



# Mettre en œuvre des tests structurels en Java

Fabrice AMBERT – [fabrice.ambert@femto-st.fr](mailto:fabrice.ambert@femto-st.fr)  
Fabrice BOUQUET – [fabrice.bouquet@femto-st.fr](mailto:fabrice.bouquet@femto-st.fr)  
Fabien PEUREUX – [fabien.peureux@femto-st.fr](mailto:fabien.peureux@femto-st.fr)  
Ivan ENDERLIN, Jean-Marie GAUTHIER  
Cédric JOFFROY, Alexandre VERNOTTE

# Outils mis en œuvre





# Anatomie d'un test unitaire

```
@Test    Annotation désignant la méthode comme un test
public void testXXX() {
    //Define    Instructions de mise en contexte
    //When    Instruction sous test
    //Then
    Observation et vérification de l'oracle
}
```

# Tester l'apparition d'une exception

```
@Test (expected = ClasseException.class)
public void testXXX() {
    //Define

    //When

}
```

# Quelques annotations

@Ignore

```
@Ignore
@Test(expected = ClasseException.class)
public void testXXX() {
    //Define

    //When

}
```

Lors de l'exécution de la suite, le test est ignoré mais est mentionné dans le rapport d'exécution

# Quelques annotations - 2



@Before

@After

@Before

```
public void setUp() {
```

```
}
```

Cette méthode est exécuté avant chaque test de la suite

@After

```
public void tearDown() {
```

```
}
```

Celle ci après chaque test de la suite



# Test en isolation

```
public class Location {  
    private Film film ;  
    private Client client ;  
    ...  
    public float montant(int duree) {  
        if (client.getCat() == PRIVILEGE)  
            return film.prixJour()*(duree-1) ;  
        else ...  
    }  
}
```

```
public class Film {  
    private Categorie categorie ;  
    private String titre ;  
    ...  
    public float prixJour() {  
        switch (categorie) {  
            case Categorie.NOUVEAUTE :  
                return categorie.prixBase() * ... ; ...  
        }  
    }  
}
```



# Test en isolation

```
@Test
public void testMontant() {
    Film film = Mockito.mock(Film.class) ;
    Mockito.when(film.prixJour()).thenReturn(3.5) ;
    Client client = Mockito.mock(Client.class) ;
    Mockito.when(client.getCat()).thenReturn(PRIVILEGE) ;
    Location loc = new Location(film, client) ;

    Assert.assertEquals(3.5, loc.montant(2)) ;
}
```

Le mock retourne les réponses attendues par la classe sous test sans faire appel à la classe mockée





# Test en isolation

@Test

```
public void testMontant() {  
    Film film = Mockito.mock(Film.class) ;  
    Mockito.when(film.prixJour()).thenReturn(3.5) ;  
    Client client = Mockito.mock(Client.class) ;  
    Mockito.when(client.getCat()).thenReturn(PRIVILEGE) ;  
    Location loc = new Location(film, client) ;  
  
    loc.montant(2) ;  
  
    Mockito.verify(film).prixJour();  
}
```

Le mock mémorise les appels qui lui sont fait. On peut ensuite l'interroger sur les invocations auxquelles il a répondu.

# Merci pour votre attention...



*"Testing is always model-based!"*  
Robert Binder

