        friend.addFriend(ANOTHER_USER);
        friend.addTrip(TO_ITALY);
        //...
    }

    @Test
    public void should_return_trip_when_users_are_friends() {
        User friend = new User();
        friend.addFriend(ANOTHER_USER);
        friend.addFriend(loggedInUser);
        friend.addTrip(TO_ITALY);
        friend.addTrip(TO_SPAIN);
        //...
    }
}

    p    Améliorer la lisibilité :
        plus simple et plus rapide à comprendre

```

*Factorisation dans un `setUp`
et choix sur l'état de connection
le plus frequent...*



**Encore un dernier
petit effort ?**



```

User friend = UserBuilder.aUser()
    .friendsWith(ANOTHER_USER, loggedInUser)
    .withTrips(TO_ITALY, TO_SPAIN)
    .build();

```

service {
//...}

Implémentation du UserBuilder

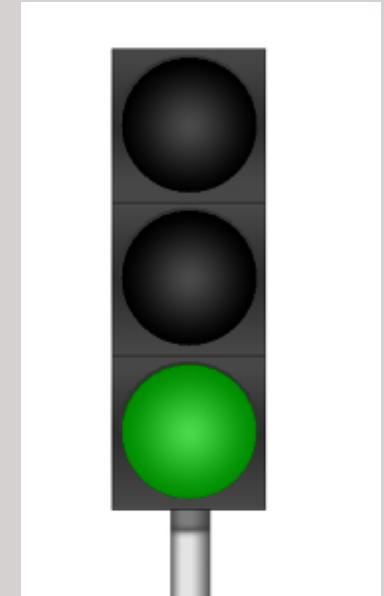
```
public class UserBuilder {  
    private User[] friends;  
    private Trip[] trips;  
  
    public static UserBuilder aUser() {  
        return new UserBuilder();  
    }  
  
    public UserBuilder friendsWith(User...friends) {  
        this.friends = friends;  
        return this;  
    }  
  
    public UserBuilder withTrips(Trip...trips) {  
        this.trips = trips;  
        return this;  
    }  
  
    public User build() {  
        User user = new User();  
        addFriendsTo(user);  
        addTripsTo(user);  
        return user;  
    }  
}  
// suite ci-contre ...
```

Bonne Pratique :
Améliorer la lisibilité de vos tests
(⇒ construction de vos objets
plus simple et plus sûre)

```
private void addFriendsTo(User user) {  
    for (User friend : friends) {  
        user.addFriend(friend);  
    }  
}  
  
private void addTripsTo(User user) {  
    for (Trip trip : trips) {  
        user.addTrip(trip);  
    }  
}
```

Etapes Arrange du code de test refactorées ...

```
public class TripServiceTest {  
    //...  
  
    @Before  
    public void setUp() {  
        tripService = new TestableTripService();  
        loggedInUser = LOGGED_IN_USER;  
    }  
  
    @Test(expected = UserNotLoggedInException.class)  
    public void should_throw_an_exception_when_user_is_not_logged_in() {  
        loggedInUser = GUEST;  
        //...  
    }  
  
    @Test  
    public void should_not_return_any_trips_when_users_are_not_friends() {  
        User friend = aUser()  
            .friendsWith(ANOTHER_USER)  
            .withTrips(TO_ITALY)  
            .build();  
        //...  
    }  
  
    @Test  
    public void should_return_trip_when_users_are_friends() {  
        User friend = aUser()  
            .friendsWith(ANOTHER_USER, loggedInUser)  
            .withTrips(TO_ITALY, TO_SPAIN)  
            .build();  
        //...  
    }  
    private class TestableTripService extends TripService {}//...}
```





Seconde Partie

Refactorer !!!

Que cherche-t-on en refactorant ?

... Façonner un code plus propre en essayant de ...

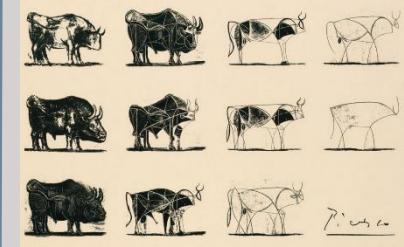
Supprimer un(des) code smell(s)

Simplifier et réduire la complexité cylomatique

Rendre le code plus SOLID et rompre les dépendances

Améliorer la lisibilité du test

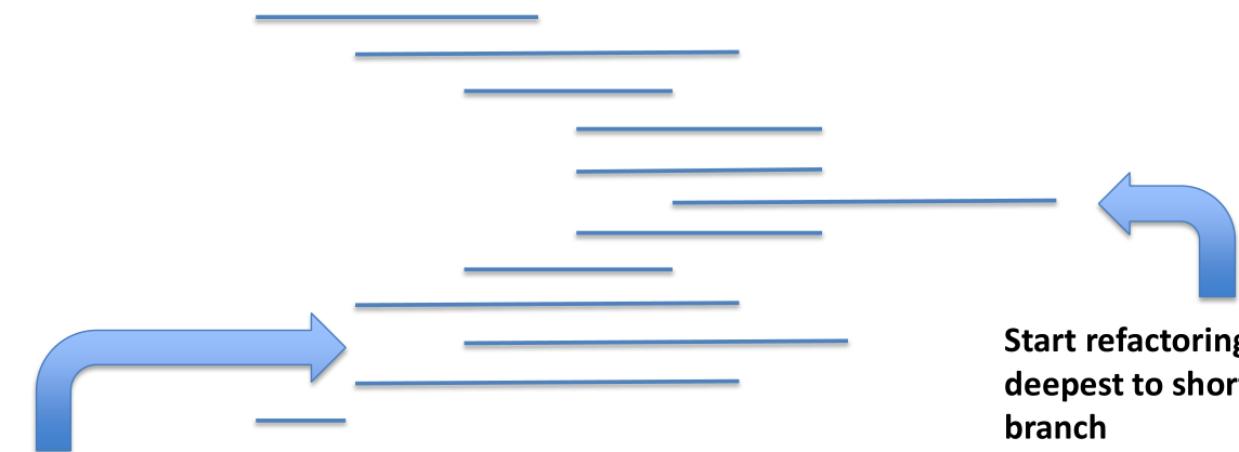
en portant une attention particulière au **nommage**
(pour rendre explicite l'intention métier)



Par où commencer ? ...

Petit conseil de Sandro pour se lancer dans la phase de refactoring

Working with Legacy Code Tips



Start testing from
shortest to
deepest branch

Start refactoring from
deepest to shortest
branch

*Commencer le refactoring
par la branche la plus profonde*

Par où commencer ? ... Sur quelle partie du code démarrer le refactoring?



```
public class TripService {  
  
    public List<Trip> getTripsByUser(User user) throws UserNotLoggedInException {  
        List<Trip> tripList = new ArrayList<Trip>();  
        User loggedUser = getLoggedInUser();  
        boolean isFriend = false;  
        if (loggedUser != null) {  
            for (User friend : user.getFriends()) {  
                if (friend.equals(loggedUser)) {  
                    isFriend = true;  
                    break;  
                }  
            }  
            if (isFriend) {  
                tripList = tripsBy(user);  
            }  
        }  
        return tripList;  
    } else {  
        throw new UserNotLoggedInException();  
    }  
}  
//...
```

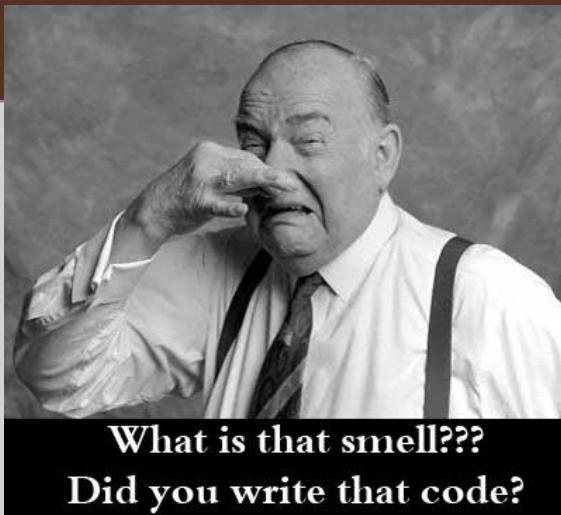
*2^{ème} branche la plus profonde
⇒ Le refactoring pourrait commencer avec ce bout de code ...*

1^{ère} branche la plus profonde mais rien à refactorer...



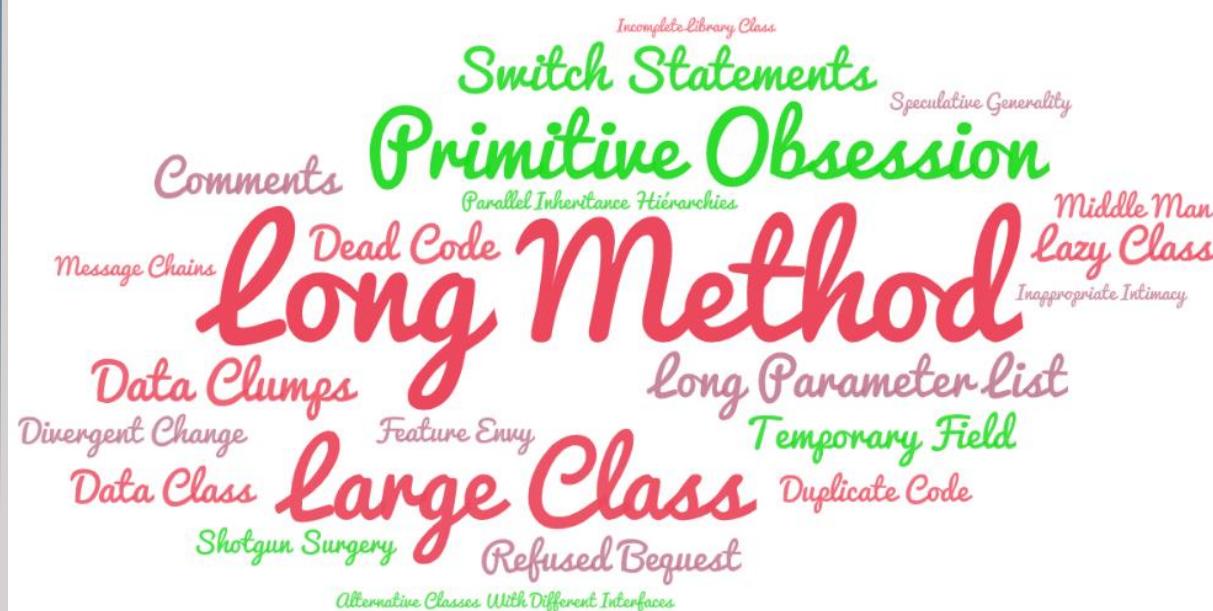
Revue de code à la recherche des mauvaises odeurs (Code smell)

Qu'est-ce qui sent mauvais dans ce code ?



```
public class TripService {  
  
    public List<Trip> getTripsByUser(User user)  
        throws UserNotLoggedInException {  
  
        //...  
        for (User friend : user.getFriends()) {  
            if (friend.equals(loggedUser)) {  
                isFriend = true;  
                break;  
            }  
        }  
        //...  
    }  
}
```

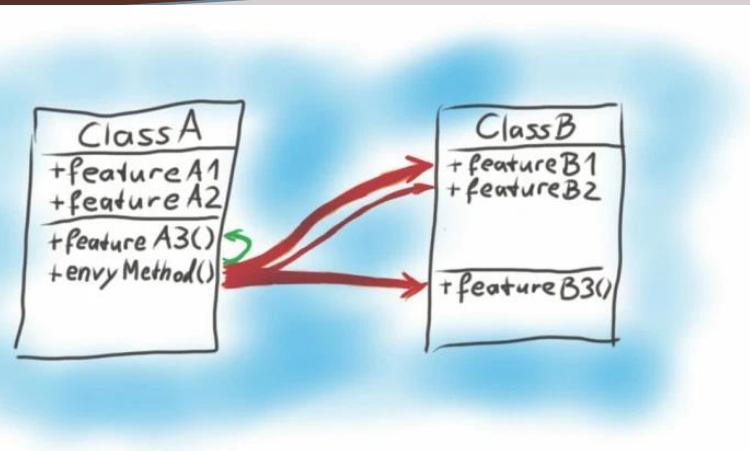
Quid des responsabilités ?



Avez-vous reconnu le code smell correspondant à cette mauvaise odeur ?

En savoir plus sur les code smells :
<https://sourcemaking.com/refactoring/smells>

Détecter un code Smell Feature Envy

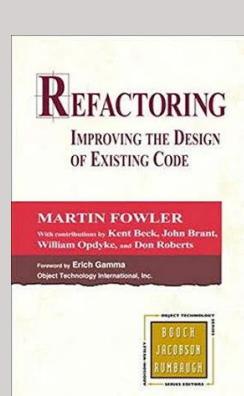


Methods suffer from Feature Envy,
if they use other classes more than their own.

Martin Fowler, the inventor of Code Smells and Feature Envy, puts it like this:

The whole point of objects is that they are a technique to package data with the processes used on that data. A classic smell is a method that seems more interested in a class other than the one it is in. The most common focus of the envy is the data.

We've lost count of the times we've seen a method invokes half-dozen getting methods on another object to calculate some value



Le Feature Envy de notre legacy

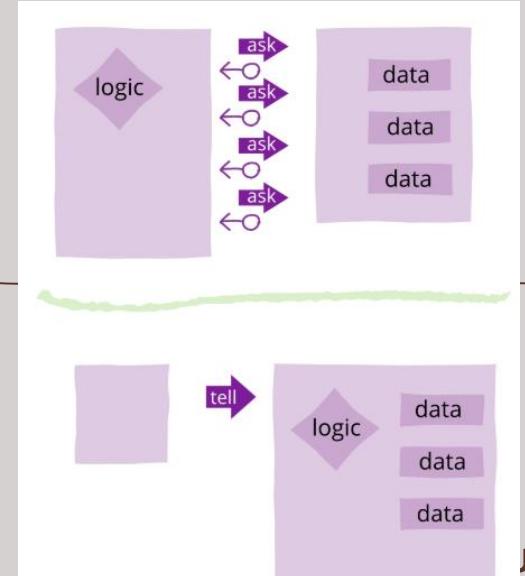


```
public class TripService {  
  
    public List<Trip> getTripsByUser(User user)  
        throws UserNotLoggedInException  
{  
    //...  
    for (User friend : user.getFriends()) {  
        if (friend.equals(loggedUser)) {  
            isFriend = true;  
            break;  
        }  
    //...  
    }  
}
```

```
public class User {  
//...  
    public List<User> getFriends() {  
        return friends;  
    }  
//...  
}
```

La solution :

Tell don't ask !



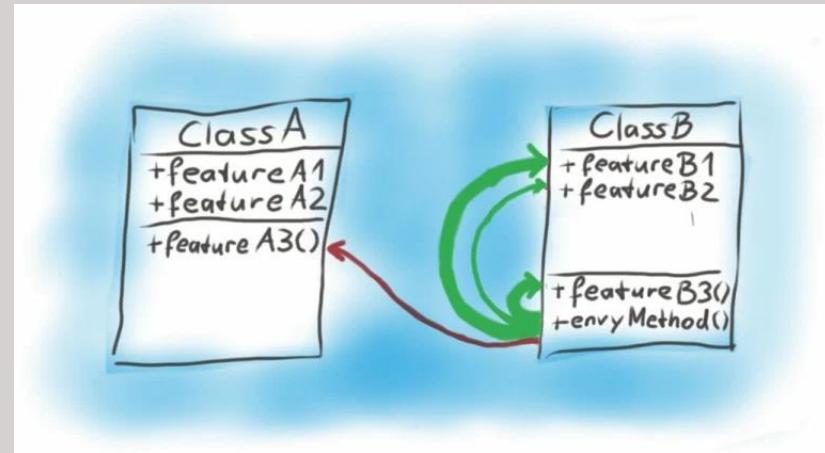
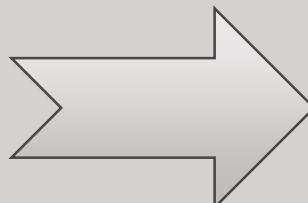
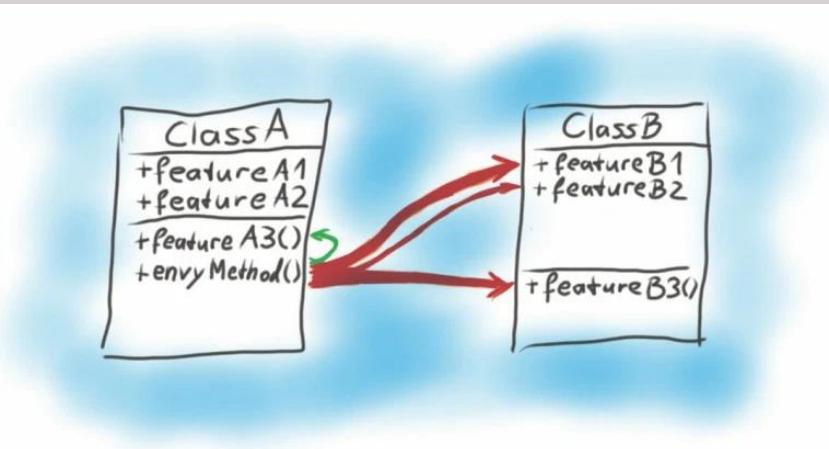
Refactorer un code smell Feature Envy

Tell don't ask !



Give the method what it desires – the other class!

*The method clearly wants to be elsewhere, so you use **Move Method** to get it there.*



After moving the envy method to the desired class, the green arrows total thickness exceed the others.

We have *no more Feature Envy*.

We reduced the coupling (red arrows) between our classes and raised the cohesion (green arrows) inside our classes.

Isabelle BLASQUEZ

Refactorer notre Feature Envy : comment faire ?

Tell don't ask !



*Ne vous contentez pas d'extraire le code **TripService** et de déplacer ce code dans **User** ...*

... mais pour avoir une implémentation aussi simple que possible et bien adapter au besoin...

*Implémenter plutôt en TDD une nouvelle méthode dans **User**
qui rendra le service attendu par **TripService***

Etapes pour refactorer notre Feature Envy

Tell don't ask !

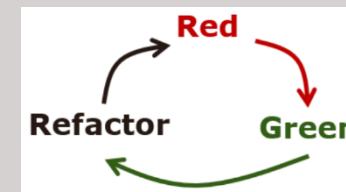
1. Bien nommer la nouvelle méthode qui doit refléter l'intention métier

D'un point de vue métier, il semblerait qu'on cherche à savoir
si un utilisateur est ami avec un autre utilisateur

⇒ Une nouvelle méthode **isFriendWith(anotherUser)** dans la classe **User**

2. Développer en TDD

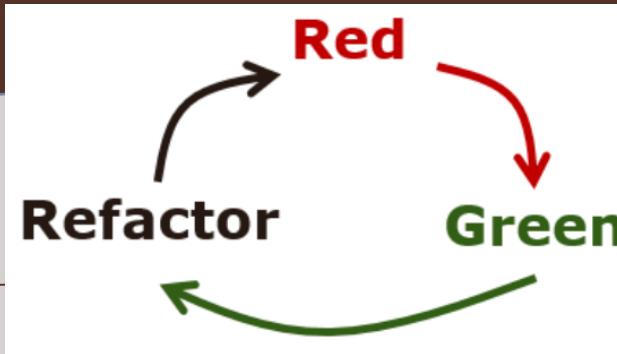
la nouvelle méthode **isFriendWith** de la classe **User**



3. Appeler la méthode **isFriendWith** depuis **TripService**

Développer en TDD de isFriendWith de la classe User

```
public class UserTest {  
  
    private static final User ALICE = new User();  
    private static final User BOB = new User();  
  
    @Test  
    public void should_inform_when_users_are_not_friends() {  
        User user = aUser().friendsWith(ALICE).build();  
  
        assertThat(user.isFriendWith(BOB)).isFalse();  
    }  
  
    @Test  
    public void should_inform_when_users_are_friends() {  
        User user = aUser().friendsWith(ALICE, BOB).build();  
  
        assertThat(user.isFriendWith(BOB)).isTrue();  
    }  
}
```



```
public class User {  
    //...  
  
    public boolean isFriendWith(User anotherUser) {  
        return friends.contains(anotherUser);  
    }  
}
```



Pour éviter le **NullPointerException**,
vérifier que les tableaux aient
bien été initialisés dans le **UserBuilder**

```
public class UserBuilder {  
    private User[] friends = new User[]{};  
    private Trip[] trips = new Trip[]{};  
  
    //...  
}
```

Aviez-vous bien prévu dans le monteur
qu'un utilisateur pourrait n'avoir
aucun ami et/ou aucun voyage ?

Appeler la méthode isFriendWith depuis TripService

Tell don't ask !

```
public class TripService {  
  
    public List<Trip> getTripsByUser(User user) throws UserNotLoggedInException {  
        List<Trip> tripList = new ArrayList<Trip>();  
        User loggedUser = getLoggedInUser();  
        boolean isFriend = false;  
        if (loggedUser != null) {  
            isFriend = user.isFriendWith(loggedUser);  
            if (isFriend) {  
                tripList = tripsBy(user);  
            }  
            return tripList;  
        } else {  
            throw new UserNotLoggedInException();  
        }  
    }  
    //...  
}
```

```
public class User {  
  
    //...  
  
    public boolean isFriendWith(User anotherUser) {  
        return friends.contains(anotherUser);  
    }  
}
```



Refactoring :

Un peu de nettoyage autour de isFriend

```
public class TripService {  
  
    public List<Trip> getTripsByUser(User user)  
        throws UserNotLoggedInException {  
        //...  
        boolean isFriend = false;  
        if (loggedUser != null) {  
            isFriend = user.isFriendWith(loggedUser);  
            if (isFriend) {  
                tripList = tripsBy(user);  
            }  
        }  
        return tripList;  
    }  
    //...  
}
```

Bonne Pratique :
Rapprocher la déclaration de la variable
isFriend
au plus près du code où elle est utilisée
et simplifier en une ligne

```
if (loggedUser != null) {  
    boolean isFriend = user.isFriendWith(loggedUser);  
    if (isFriend) {  
        tripList = tripsBy(user);  
    }  
}
```

```
if (loggedUser != null) {  
    if (user.isFriendWith(loggedUser)) {  
        tripList = tripsBy(user);  
    }  
}
```

Refactoring Inline de l'IDE
(suppression de la variable isFriend)



Après le petit nettoyage autour de isFriend ...

```
public class TripService {  
  
    public List<Trip> getTripsByUser(User user) throws UserNotLoggedInException {  
        List<Trip> tripList = new ArrayList<Trip>();  
        User loggedUser = getLoggedInUser();  
        if (loggedUser != null) {  
            if (user.isFriendWith(loggedUser)) {  
                tripList = tripsBy(user);  
            }  
            return tripList;  
        } else {  
            throw new UserNotLoggedInException();  
        }  
    }  
    //... méthodes protected tripsBy et getLoggedInUser...  
}
```



Bonne pratique :

Améliorer la lisibilité et
réduire la complexité
grâce aux clauses de garde
(guard clauses)

Clause de garde: Définition



Guard Clause

A **GuardClause** (one of the [SmalltalkBestPracticePatterns](#), and equally applicable in a whole bunch of languages) is a chunk of code at the top of a function (or block) that serves a similar purpose to a Precondition.

It typically does one (or any or all) of the following:

- checks the passed-in parameters, and returns with an error if they're not suitable.
- checks the state of the object, and bails out if the function call is inappropriate.
- checks for trivial cases, and gets rid of them quickly.

For example ...

```
draw() {  
    if (! isVisible()) return;  
    ...  
}
```

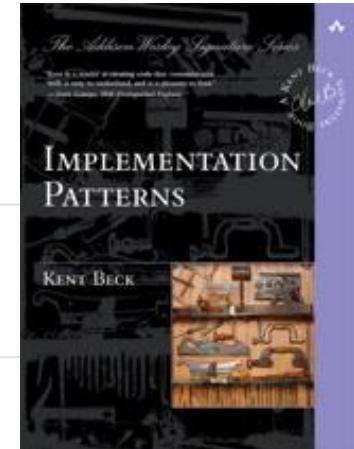
Guard Clause

While programs have a main flow, some situations require deviations from the main flow. The guard clause is a way to express simple and local exceptional situations with purely local consequences. Compare the following:

```
void initialize() {  
    if (!isInitialized()) {  
        ...  
    }  
}
```

with:

```
void initialize() {  
    if (isInitialized())  
        return;  
    ...  
}
```



When I read the first version, I make a note to look for an else clause while I am reading the then clause. I mentally put the condition on a stack. All of this is a distraction while I am reading the body of the then clause. The first two lines of the second version simply give me a fact to note: the receiver hasn't been initialized.

If-then-else expresses alternative, equally important control flows. A guard clause is appropriate for expressing a different situation, one in which one of the control flows is more important than the other. In the initialization example above, the important control flow is what happens when the object is initialized. Other than that, there is just a simple fact to notice, that even if an object is asked to initialize multiple times it will only execute the initialization code once.

Clause de garde: Exemples & Avantages

Améliorer la lisibilité

Réduire la complexité

```
1 function calculateInsurance(userID: number){  
2     const user = myDB.findOne(userID);  
3     if(user){  
4         if(user.insurance === 'Allianz' or user.insurance === 'AXA'){  
5             if(user.nationality === 'Spain'){  
6                 const value = /***  
7                     Complex Algorithm  
8                     */  
9                 return value;  
10            }else{  
11                throw new UserIsNotSpanishException(user);  
12            }  
13        }else{  
14            throw new UserInsuranceNotFoundException(user);  
15        }  
16    }else{  
17        throw new UserNotFoundException('User NotFound!');  
18    }  
19 }
```

```
1 function calculateInsurance(userID: number){  
2     const user = myDB.findOne(userID);  
3     if(!user){  
4         throw new UserNotFoundException('User NotFound!');  
5     }  
6     if(!isValidInsurante(user)){  
7         throw new UserInsuranceNotFoundException(user);  
8     }  
9     if(!isSpanish(user)){  
10        throw new UserIsNotSpanishException(user);  
11    }  
12  
13    const value = /***  
14        Complex Algorithm  
15        */  
16    return value;  
17 }
```



Mise en place de la clause de garde (1/2)

```
public class TripService {  
  
    public List<Trip> getTripsByUser(User user)  
        throws UserNotLoggedInException {  
        List<Trip> tripList = new ArrayList<Trip>();  
        User loggedUser = getLoggedInUser();  
        if (loggedUser != null) {  
            if (user.isFriendWith(loggedUser)) {  
                tripList = tripsBy(user);  
            }  
            return tripList;  
        } else {  
            throw new UserNotLoggedInException();  
        }  
    }  
    //... méthodes protected tripsBy et getLoggedInUser...  
}
```

```
public class TripService {  
  
    public List<Trip> getTripsByUser(User user)  
        throws UserNotLoggedInException {  
        List<Trip> tripList = new ArrayList<Trip>();  
        User loggedUser = getLoggedInUser();  
        if (loggedUser == null) {  
            throw new UserNotLoggedInException();  
        } else {  
            if (user.isFriendWith(loggedUser)) {  
                tripList = tripsBy(user);  
            }  
            return tripList;  
        }  
    }  
    //... méthodes protected tripsBy et getLoggedInUser...  
}
```

Inversion du if
À l'aide de l'IDE ...



Mise en place de la clause de garde (2/2)

```
public class TripService {  
  
    public List<Trip> getTripsByUser(User user)  
        throws UserNotLoggedInException {  
        List<Trip> tripList = new ArrayList<Trip>();  
        User loggedUser = getLoggedInUser();  
        if (loggedUser != null) {  
            throw new UserNotLoggedInException();  
        } else {  
            if (user.isFriendWith(loggedUser)) {  
                tripList = tripsBy(user);  
            }  
            return tripList;  
        }  
    }  
    //... méthodes protected tripsBy et getLoggedInUser...  
}
```

```
public class TripService {  
  
    public List<Trip> getTripsByUser(User user)  
        throws UserNotLoggedInException {  
        List<Trip> tripList = new ArrayList<Trip>();  
        User loggedUser = getLoggedInUser();  
        if (loggedUser != null) {  
            throw new UserNotLoggedInException();  
        }  
        if (user.isFriendWith(loggedUser)) {  
            tripList = tripsBy(user);  
        }  
        return tripList;  
    }  
    //... méthodes protected tripsBy et getLoggedInUser...  
}
```

Réduction d'un niveau de complexité



Rendre explicite l'intention de la clause de garde

```
public class TripService {  
  
    public List<Trip> getTripsByUser(User user)  
        throws UserNotLoggedInException {  
  
        List<Trip> tripList = new ArrayList<Trip>();  
  
        User loggedUser = getLoggedInUser();  
        if (loggedUser != null) {  
            throw new UserNotLoggedInException();  
        }  
  
        if (user.isFriendWith(loggedUser)) {  
            tripList = tripsBy(user);  
        }  
        return tripList;  
    }  
    //... méthodes protected tripsBy et getLoggedInUser...  
}
```

```
public class TripService {  
  
    public List<Trip> getTripsByUser(User user)  
        throws UserNotLoggedInException {  
        List<Trip> tripList = new ArrayList<Trip>();  
  
        User loggedUser = getLoggedInUser();  
        validate(loggedUser);  
  
        if (user.isFriendWith(loggedUser)) {  
            tripList = tripsBy(user);  
        }  
        return tripList;  
    }  
  
    private void validate(User loggedUser) {  
        if (loggedUser == null) {  
            throw new UserNotLoggedInException();  
        }  
    }  
    //... méthodes protected tripsBy et getLoggedInUser...  
}
```

Extract Method sur la clause de garde

dans une méthode qui a la **responsabilité de valider l'état de l'objet**



Refactoring :

Un peu de nettoyage

autour des *trips*



Rapprocher la déclaration au plus près de son utilisation ...

```
public class TripService {  
  
    public List<Trip> getTripsByUser(User user)  
        throws UserNotLoggedInException {  
  
        List<Trip> tripList = new ArrayList<Trip>();  
  
        User loggedUser = getLoggedInUser();  
        validate(loggedUser);  
  
        if (user.isFriendWith(loggedUser)) {  
            tripList = tripsBy(user);  
        }  
        return tripList;  
    }  
  
    //... méthode private validate  
    //... méthodes protected tripsBy et getLoggedInUser  
}
```

```
public class TripService {  
  
    public List<Trip> getTripsByUser(User user)  
        throws UserNotLoggedInException {  
  
        User loggedUser = getLoggedInUser();  
        validate(loggedUser);  
  
        List<Trip> tripList = new ArrayList<Trip>();  
        if (user.isFriendWith(loggedUser)) {  
            tripList = tripsBy(user);  
        }  
        return tripList;  
    }  
  
    //... méthode private validate  
    //... méthodes protected tripsBy et getLoggedInUser  
}
```

Rapprocher la déclaration au plus près de son utilisation

Isabelle BLASQUEZ



Supprimer la variable tripList ... (permettre des return multiple)

```
public class TripService {  
  
    public List<Trip> getTripsByUser(User user)  
        throws UserNotLoggedInException {  
  
        User loggedUser = getLoggedInUser();  
        validate(loggedUser);  
  
        List<Trip> tripList = new ArrayList<Trip>();  
        if (user.isFriendWith(loggedUser)) {  
            tripList = tripsBy(user);  
        }  
        return tripList;  
    }  
    //... méthode private validate  
    //... méthodes protected tripsBy et getLoggedInUser  
}
```

```
public class TripService {  
  
    public List<Trip> getTripsByUser(User user)  
        throws UserNotLoggedInException {  
  
        User loggedUser = getLoggedInUser();  
        validate(loggedUser);  
  
        if (user.isFriendWith(loggedUser)) {  
            return tripsBy(user);  
        }  
        return new ArrayList<>();  
    }  
    //... méthode private validate  
    //... méthodes protected tripsBy et getLoggedInUser  
}
```

Remplacer le premier **tripList** par **return**

Inliner via l'IDE le deuxième **tripList**



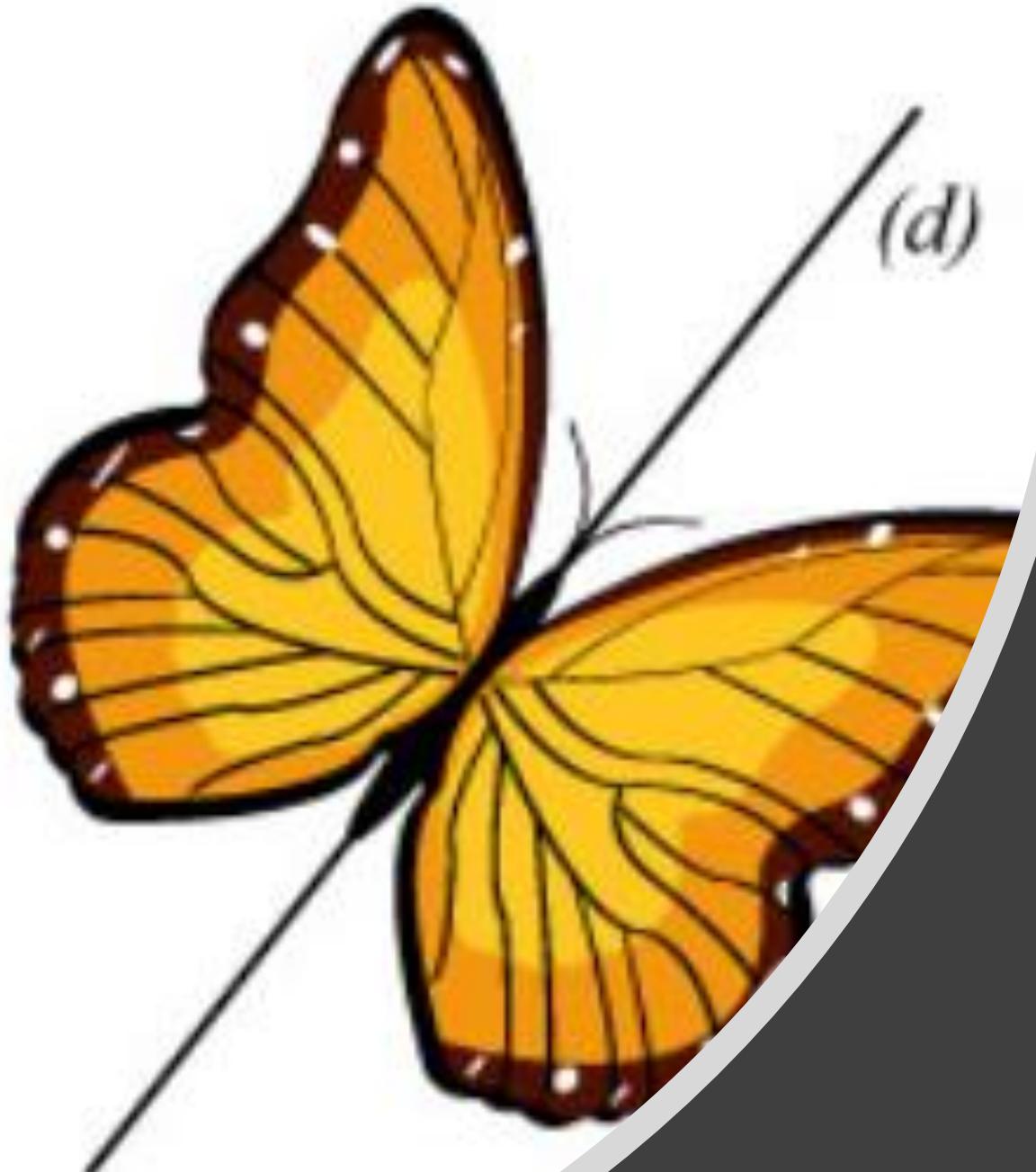
Améliorer la lisibilité en explicitant l'intention métier

```
public class TripService {  
  
    public List<Trip> getTripsByUser(User user)  
        throws UserNotLoggedInException {  
  
        User loggedUser = getLoggedInUser();  
        validate(loggedUser);  
  
        if (user.isFriendWith(loggedUser)) {  
            return tripsBy(user);  
        }  
        return new ArrayList<>();  
    }  
    //... méthode private validate  
    //... méthodes protected tripsBy et getLoggedInUser  
}
```



```
public class TripService {  
  
    public List<Trip> getTripsByUser(User user)  
        throws UserNotLoggedInException {  
  
        User loggedUser = getLoggedInUser();  
        validate(loggedUser);  
  
        if (user.isFriendWith(loggedUser)) {  
            return tripsBy(user);  
        }  
        return noTrips();  
    }  
  
    private ArrayList<Trip> noTrips() {  
        return new ArrayList<>();  
    }  
    //... méthode private validate  
    //... méthodes protected tripsBy et getLoggedInUser  
}
```

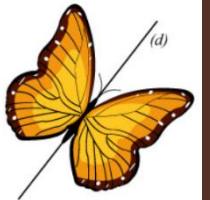
**Rendre explicite l'intention métier : aucun voyage
via un Extract Method bien nommé**



Bonne pratique :

Ajouter du code
pour faire ressortir
la symétrie

permet parfois
de simplifier
plus facilement le code

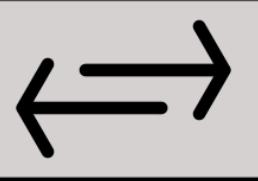


Rendre le if symétrique avec un else ...

```
public class TripService {  
  
    public List<Trip> getTripsByUser(User user)  
        throws UserNotLoggedInException {  
  
        User loggedUser = getLoggedInUser();  
        validate(loggedUser);  
  
        if (user.isFriendWith(loggedUser)) {  
            return tripsBy(user);  
        }  
        return noTrips();  
    }  
  
    //... méthodes private validate et noTrips  
    //... méthodes protected tripsBy et getLoggedInUser  
}
```



```
public class TripService {  
  
    public List<Trip> getTripsByUser(User user)  
        throws UserNotLoggedInException {  
  
        User loggedUser = getLoggedInUser();  
        validate(loggedUser);  
  
        if (user.isFriendWith(loggedUser)) {  
            return tripsBy(user);  
        } else {  
            return noTrips();  
        }  
  
    }  
  
    //... méthodes private validate et noTrips  
    //... méthodes protected tripsBy et getLoggedInUser  
}
```



if classique ou opérateur ternaire selon sa préférence ...

```
public class TripService {  
  
    public List<Trip> getTripsByUser(User user)  
        throws UserNotLoggedInException {  
  
        User loggedUser = getLoggedInUser();  
        validate(loggedUser);  
  
        if (user.isFriendWith(loggedUser)) {  
            return tripsBy(user);  
        } else {  
            return noTrips();  
        }  
    }  
  
    //... méthodes private validate et noTrips  
    //... méthodes protected tripsBy et getLoggedInUser  
}
```

```
public class TripService {  
  
    public List<Trip> getTripsByUser(User user)  
        throws UserNotLoggedInException {  
  
        User loggedUser = getLoggedInUser();  
        validate(loggedUser);  
  
        return user.isFriendWith(loggedUser)  
            ? tripsBy(user)  
            : noTrips();  
    }  
  
    //... méthodes private validate et noTrips  
    //... méthodes protected tripsBy et getLoggedInUser  
}
```

*If symétrique
Facilite écriture ternaire
(conversion auto. via IDE)*

REFACTOR

ALL THE CODE

Les tests et le code parlent le même langage ...

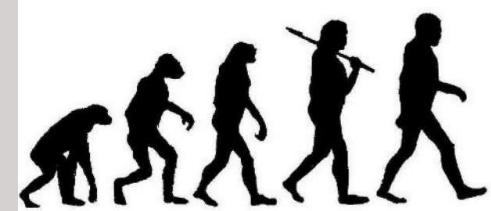
```
public class TripServiceTest {  
    //...  
    @Before  
    public void setUp() { //...}  
  
    @Test(expected = UserNotLoggedInException.class)  
    public void  
    should_throw_an_exception_when_user_is_not_logged_in()  
    { //...}  
  
    public void  
    should_not_return_any_trips_when_users_are_not_friends()  
    { //...}  
  
    @Test  
    public void should_return_trip_when_users_are_friends()  
    { //...}  
  
    private class TestableTripService extends TripService  
    { //...}  
}
```

```
public class TripService {  
  
    public List<Trip> getTripsByUser(User user)  
        throws UserNotLoggedInException{  
  
        User loggedUser = getLoggedInUser();  
        validate(loggedUser);  
  
        return user.isFriendWith(loggedUser)  
            ? tripsBy(user)  
            : noTrips();  
    }  
  
    private ArrayList<Trip> noTrips() {  
        return new ArrayList<>();  
    }  
    //... méthodes protected tripsBy  
    // et getLoggedInUser  
}
```

```
public class TripService {
```

```
    public List<Trip> getTripsByUser(User user)
        throws UserNotLoggedInException{
        List<Trip> tripList = new ArrayList<Trip>();
        User loggedUser =
            UserSession.getInstance().getLoggedUser();
        boolean isFriend = false;
        if (loggedUser != null) {
            for (User friend : user.getFriends()) {
                if (friend.equals(loggedUser)) {
                    isFriend = true;
                    break;
                }
            }
            if (isFriend) {
                tripList = TripDAO.findTripsByUser(user);
            }
        }
        return tripList;
    } else {
        throw new UserNotLoggedInException();
    }
}
```

Avant



Refactoring

Improving the Design of Existing Code

Après

```
public class TripService {
```

```
    public List<Trip> getTripsByUser(User user)
        throws UserNotLoggedInException{
```

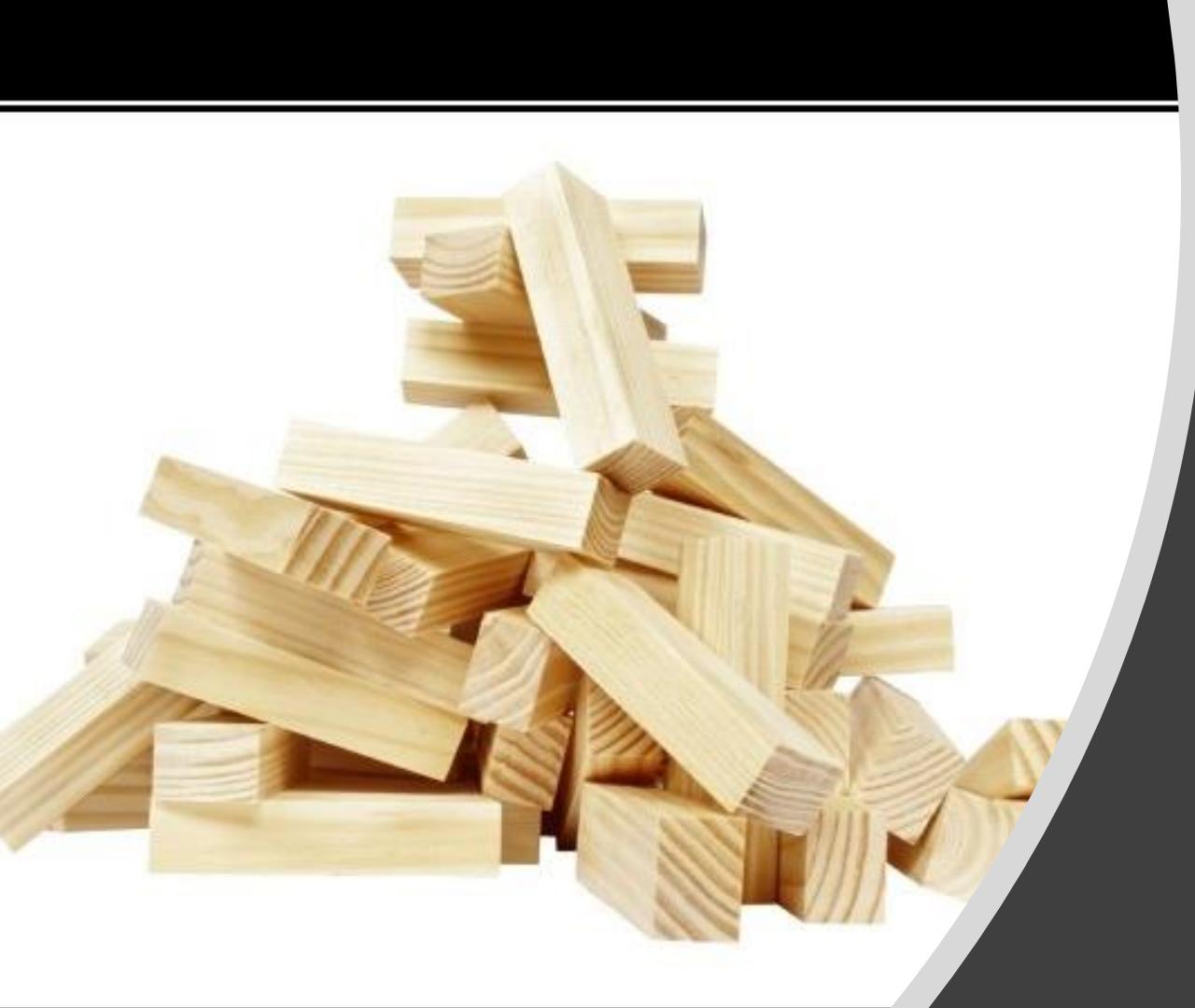
```
        User loggedUser = getLoggedInUser();
        validate(loggedUser);
```

```
        return user.isFriendWith(loggedUser)
            ? tripsBy(user)
            : noTrips();
    }
```

//... méthodes private validate et noTrips

*//... méthodes protected tripsBy et
getLoggedInUser*

```
}
```



SOLID
Software Development is not dead

Rendre le code plus SOLID

Problème de
conception

et rompre les
dépendances

Problème de couplage

Problème de
testabilité

Rompre la dépendance avec le Singleton

permettant
d'en savoir plus
l'utilisateur
en session

(UserSession)



```
public class TripService {  
  
    public List<Trip> getTripsByUser(User user)  
        throws UserNotLoggedInException {  
  
        User loggedUser = getLoggedInUser();  
        validate(loggedUser);  
  
        return user.isFriendWith(loggedUser)  
            ? tripsBy(user)  
            : noTrips();  
    }  
  
    //... méthodes noTrips et tripsBy  
  
    private void validate(User loggedUser) {  
        if (loggedUser == null) {  
            throw new UserNotLoggedInException();  
        }  
    }  
  
    protected User getLoggedInUser() {  
        return UserSession.getInstance().getLoggedUser();  
    }  
}
```

Bonne Pratique : Qui est Responsable ? (SRP)

La classe TripService traite des voyages
⇒
doit-elle s'occuper aussi de récupérer
l'utilisateur en session ?



Injecter l'utilisateur en session
comme paramètre de la
méthode getTripsByUser

Prod : injecter l'utilisateur en session comme paramètre

```
public class TripService {  
  
    public List<Trip> getTripsByUser(User user)  
        throws UserNotLoggedInException{  
  
        User loggedUser = getLoggedInUser();  
        validate(loggedUser);  
  
        return user.isFriendWith(loggedUser)  
            ? tripsBy(user)  
            : noTrips();  
    }  
  
    //... méthodes noTrips et tripsBy  
  
    private void validate(User loggedUser) {  
        if (loggedUser == null) {  
            throw new UserNotLoggedInException();  
        }  
    }  
  
    protected User getLoggedInUser() {  
        return UserSession.getInstance().getLoggedUser();  
    }  
}
```

```
public class TripService {  
  
    public List<Trip> getTripsByUser(User user,  
        User loggedInUser)  
        throws UserNotLoggedInException{  
  
        User loggedUser = getLoggedInUser();  
        validate(loggedUser);  
  
        return user.isFriendWith(loggedUser)  
            ? tripsBy(user)  
            : noTrips();  
    }  
  
    //... méthodes noTrips et tripsBy  
  
    private void validate(User loggedUser) {  
        if (loggedUser == null) {  
            throw new UserNotLoggedInException();  
        }  
    }  
  
    protected User getLoggedInUser() {  
        return UserSession.getInstance().getLoggedUser();  
    }  
}
```

Refactoring automatique
à l'IDE :
Change Method Signature

Prod : Faire en sorte de bien utiliser le paramètre injecté

```
public class TripService {  
  
    public List<Trip> getTripsByUser(User user,  
                                      User loggedInUser)  
        throws UserNotLoggedInException{  
  
        User loggedUser = getLoggedInUser();  
        validate(loggedUser);  
        return user.isFriendWith(loggedUser)  
            ? tripsBy(user)  
            : noTrips();  
    }  
  
    //... méthodes noTrips et tripsBy  
  
    private void validate(User loggedUser) {  
        if (loggedUser == null) {  
            throw new UserNotLoggedInException();  
        }  
    }  
  
    protected User getLoggedInUser() {  
        return UserSession.getInstance().getLoggedUser();  
    }  
}
```

loggedInUser

loggedInUser

Modifications manuelles

```
public class TripService {  
  
    public List<Trip> getTripsByUser(User user,  
                                      User loggedInUser)  
        throws UserNotLoggedInException{  
  
        validate(loggedInUser);  
        return user.isFriendWith(loggedInUser)  
            ? tripsBy(user)  
            : noTrips();  
    }  
  
    //... méthodes noTrips et tripsBy  
  
    private void validate(User loggedUser) {  
        if (loggedUser == null) {  
            throw new UserNotLoggedInException();  
        }  
    }  
}
```

Code test : remplacer les null par des valeurs appropriés

```
public class TripServiceTest {  
  
    private static final User GUEST = null;  
    private static final User UNUSED_USER = null;  
    private static final User LOGGED_IN_USER = new User();  
    private static final User ANOTHER_USER = new User();  
    private static final Trip TO_ITALY = new Trip();  
    private static final Trip TO_SPAIN = new Trip();  
  
    private User loggedInUser;  
    TripService tripService;  
  
    @Before  
    public void setUp() {  
        tripService = new TestableTripService();  
        loggedInUser = LOGGED_IN_USER;  
    }  
  
    @Test(expected = UserNotLoggedInException.class)  
    public void should_throw_an_exception_when_user_is_not_logged_in()  
    {  
        loggedInUser = GUEST;  
        tripService.getTripsByUser(UNUSED_USER, loggedInUser);  
    }  
  
    @Test  
    public void  
    should_not_return_any_trips_when_users_are_not_friends() {  
        User friend = aUser()  
            .friendsWith(ANOTHER_USER)  
            .withTrips(TO_ITALY)  
            .build();  
    }  
}
```

```
List<Trip> friendTrips = tripService.getTripsByUser(friend, null);  
assertThat(friendTrips).hasSize(0);  
}  
  
@Test  
public void should_return_trip_when_users_are_friends() {  
  
    User friend = aUser()  
        .friendsWith(ANOTHER_USER, loggedInUser)  
        .withTrips(TO_ITALY, TO_SPAIN)  
        .build();  
  
    List<Trip> friendTrips = tripService.getTripsByUser(friend, null);  
    assertThat(friendTrips).hasSize(2);  
    assertThat(friendTrips).contains(TO_ITALY, TO_SPAIN);  
}  
  
  
private class TestableTripService extends TripService {  
  
    @Override  
    protected User getLoggedInUser() {  
        return loggedInUser;  
    }  
  
    @Override  
    protected List<Trip> tripsBy(User user) {  
        return user.trips();  
    }  
}
```

...et supprimer la couture



Code test : Utilité variable globale loggedInUser vs Lisible

```
public class TripServiceTest {  
  
private static final User GUEST = null;  
private static final User UNUSED_USER = null;  
private static final User LOGGED_IN_USER = new User();  
private static final User ANOTHER_USER = new User();  
private static final Trip TO_ITALY = new Trip();  
private static final Trip TO_SPAIN = new Trip();  
  
private User loggedInUser;  
TripService tripService;  
  
@Before  
public void setUp() {  
tripService = new TestableTripService();  
loggedInUser = LOGGED_IN_USER;  
}  
  
@Test(expected = UserNotLoggedInException.class)  
public void should_throw_an_exception_when_user_is_not_logged_in() {  
loggedInUser = GUEST;  
tripService.getTripsByUser(UNUSED_USER, loggedInUser);  
}  
  
@Test  
public void should_not_return_any_trips_when_users_are_not_friends()  
{  
User friend = aUser()  
    .friendsWith(ANOTHER_USER)  
    .withTrips(TO_ITALY)  
    .build();  
    GUEST
```

```
List<Trip> friendTrips = tripService.getTripsByUser(friend, loggedInUser);  
assertThat(friendTrips).hasSize(0);  
}  
  
@Test  
public void should_return_trip_when_users_are_friends() {  
User friend = aUser()  
    .friendsWith(ANOTHER_USER, loggedInUser)  
    .withTrips(TO_ITALY, TO_SPAIN)  
    .build();  
  
List<Trip> friendTrips = tripService.getTripsByUser(friend, loggedInUser);  
assertThat(friendTrips).hasSize(2);  
assertThat(friendTrips).contains(TO_ITALY, TO_SPAIN);  
}  
  
private class TestableTripService extends TripService {  
  
    @Override  
    protected List<Trip> tripsBy(User user) {  
        return user.trips();  
    }  
}
```





Améliorer la lisibilité

```
public class TripService {  
  
    public List<Trip> getTripsByUser(User user,  
                                      User loggedInUser)  
        throws UserNotLoggedInException{  
  
        validate(loggedInUser);  
  
        return user.isFriendWith(loggedInUser)  
            ? tripsBy(user)  
            : noTrips();  
    }  
  
    private void validate(User loggedInUser){  
        if (loggedInUser == null) {  
            throw new UserNotLoggedInException();  
        }  
    }  
  
    private ArrayList<Trip> noTrips() {  
        return new ArrayList<Trip>();  
    }  
  
    protected List<Trip> tripsBy(User user) {  
        return TripDAO.findTripsByUser(user);  
    }  
}
```

Rename IDE
user → friend

Rename IDE
loggedInUser →
loggedInUser

```
public class TripService {  
  
    public List<Trip> getTripsByUser(User friend,  
                                      User loggedInUser)  
        throws UserNotLoggedInException{  
  
        validate(loggedInUser);  
  
        return friend.isFriendWith(loggedInUser)  
            ? tripsBy(friend)  
            : noTrips();  
    }  
  
    private void validate(User loggedInUser){  
        if (loggedInUser == null) {  
            throw new UserNotLoggedInException();  
        }  
    }  
  
    private ArrayList<Trip> noTrips() {  
        return new ArrayList<Trip>();  
    }  
  
    protected List<Trip> tripsBy(User user) {  
        return TripDAO.findTripsByUser(user);  
    }  
}
```



Rompre la dépendance avec la couche d'accès aux données

(TripDAO)



```

public class TripService {

    public List<Trip> getTripsByUser(User friend,
                                      User loggedInUser)
        throws UserNotLoggedInException {
        validate(loggedInUser);

        return friend.isFriendWith(loggedInUser)
            ? tripsBy(friend)
            : noTrips();
    }

    private void validate(User loggedInUser) {
        if (loggedInUser == null) {
            throw new UserNotLoggedInException();
        }
    }

    private ArrayList<Trip> noTrips() {
        return new ArrayList<Trip>();
    }

    protected List<Trip> tripsBy(User user) {
        return TripDAO.findTripsByUser(user);
    }
}

```

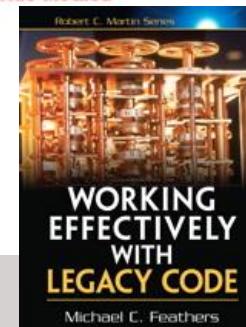
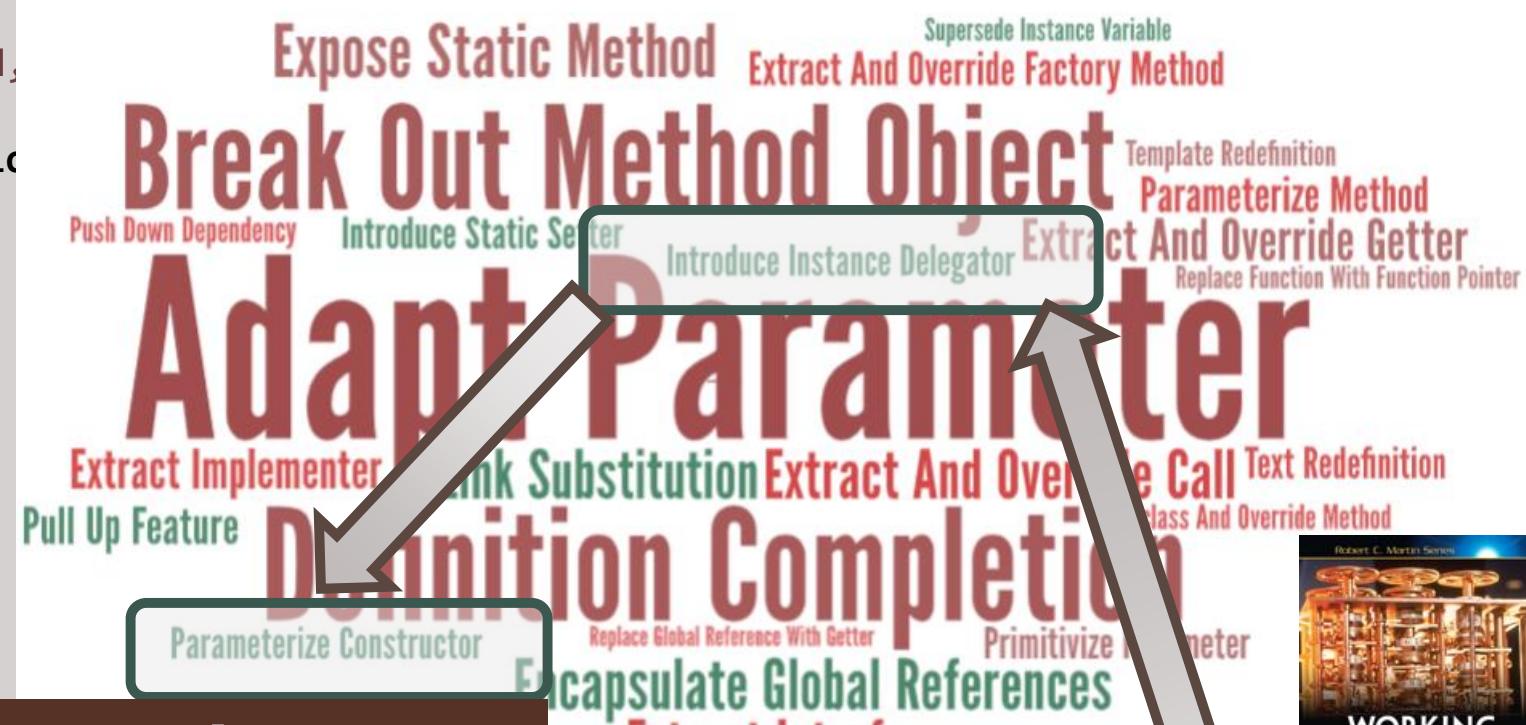
Bonne Pratique :
Injecter la dépendance

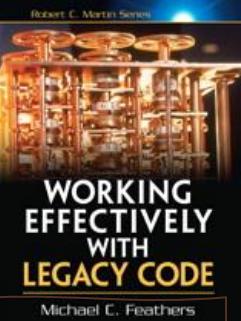


Casser la dépendance
à la couche d'accès aux données

Bonne Pratique :
dépendance déterminée dynamiquement à l'exécution

Faire disparaître statique ⇒ Méthode d'instance (plus facilement mockable)





Dependency-Breaking Techniques

Introduce Instance Delegator

findTripsByUser
dans la classe **TripDAO**

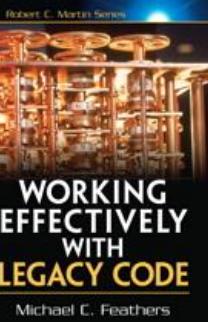
1. Identify a static method that is problematic to use in a test.

2. Create an instance method for the method on the class.

Remember to Preserve Signatures.

Make the instance method delegate to the static method.

3. Find places where the static methods are used in the class you have under test. Use Parameterize Method or another dependency-breaking technique **to supply an instance** to the location where the static method call was made.



Introduce Instance Delegator

2. Create en instance method (Zoom TripDAO)

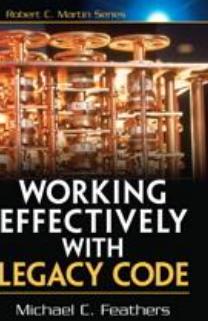
```
public class TripDAO {  
  
    public static List<Trip> findTripsByUser(User user) {  
        throw new CollaboratorCallException("TripDAO should not be invoked on an unit test.");  
    }  
}
```

```
public List<Trip> tripsBy(User user) {  
  
    return findTripsByUser(user);  
}  
}
```

Créer méthode d'instance :

- Signature identique
- Nom qui conserve intention métier (sens fonctionnel)

**La nouvelle méthode d'instance (nouvelle porte)
doit invoquer l'ancienne méthode statique (redirection vers le legacy)**



Introduce Instance Delegator

2. Create en instance method

Vérifier le *bon* comportement par un test

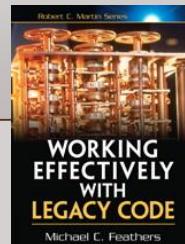
```
public class TripDAO {  
  
    public static List<Trip> findTripsByUser(User user) {  
        throw new CollaboratorCallException("TripDAO should not be invoked on an unit test.");  
    }  
  
    public List<Trip> tripsBy(User user) {  
        return findTripsByUser(user);  
    }  
}
```

```
public class TripDAOTest {  
  
    @Test(expected = CollaboratorCallException.class)  
    public void should_throw_exception_when_retrieving_users_trips() {  
        new TripDAO().tripsBy(new User());  
    }  
}
```



Introduce Instance Delegator

3. Find places where the static methods are used in the class you have under test. Use Parameterize Method or another dependency-breaking technique to supply an instance to the location where the static method call was made.



Remplacer les appels statiques par des appels
à la méthode d'instance (dans la classe sous test **TripService**)



Fournir un objet de type **TripDAO** à la classe **TripService**

Injection de dépendance
via IoC (Inversion of Control)
d'un framework tel que Spring
(**@Intent @autowired**)



Injection de dépendance
manuelle à l'aide de la
dependency-breaking technique
Parametrize Constructor



Dependency-Breaking Techniques

Parametrize Constructor

Injection **TripDAO**
dans la classe **TripService**

1. Identify the constructor that you want to parameterize and make a copy of it.
2. Add a parameter to the constructor for the object whose creation you are going to replace. Remove the object creation and add an assignment from the parameter to the instance variable for the object.
3. If you can call a constructor from a constructor in your language, remove the body of the old constructor and replace it with a call to the old constructor. Add a new expression to the call of the new constructor in the old constructor. If you can't call a constructor from another constructor in your language, you may have to extract any duplication among the constructors to a new method.

Dependency-Breaking Techniques

Parametrize Constructor

Injection **TripDAO**
dans la classe **TripService**

```
public class TripService {  
    TripDAO tripDAO;  
  
    public TripService(){  
        this (new TripDAO());  
    }  
  
    public TripService(TripDAO tripDAO){  
        this.tripDAO=tripDAO;  
    }  
  
    public List<Trip> getTripsByUser(User user, User loggedIn  
        throws UserNotLoggedInException { //...}  
  
    private ArrayList<Trip> noTrips() { //...}  
  
    private void validate(User loggedUser) { //...}  
  
    protected List<Trip> tripsBy(User user) {  
        return TripDAO.findTripsByUser(user);  
    }  
}
```

1. Identifier EXPLICITEMENT le constructeur à copier

2. Ajouter un paramètre au constructeur
et implémenter après déclaration d'une
nouvelle variable d'instance TripDAO

3. L'ancien constructeur retrouve l'implémentation qu'il avait avant
en faisant appel au nouveau constructeur

Introduce Instance Delegator : Retour au point 3 ...

3. Find places where the static methods are used in the class you have under test ...

```
public class TripService {  
  
    TripDAO tripDAO;  
  
    public TripService() {  
        this(new TripDAO());  
    }  
  
    public TripService(TripDAO tripDAO) {  
        this.tripDAO = tripDAO;  
    }  
  
    public List<Trip> getTripsByUser(User user, User loggedInUser)  
        throws UserNotLoggedInException { //... }  
  
    private ArrayList<Trip> noTrips() { //... }  
  
    private void validate(User loggedUser) { //... }  
  
    protected List<Trip> tripsBy(User user) {  
        return TripDAO.findTripsByUser(user),  
               tripDAO.tripsBy(user)  
    }  
}
```

Code de prod



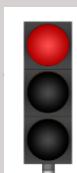
Remplacer la méthode statique
par la méthode d'instance

Introduce Instance Delegator : Retour au point 3 (Code de test)

```
public class TripServiceTest {  
  
    private static final User GUEST = null;  
    private static final User UNUSED_USER = null;  
    private static final User LOGGED_IN_USER = new User();  
    private static final User ANOTHER_USER = new User();  
    private static final Trip TO_ITALY = new Trip();  
    private static final Trip TO_SPAIN = new Trip();  
  
    TripService tripService;  
  
    @Before  
    public void setUp() {  
        tripService = new TestableTripService();  
    }  
  
    @Test(expected = UserNotLoggedInException.class)  
    public void should_throw_an_exception_when_user_is_not_logged_in()  
    {  
        tripService = new TripService();  
    }  
  
    @Test  
    public void should_not_return_any_trips_when_users_are_not_friends()  
    {  
        User friend = aUser()  
            .friendsWith(ANOTHER_USER)  
            .withTrips(TO_ITALY)  
            .build();  
    }  
}
```

```
tripService = new TripService();  
  
List<Trip> friendTrips = tripService.getTripsByUser(friend,  
    LOGGED_IN_USER);  
  
assertThat(friendTrips).hasSize(0);  
}  
  
@Test  
public void should_return_trip_when_users_are_friends() {  
  
    User friend = aUser()  
        .friendsWith(ANOTHER_USER, LOGGED_IN_USER)  
        .withTrips(TO_ITALY, TO_SPAIN)  
        .build();  
  
    tripService = new TripService();  
  
    List<Trip> friendTrips = tripService.getTripsByUser(friend,  
        LOGGED_IN_USER);  
  
    assertThat(friendTrips).hasSize(2);  
    assertThat(friendTrips).contains(TO_ITALY, TO_SPAIN);  
}  
  
private class TestableTripService extends TripService {  
    @Override  
    protected List<Trip> tripsByUser(User user)  
    {  
        return user.trips();  
    }  
}
```



CollaboratorCallException

Il faut injecter le TripDAO !!!

Injection de TripDAO dans le code de test via un Mock

```
public class TripServiceTest {  
    //...  
    TripService tripService;  
  
    @Before  
    public void setUp() {  
        tripService = new TestableTripService();  
    }  
  
    // Test should_throw_an_exception_when_user_is_not_logged_in  
    // Test should_not_return_any_trips_when_users_are_not_friends  
  
    @Test  
    public void should_return_trip_when_users_are_friends() {  
        TripDAO tripDAO = Mockito.mock(TripDAO.class);  
        User friend = aUser()  
            .friendsWith(ANOTHER_USER, LOGGED_IN_USER)  
            .withTrips(TO_ITALY, TO_SPAIN)  
            .build();  
        Mockito.when(tripDAO.tripsBy(friend)).thenReturn(friend.trips());  
        tripService = new TripService(tripDAO);  
        List<Trip> friendTrips = tripService.getTripsByUser(friend, LOGGED_IN_USER);  
  
        assertThat(friendTrips).hasSize(2);  
        assertThat(friendTrips).contains(TO_ITALY, TO_SPAIN);  
    }  
  
    private class TestableTripService extends TripService {  
        @Override  
        protected List<Trip> tripsBy(User user) {  
            return user.trips();  
        }  
    }  
}
```

1. Déclarer le mock

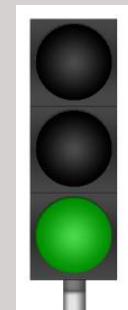
2. Eduquer le mock

3. Injecter le mock



Supprimer la couture après injection du tripDAO

```
public class TripServiceTest {  
    //...  
    TripService tripService;  
  
    @Before  
    public void setUp() {  
        tripService = new TestableTripService();  
    }  
  
    // Test should_throw_an_exception_when_user_is_not_logged_in  
    // Test should_not_return_any_trips_when_users_are_not_friends  
  
    @Test  
    public void should_return_trip_when_users_are_friends() {  
  
        TripDAO tripDAO = Mockito.mock(TripDAO.class);  
        User friend = aUser()  
            .friendsWith(ANOTHER_USER, LOGGED_IN_USER)  
            .withTrips(TO_ITALY, TO_SPAIN)  
            .build();  
        Mockito.when(tripDAO.tripsBy(friend)).thenReturn(friend.trips());  
        tripService = new TripService(tripDAO);  
  
        List<Trip> friendTrips = tripService.getTripsByUser(friend, LOGGED_IN_USER);  
  
        assertThat(friendTrips).hasSize(2);  
        assertThat(friendTrips).contains(TO_ITALY, TO_SPAIN);  
    }  
  
    private class TestableTripService extends TripService {  
        @Override  
        protected List<Trip> tripsBy(User user) {  
            return user.trips();  
        }  
    }  
}
```



Dernier petit nettoyage/peaufinage

```
public class TripService {  
    TripDAO tripDAO;  
  
    public TripService() {  
        this(new TripDAO());  
    }  
  
    public TripService(TripDAO tripDAO) {  
        this.tripDAO = tripDAO;  
    }  
  
    public List<Trip> getFriendTrips(User friend, User loggedInUser)  
        throws UserNotLoggedInException {  
        validate(loggedInUser);  
  
        return friend.isFriendWith(loggedInUser)  
            ? tripsBy(friend)  
            : noTrips();  
    }  
  
    private ArrayList<Trip> noTrips() {  
        return new ArrayList<Trip>();  
    }  
  
    private void validate(User loggedUser) {  
        if (loggedUser == null) {  
            throw new UserNotLoggedInException();  
        }  
    }  
  
    private List<Trip> tripsBy(User user) {  
        return tripDAO.tripsBy(user);  
    }  
}
```

Intention métier plus explicite

*A l'image du potier
qui travaille son argile en continu,
le développeur travaille
son code en permanence.*

Benoit Gantaume



Révision visibilité
(après suppression couture)



```
public class TripService {

    public List<Trip> getTripsByUser(User user)
        throws UserNotLoggedInException {
        List<Trip> tripList = new ArrayList<Trip>();
        User loggedUser =
            UserSession.getInstance().getLoggedUser();
        boolean isFriend = false;
        if (loggedUser != null) {
            for (User friend : user.getFriends()) {
                if (friend.equals(loggedUser)) {
                    isFriend = true;
                    break;
                }
            }
            if (isFriend) {
                tripList = TripDAO.findTripsByUser(user);
            }
        }
        return tripList;
    } else {
        throw new UserNotLoggedInException();
    }
}
```

Avant

```
public class TripService {
    TripDAO tripDAO;

    public TripService() {
        this(new TripDAO());
    }

    public TripService(TripDAO tripDAO) {
        this.tripDAO = tripDAO;
    }

    public List<Trip> getFriendTrips(User friend, User loggedInUser)
        throws UserNotLoggedInException {
        validate(loggedInUser);

        return friend.isFriendWith(loggedInUser)
            ? tripsBy(friend)
            : noTrips();
    }

    private ArrayList<Trip> noTrips() {
        return new ArrayList<Trip>();
    }

    private void validate(User loggedUser) {
        if (loggedUser == null) {
            throw new UserNotLoggedInException();
        }
    }

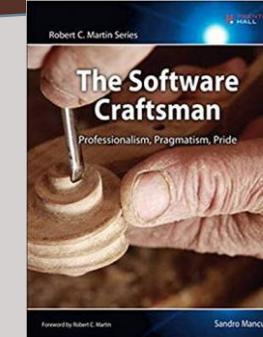
    private List<Trip> tripsBy(User user) {
        return tripDAO.tripsBy(user);
    }
}
```

Références

Idée originale de Sandro Mancuso :

<https://github.com/sandromancuso/trip-service-kata>

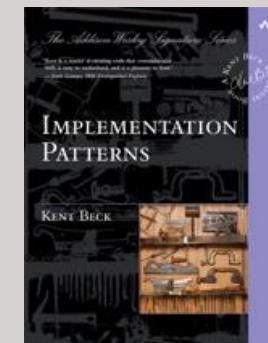
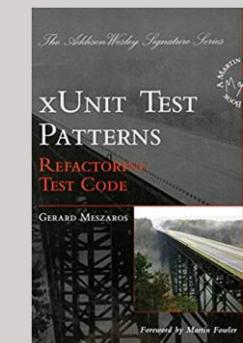
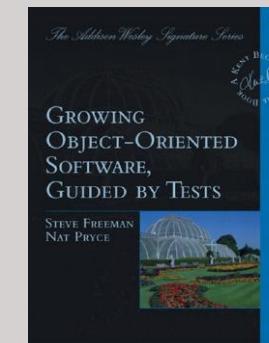
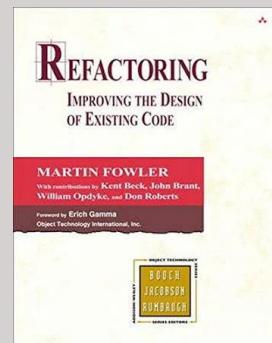
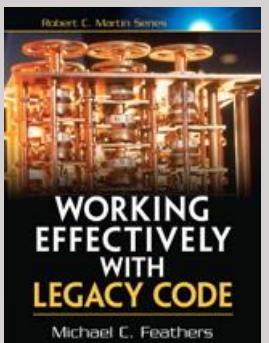
(existe pour différents langages)



- <https://codurance.com/2011/07/16/testing-legacy-hard-wired-dependencies>
- https://codurance.com/2011/07/18/testing-legacy-hard-wired-dependencies_17
- <https://blog.engineering.publicissapient.fr/2015/01/23/legacy-code-se-defaire-des-dependances-statiques>
- <https://github.com/murex/mikado-testbuilders-kata/blob/master/docs/references/Test%20Data%20Builders.md>

Vidéos :

- **Testing and Refactoring Legacy Code** de Sandro Mancuso : https://www.youtube.com/watch?v=_NnElPO5BU0
- **Legacy Club** de Bruno Boucard et Thomas Pierrain : <https://vimeo.com/215728093>
(à partir de 33:43 pour un live coding sur l'approche qui permet de casser la dépendance à tripDAO)



Isabelle BLASQUEZ

Crédits Photos

<https://fr.seacons.com/verifiez-utilisateur-icon>

<https://fr.seacons.com/icones-des-utilisateurs-65/>

[https://publicdomainvectors.org/fr/gratuitement-des-vecteurs/Image-d'E2%80%99ic%C3%B4ne-utilisateur-f%C3%A9minin/71148.html](https://publicdomainvectors.org/fr/gratuitement-des-vecteurs/Image-d%E2%80%99ic%C3%B4ne-utilisateur-f%C3%A9minin/71148.html)

https://www.aeroportlimoges.com/fileadmin/_processed_/7/2/csm_voyages-2_d1856d84b8.jpg

<https://blog.malwarebytes.com/cybercrime/exploits/2018/01/meltdown-and-spectre-fallout-patching-problems-persist/>

<https://icon-icons.com/fr/icon/cible-objectif-la-reussite-le-but-le-tir-a-arc/55987>

<https://www.classe-numerique.fr/types-dactivites/flashcards/vois-tu-un-axe-de-symetrie>

<https://medium.com/swlh/refactoring-guard-clauses-8f8e45fbc41e>

<https://fr.seacons.com/switch-icon-3/>

https://stapp.space/content/images/2016/02/refactoring_poster-r723da0bee0b0423799654d8a59350032_i5v9b_8byvr_1024.jpg