



université  
PARIS-SACLAY

université  
PARIS-SACLAY  
IUT D'ORSAY

# QUALITÉ DE DÉVELOPPEMENT

## LES TESTS UNITAIRES EN JAVA

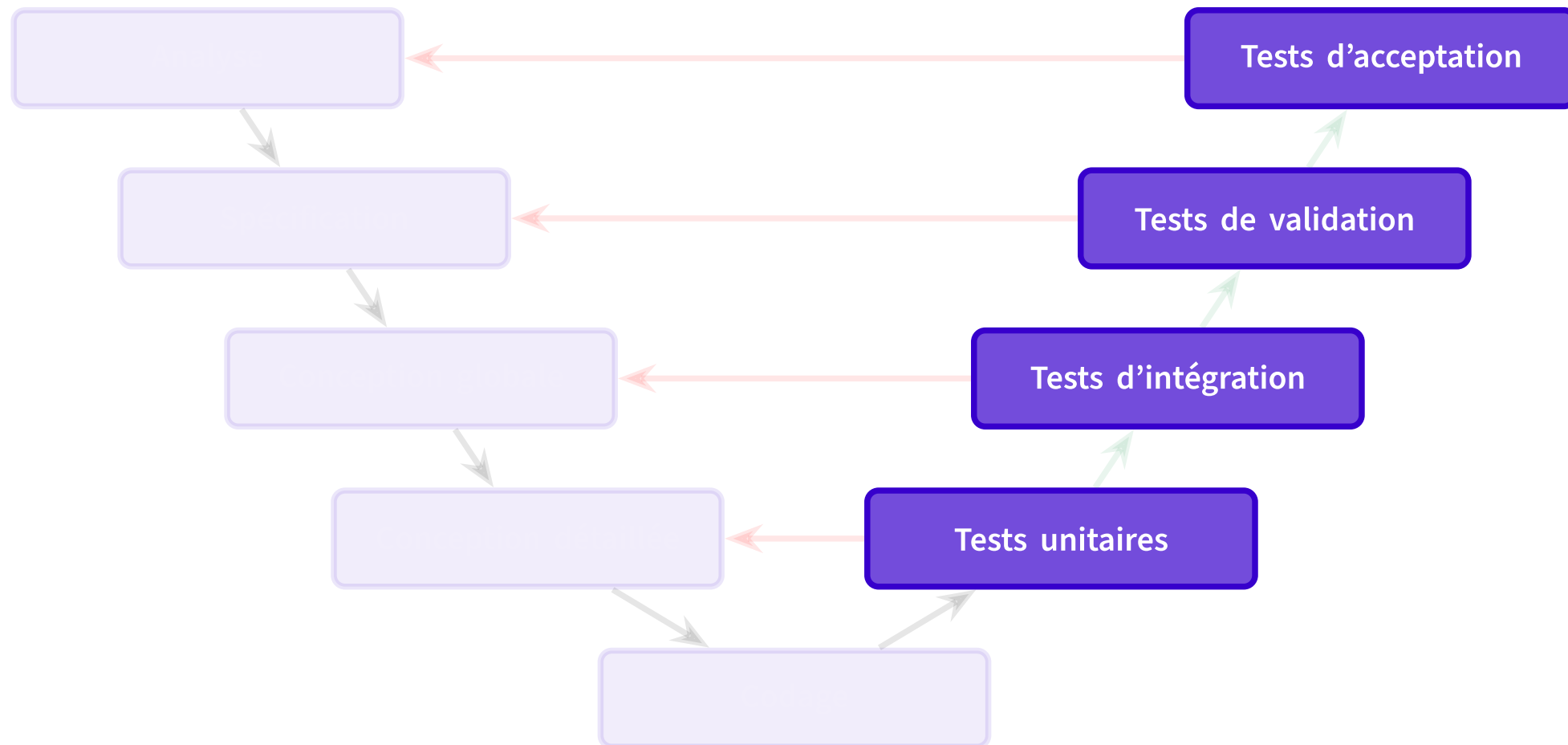
🎓 2A - Bachelor Universitaire de Technologie  
🏛️ IUT d'Orsay - Université Paris-Saclay - 2023/2024



**Idir AIT SADOUNE**

[idir.ait-sadoune@universite-paris-saclay.fr](mailto:idir.ait-sadoune@universite-paris-saclay.fr)

# PROCESSUS DE DÉVELOPPEMENT EN V



# LES ACTIVITÉS DE TESTS

Nous distinguons quatre types d'activités de tests :

1. **Les tests unitaires** : vérifier séparément le fonctionnement des modules d'un logiciel.
2. **Les tests d'intégration** : regrouper l'ensemble des modules et valider le fonctionnement global du logiciel.
3. **Les tests de validation** : valider la conformité du logiciel par rapport aux spécifications.
4. **Les tests d'acceptation** : vérifier avec le client que la solution proposée est conforme à ses exigences.

# DANS CE COURS

Nous distinguons quatre types d'activités de tests :

1. **Les tests unitaires** : vérifier séparément le fonctionnement des modules d'un logiciel.
2. **Les tests d'intégration** : regrouper l'ensemble des modules et valider le fonctionnement global du logiciel.
3. **Les tests de validation** : valider la conformité du logiciel par rapport aux spécifications.
4. **Les tests d'acceptation** : vérifier avec le client que la solution proposée est conforme à ses exigences.

# LA PHILOSOPHIE DU TEST

- Pas de différence entre test et débogage.
  - ▢▢▢▢➡ déboguer avec des jeux d'entrées
- Le test sert à montrer que le logiciel fonctionne.
  - ▢▢▢▢➡ exécuter une suite de tests sans défaillance
- Le test montre la présence d'erreurs mais pas leur absence.
  - ▢▢▢▢➡ permettre de diminuer le risque.

# LES TESTS UNITAIRES

- **Un test unitaire** est **un script** permettant de **valider** le fonctionnement d'une partie d'un module.
  - ➡ une méthode, une class, une interface, ...
- En **Java**, le framework (**JUnit**), offre des fonctionnalités permettant le développement des tests unitaires.

# LES ANNOTATIONS

- **JUnit** utilise un système d'annotations définissant le comportement d'une méthode dans une classe de test :
  - **@Test** : spécifie que la méthode annotée est un test.
  - **@Before** ou **@BeforeEach** (**JUnit5**) : spécifie que la méthode annotée sera exécutée avant chaque méthode de test.
  - **@BeforeClass** ou **@BeforeAll** (**JUnit5**): spécifie que la méthode annotée sera exécutée une fois avant toutes les méthodes de test.

# LES ANNOTATIONS

- **JUnit** utilise un système d'annotations définissant le comportement d'une méthode dans une classe de test :
  - ...
  - **@After** ou **@AfterEach** (**JUnit5**) : spécifie que la méthode annotée sera exécutée après chaque méthode de test.
  - **@AfterClass** ou **@AfterAll** (**JUnit5**): spécifie que la méthode annotée sera exécutée une fois après toutes les méthodes de test.



# LA CLASSE ASSERTIONS

La documentation de la classe [Assertions](#)

# LA CLASSE ASSERTIONS

- `void assertTrue(boolean condition)`  
vérifie qu'une condition est vraie.
- `void assertFalse(boolean condition)`  
vérifie qu'une condition est fausse.
- `void assertNotNull(Object object)`  
vérifie qu'un objet n'est pas null.
- `void assertNull(Object object)`  
vérifie qu'un objet est null.
- `void assertEquals(Object expected, Object actual)`  
vérifie l'égalité entre deux objets ou deux types primitifs.

# LA CLASSE ASSERTIONS

- ...
- `void assertEquals(Object expected, Object actual)`  
vérifie l'égalité entre deux références d'objets.
- `void assertNotSame(Object expected, Object actual)`  
vérifie que deux références d'objets sont différentes.
- `void assertEquals(expectedArray, resultArray)`  
vérifie que deux tableaux sont égaux.
- `void fail(String)`  
force un test à l'échec.

# CODE JAVA À TESTER

```
1 public class Contact {
2     private String firstName;
3     private String lastName;
4     private String phoneNumber;
5
6     public Contact(String firstName, String lastName, String phoneNumber) {
7         this.firstName = firstName;
8         this.lastName = lastName;
9         this.phoneNumber = phoneNumber;
10    }
11
12    public String getFirstName() {
13        return firstName;
14    }
15
16    public void setFirstName(String firstName) {
17        this.firstName = firstName;
18    }
19
20    public String getLastName() {
21        return lastName;
```

# CODE JAVA À TESTER

```
1 import java.util.Collection;
2 import java.util.Map;
3 import java.util.concurrent.ConcurrentHashMap;
4
5 public class ContactManager {
6
7     Map<String, Contact> contactList = new ConcurrentHashMap<String, Contact>();
8
9     public void addContact(String firstName, String lastName, String phoneNumber) {
10         Contact contact = new Contact(firstName, lastName, phoneNumber);
11         this.validateContact(contact);
12         this.checkIfContactAlreadyExist(contact);
13         this.contactList.put(this.generateKey(contact), contact);
14     }
15
16     public Collection<Contact> getAllContacts() {
17         return this.contactList.values();
18     }
19
20     private void checkIfContactAlreadyExist(Contact contact) {
21         if (this.contactList.containsKey(this.generateKey(contact)))
```

# PREMIER TEST JUNIT5

```
1 import org.junit.jupiter.api.*;
2 import static org.junit.jupiter.api.Assertions.*;
3
4 public class ContactManagerTest {
5
6     @Test
7     public void shouldCreateContact() {
8         ContactManager contactManager = new ContactManager();
9         contactManager.addContact("John", "Doe", "0123456789");
10
11         assertTrue(! contactManager.getAllContacts().isEmpty());
12         assertFalse(contactManager.getAllContacts().isEmpty());
13         assertEquals(1, contactManager.getAllContacts().size());
14     }
15 }
```

# CYCLE DE VIE D'UN TEST JUNIT

```
1 import org.junit.jupiter.api.*;
2 import org.junit.jupiter.api.condition.EnabledOnOs;
3 import org.junit.jupiter.api.condition.OS;
4 import org.junit.jupiter.params.ParameterizedTest;
5 import org.junit.jupiter.params.provider.CsvFileSource;
6 import org.junit.jupiter.params.provider.CsvSource;
7 import org.junit.jupiter.params.provider.MethodSource;
8 import org.junit.jupiter.params.provider.ValueSource;
9
10 import java.util.Arrays;
11 import java.util.List;
12
13 import static org.junit.jupiter.api.Assertions.assertEquals;
14 import static org.junit.jupiter.api.Assertions.assertFalse;
15
16 public class ContactManagerTest {
17
18     private ContactManager contactManager;
19
20     @BeforeAll
21     public static void setUpAll() {
```

# RÉPÉTER DES TESTS AVEC JUNIT

```
1 import org.junit.jupiter.api.*;
2 import org.junit.jupiter.api.condition.EnabledOnOs;
3 import org.junit.jupiter.api.condition.OS;
4 import org.junit.jupiter.params.ParameterizedTest;
5 import org.junit.jupiter.params.provider.CsvFileSource;
6 import org.junit.jupiter.params.provider.CsvSource;
7 import org.junit.jupiter.params.provider.MethodSource;
8 import org.junit.jupiter.params.provider.ValueSource;
9
10 import java.util.Arrays;
11 import java.util.List;
12
13 import static org.junit.jupiter.api.Assertions.assertEquals;
14 import static org.junit.jupiter.api.Assertions.assertFalse;
15
16 public class ContactManagerTest {
17
18     private ContactManager contactManager;
19
20     @BeforeAll
21     public static void setUpAll() {
```



# LES TESTS PARAMÉTRÉS

```
1 import org.junit.jupiter.api.*;
2 import org.junit.jupiter.api.condition.EnabledOnOs;
3 import org.junit.jupiter.api.condition.OS;
4 import org.junit.jupiter.params.ParameterizedTest;
5 import org.junit.jupiter.params.provider.CsvFileSource;
6 import org.junit.jupiter.params.provider.CsvSource;
7 import org.junit.jupiter.params.provider.MethodSource;
8 import org.junit.jupiter.params.provider.ValueSource;
9
10 import java.util.Arrays;
11 import java.util.List;
12
13 import static org.junit.jupiter.api.Assertions.assertEquals;
14 import static org.junit.jupiter.api.Assertions.assertFalse;
15
16 public class ContactManagerTest {
17
18     private ContactManager contactManager;
19
20     @BeforeAll
21     public static void setUpAll() {
```

# LES TESTS IGNORÉS

```
1 import org.junit.jupiter.api.*;
2 import org.junit.jupiter.api.condition.EnabledOnOs;
3 import org.junit.jupiter.api.condition.OS;
4 import org.junit.jupiter.params.ParameterizedTest;
5 import org.junit.jupiter.params.provider.CsvFileSource;
6 import org.junit.jupiter.params.provider.CsvSource;
7 import org.junit.jupiter.params.provider.MethodSource;
8 import org.junit.jupiter.params.provider.ValueSource;
9
10 import java.util.Arrays;
11 import java.util.List;
12
13 import static org.junit.jupiter.api.Assertions.assertEquals;
14 import static org.junit.jupiter.api.Assertions.assertFalse;
15
16 public class ContactManagerTest {
17
18     private ContactManager contactManager;
19
20     @BeforeAll
21     public static void setUpAll() {
```

# TESTS DES EXCEPTIONS AVEC JUNIT

Utilisation de la méthode `assertThrows`

```
1 import org.junit.jupiter.api.*;
2 import static org.junit.jupiter.api.Assertions.*;
3
4 public class ContactManagerTest {
5
6     @Test
7     public void shouldThrowRuntimeExceptionWhenFirstNameIsNull() {
8         ContactManager contactManager = new ContactManager();
9
10        Assertions.assertThrows(RuntimeException.class, () -> {
11            contactManager.addContact(null, "Doe", "0123456789");
12        });
13    }
14 }
```

# MERCI

[Retour à l'accueil](#) - [Retour au plan](#)