ATELIER : DEVOPS

Mohamed HAMMOUDA

PLAN DE L'ATELIER

1	INTRODUCTION AU DEVOPS
2	LE CONTRÔLE DES VERSIONS : GIT & GITLAB
3	LE CONTRÔLE DE QUALITÉ DES LOGICIELS
4	LES CONTENEURS APPLICATIVES : DOCKER
5	INTÉGRATION CONTINUE ET DÉPLOIEMENT CONTINU

PLAN DE L'ATELIER

1	INTRODUCTION AU DEVOPS
2	LE CONTRÔLE DES VERSIONS : GIT & GITLAB
3	LE CONTRÔLE DE QUALITÉ DES LOGICIELS
4	LES CONTENEURS APPLICATIVES : DOCKER
5	INTÉGRATION CONTINUE ET DÉPLOIEMENT CONTINU

TEST & QUALITÉ LOGICIELLE

LE TEST EN GÉNIE LOGICIEL

- As per ANSI/IEEE 1059, **Testing in Software Engineering** is a process of evaluating a software product to find whether the current software product **meets the required conditions or not.**
- The testing process involves evaluating the features of the software product for requirements in terms of any **missing requirements, bugs or errors, security, reliability and performance.**

- Exigences fonctionnelles
  - use cases
- Exigences non-fonctionnelles :
  - Security
  - Fiabilité
  - Performance
- Exigence de maintenabilité

TEST & QUALITÉ LOGICIELLE

POURQUOI TESTEZ-VOUS VOS LOGICIELS

TEST & QUALITÉ LOGICIELLE

POURQUOI TESTEZ-VOUS VOS LOGICIELS

TEST & QUALITÉ LOGICIELLE

POURQUOI TESTEZ-VOUS VOS LOGICIELS

TEST & QUALITÉ LOGICIELLE

POURQUOI TESTEZ-VOUS VOS LOGICIELS

Nissan cars recalled over 1 million cars from the market due to software failure in the airbag sensory detectors. There has been reported two accident due to this software failure.

- Détecter les anomalies fonctionnelles et non-fonctionelle avant la livraison finale au client

- Rentabilité

TEST & QUALITÉ LOGICIELLE

POURQUOI TESTEZ-VOUS VOS LOGICIELS

Nissan cars recalled over 1 million cars from the market due to software failure in the airbag sensory detectors. There has been reported two accident due to this software failure.

- Détecter les anomalies fonctionnelles et non-fonctionelles avant la livraison finale au client

- Rentabilité
- Satisfaction du client

TEST & QUALITÉ LOGICIELLE

POURQUOI TESTEZ-VOUS VOS LOGICIELS

Nissan cars recalled over 1 million cars from the market due to software failure in the airbag sensory detectors. There has been reported two accident due to this software failure.

- Détecter les anomalies fonctionnelles et non-fonctionelle avant la livraison finale au client

- Rentabilité
- Satisfaction du client
- Sécurité

TEST & QUALITÉ LOGICIELLE

POURQUOI TESTEZ-VOUS VOS LOGICIELS

Nissan cars recalled over 1 million cars from the market due to software failure in the airbag sensory detectors. There has been reported two accident due to this software failure.

- Détecter les anomalies fonctionnelles et non-fonctionelle avant la livraison finale au client

- Rentabilité
- Satisfaction du client
- Sécurité
- **Logiciel de qualité**

TEST & QUALITÉ LOGICIELLE

POURQUOI TESTEZ-VOUS VOS LOGICIELS

Logiciel de qualité

- Fonctionnel
- Performant
- Maintenable
- Fiable
- Sécurisé

TEST & QUALITÉ LOGICIELLE

COMMENT TESTEZ-VOUS VOS LOGICIELS


```
class Etudiant {
            public int IDEtudiant { get; set; }
            public string Groupe { get; set; }
            public double MoyenneGenerale { get; set; }
            ...
        }
```

```
interface IDAOEtudiant {
      List<Etudiant> GetEtudiants();
 }


class DAOEtudiant
            implements IDAOEtudiant
  { ...}
```

```
class TraitementEtudiant
       {
           IDAOEtudiant _dBEtudiant;
           public TraitementEtudiant (IDAOEtudiant dBEtudiant )
           {
               _dBEtudiant = dBEtudiant;
           }
           public  double CalculerMoyenneGenerale(String group)
           {
               double somme = 0; double cpt = 0;
               List<Etudiant> etudiants = _dBEtudiant.GetEtudiants();
               foreach (var item in etudiants)
               {
                   if (item.Groupe.Equals(group))
                       somme += item.MoyenneGenerale;
                   cpt++;
               }
               return somme / cpt;
           }
```

TEST & QUALITÉ LOGICIELLE

LES TESTS MANUELS

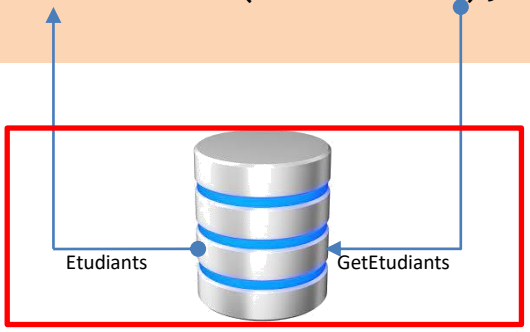
```
class Etudiant {
            public int IDEtudiant { get; set; }
            public string Groupe { get; set; }
            public double MoyenneGenerale { get;
set; }
            ...
        }
```

```
interface IDAOEtudiant {
      List<Etudiant> GetEtudiants();
 }

class DAOEtudiant  implements IDAOEtudiant
  { ...
}
```

```
class TraitementEtudiant
       {
           IDAOEtudiant _dBEtudiant;
           public TraitementEtudiant (IDAOEtudiant dBEtudiant )
           {
               _dBEtudiant = dBEtudiant;
           }
           public double CalculerMoyenneGenerale(String group)
           {
               double somme = 0; double cpt = 0;
               List<Etudiant> etudiants = _dBEtudiant.GetEtudiants();
               foreach (var item in etudiants)
               {
                   if (item.Groupe.Equals(group))
                       somme += item.MoyenneGenerale;
                   cpt++;
               }
               return somme / cpt;
           }
```

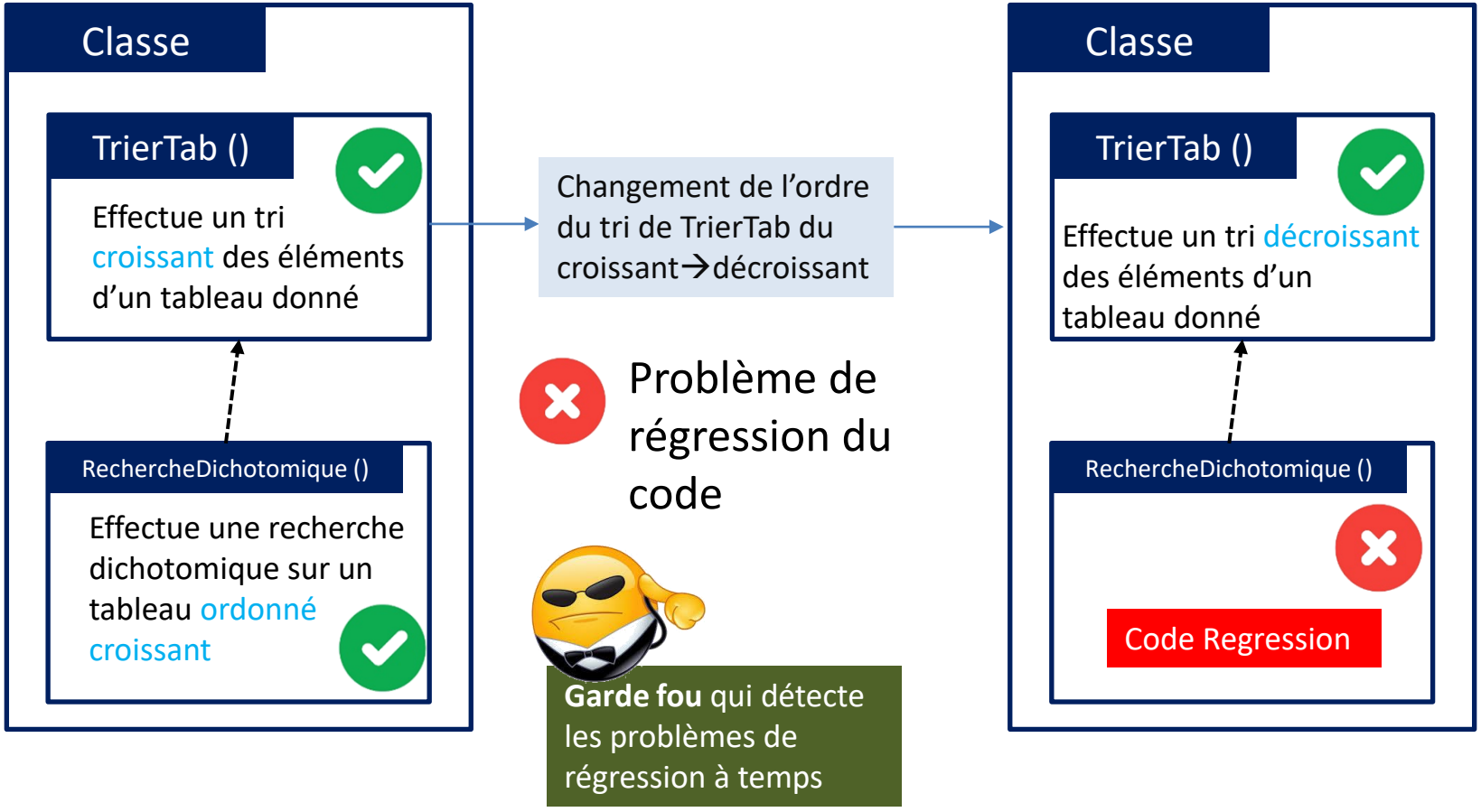
```
 static void Main(string[] args){

    IDAOEtudiant dbEtudiant = new DAOEtudiant();

    TraitementEtudiant traitementEtudiant = new TraitementEtudiant (dbEtudiant);

    double moyenne = traitementEtudiant.CalculerMoyenneGenerale(dbEtudiant);
}
```

- Perte de temps
- Ne tient pas compte de tous les scénarios possibles



TEST & QUALITÉ LOGICIELLE

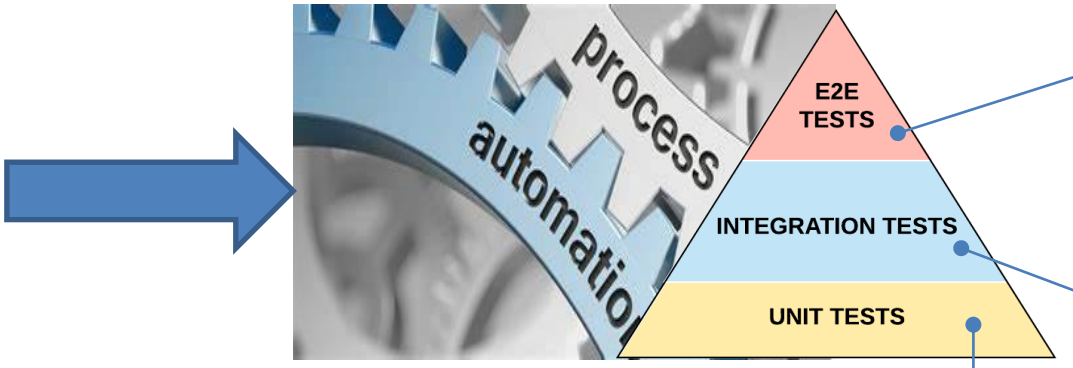
LES TESTS MANUELS



TEST & QUALITÉ LOGICIELLE

AUTOMATISATION DES TESTS

- Les tests manuels sont une tâche très exigeante :
  - Définition de tous les scénarios possibles d'utilisation du logiciel
  - Des tests fréquents (d'une manière répétitive) de ces scénarios
  - Généralement effectués à la fin de chaque release



Il s'agit d'un type de test garantissant qu'une fonction de niveau utilisateur fonctionne comme prévu

les tests d'intégration aident à découvrir les problèmes lorsque les unités de code sont assemblées pour créer des composants plus complexes **(Nigntly Test)**

Ce type de test se concentre sur le test d'une unité de code : un bloc de construction d'une application logicielle, telle qu'une fonction ou une classe

TEST & QUALITÉ LOGICIELLE

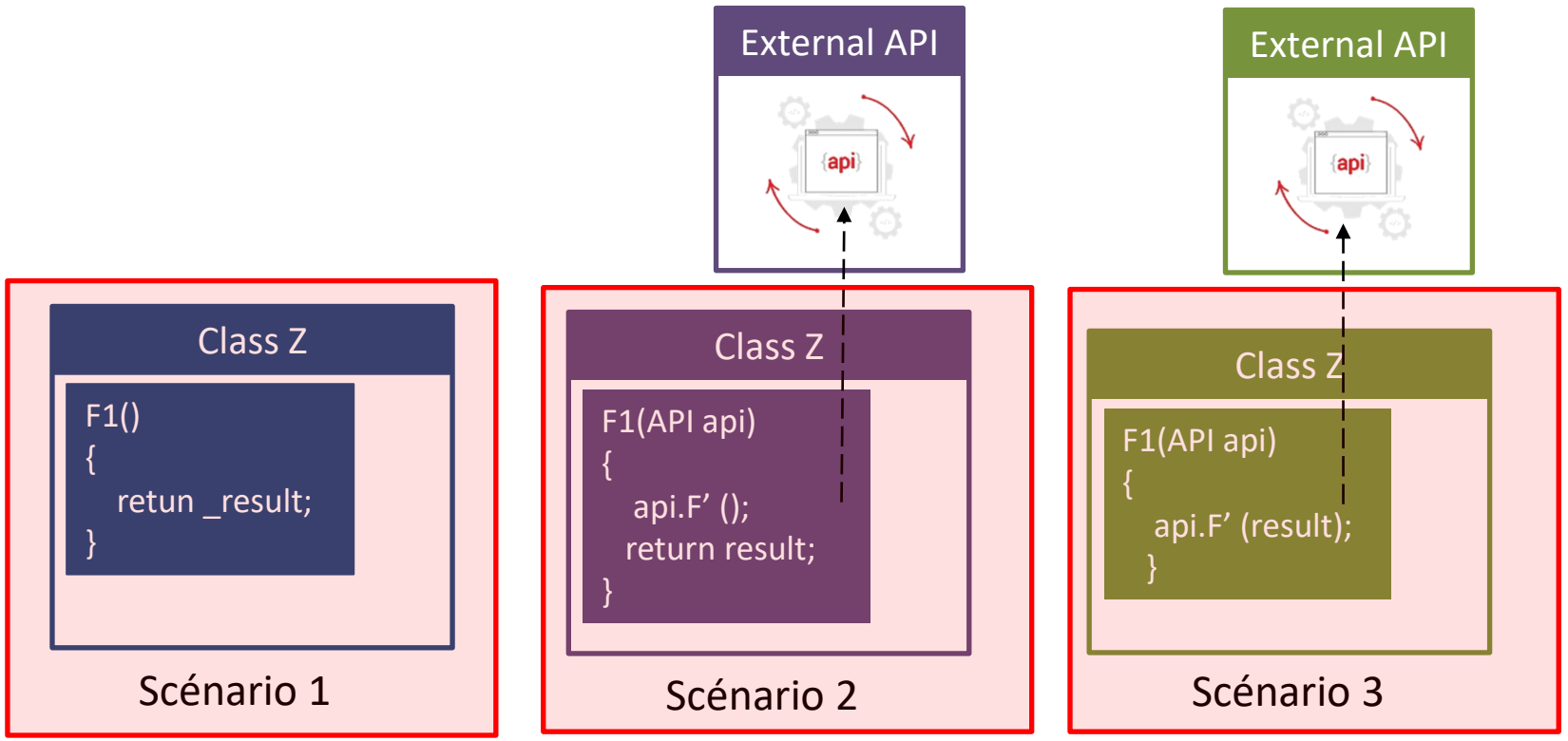
LES TESTS UNITAIRES

- Les avantages des tests unitaires :
  - Tester fréquemment, et rapidement votre code
  - Réécrire vos méthodes avec confiance (**Refectoring**)
  - Détecter les erreurs avant le déploiement
  - Eviter les problèmes de régression si un bout de code est modifié
  - Livrer votre produit final avec confiance

TEST & QUALITÉ LOGICIELLE

LES TESTS UNITAIRES

- Les tests unitaire testent les fonction d'une classe isolée de ses dépendances externes
  - **Trois scénarios de tests possibles**



TEST & QUALITÉ LOGICIELLE

LES TESTS UNITAIRES

Scénario I

```
Class Logger {
public bool IsValidLogFile(String fileName)
{
  if (fileName.EndsWith(".SLF"))
    return true;
  return false;
}
```

Logger.DLL | Logger.jar | **Logger.pyd**

Création d'un Projet de Test + FrameWork de test (Xunit, Junit, **PyUnit**)

- Une classe de test pour chaque classe métier
- Plusieurs méthodes de test pour chaque classe under test

[TextFixture]

```
Class LoggerTest {
[Test]
public bool IsValidLogFile_badExt_ReturnFalse()
{
    String fileWithbadExt = "file.bst";
    Logger logger = new Logger();
```

Nom de la fonction | scénario | Résultat attendu

Arrange

```
    bool resultat = logger.IsValidLogFile(fileWithbadExt);
```

Action

```
    Assert.False(resultat);
```

Assert

```
}
```

Les 3 A

- Faut-il ajouter un autre test?

TEST & QUALITÉ LOGICIELLE

LES TESTS UNITAIRES

- **Scénario II**
  - Nous allons supposer que nous disposons d'un fichier de configuration XML contenant toute les extensions valides des fichiers de config.
  - Nous disposons aussi d'un API FileExtention contenant une méthode qui vérifie IsValid a validité de l'extension du fichier

```
Class Logger {

StringService _stringService;

  public Logger ()
  {
    FileExtention _fileManager= new FileExtention();
  }

  public bool IsValidLogFile(String fileName)
  {
    return fileManager. IsValid(fileName);
  }
```

```
Class FileExtention {

  Bool IsValid (String fileName)
  {
    ...
  }
```

TEST & QUALITÉ LOGICIELLE

LES TESTS UNITAIRES

- **Scénario II**

```
Class Logger {

StringService _stringService;

  public Logger ()
  {
    FileExtention _fileManager= new FileExtention();
  }

  public bool IsValidLogFile(String fileName)
  {
    return fileManager. IsValid(fileName);
  }
```

```
Class FileExtention {

  Bool IsValid (String fileName)
  {
    ...
  }
```

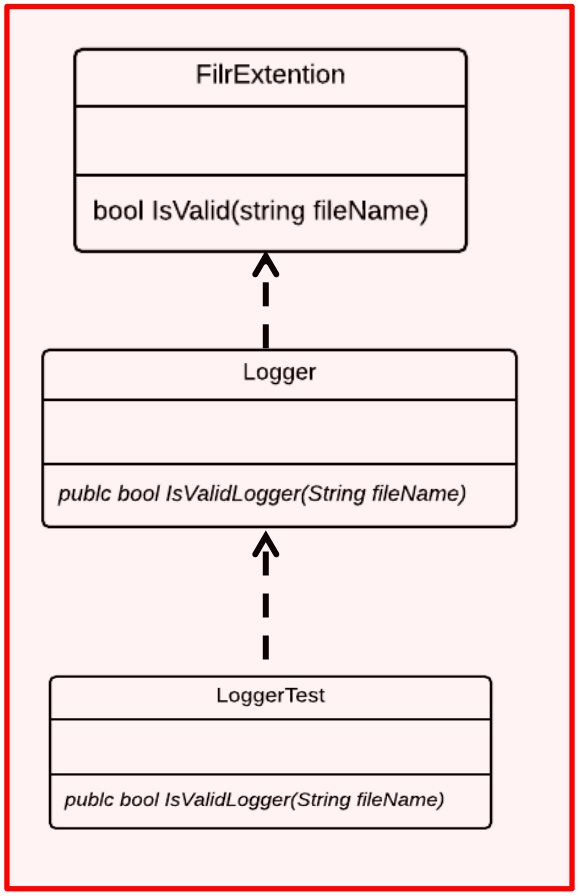
Il est claire que si on définit le test de la même manière que dans le cas du scénario 1, la fonction test sera dans l'obligation d'exécuter la méthode IsValid à l'intérieur de l'Action.

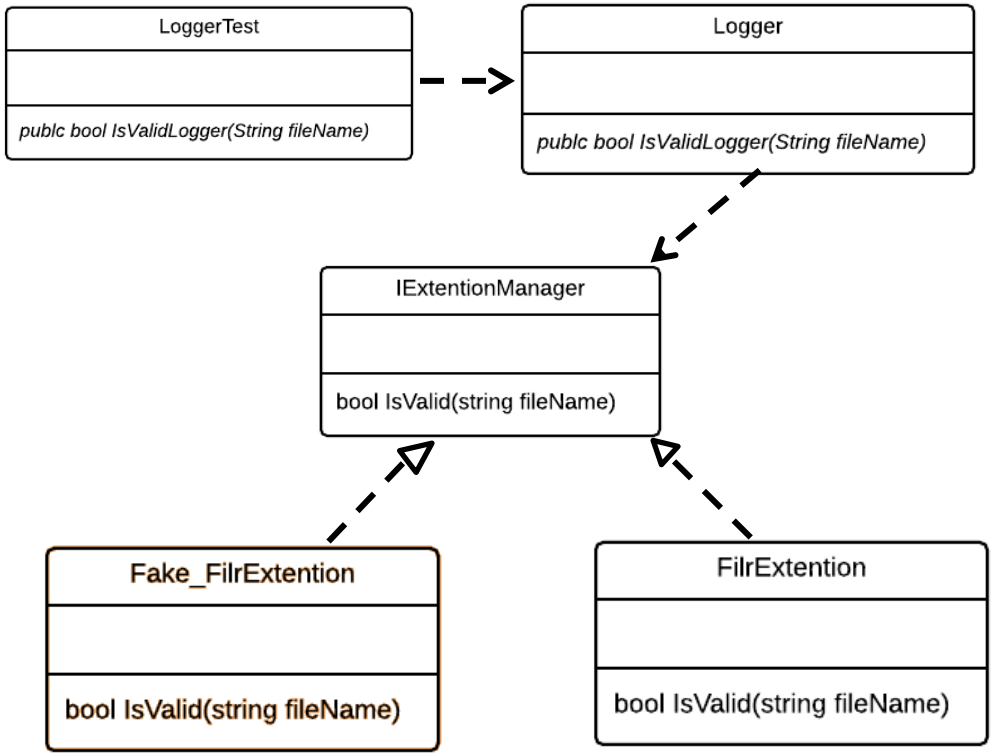
Code Refactoring

TEST & QUALITÉ LOGICIELLE

LES TESTS UNITAIRES

- **Scénario II**





TEST & QUALITÉ LOGICIELLE

LES TESTS UNITAIRES

- **Scénario II**
- **Exercice**

```
public class GererHeureSupPersonnel
{
  DAOPersonnel _dBPersonnel;
  public GererHeureSupPersonnel(DAOPersonnel dBEtudiant)
  {
    _dBPersonnel = dBEtudiant;
  }
  public double CalculerHeureSupPersonnels(Personnel p, String mois)
  {
    List<HeurSupDTO> heuresSups = _dBPersonnel.GetHeureSup(p, mois);
    double sommeHS = 0;
    foreach (HeurSupDTO heurSupDTO in heuresSups)
    {
      sommeHS += HeurSupDTO._dureeHeureSup;
    }
      return sommeHS;
}
```

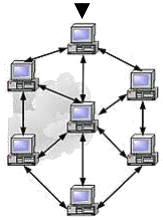
TEST & QUALITÉ LOGICIELLE

LES TESTS UNITAIRES

- **Scénario III**

```
Class Math {

  Email _email;

  public Math ()
  {
    Email _email= new Email();
  }

  public void Somme(int x, int y)
  {
    int s = x + y;
    _email.EnvoyerEmail(s)
  }
```

```
Class Email {

  Bool EnvoyerEmail (String message)
  {
    ...
  }
```



On veut tester si le résultat de la méthode *Somme* (vous pouvez bien sûr imaginez un traitement plus complexe pour calculer s) a été envoyé correctement ou pas.

LES TESTS UNITAIRES AVEC JUNIT

JUnit

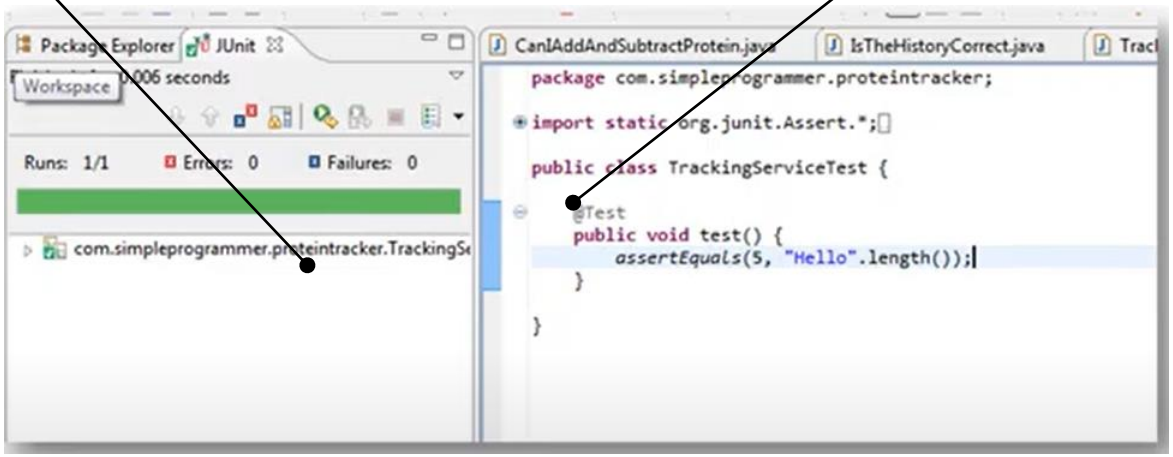
TEST & QUALITÉ LOGICIELLE

LES TESTS UNITAIRES AVEC JUNIT

- Junit est un framework de test créer spécifiquement pour java.

Junit Runner

JUnit annotation



- Chaque méthode de test est précédée par une annotation @Test
- D'autres annotations indiquent à JUnit Runner un ensemble d'informations sur la manière avec laquelle les tests doivent être exécutés

TEST & QUALITÉ LOGICIELLE

LES TESTS UNITAIRES AVEC JUNIT

JUnit

- Les Fonctionnalités de base de JUnit

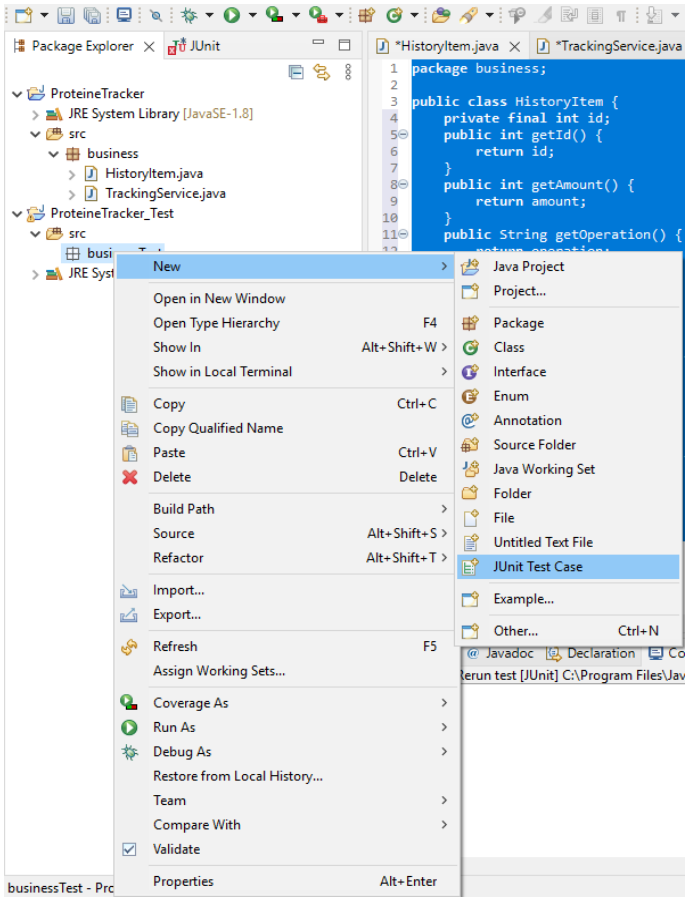
```
package business;
import java.util.ArrayList;
import java.util.List;

public class TrackingService {

private int total;
public int getTotal() {
return total;}

private int goal;
public void setGoal(int goal) {
this.goal = goal;}

private List<HistoryItem> items =new ArrayList<>();
private int historyId = 0;
public void addProteine(int amount)
{
total+=amount;
items.add(new HistoryItem(historyId++,amount,"add",total));
}
public void removeProteine(int amount)
{
total-=amount;
if (total<0) total=0;
items.add(new HistoryItem(historyId++,amount,"substract",total));

}
public boolean goalMet()
{
return total>=goal;
}
public List<HistoryItem> getItems() {
return items;
} }
```

```
package business;

public class HistoryItem {
private final int id;
public int getId() {
return id;
}
public int getAmount() {
return amount;
}
public String getOperation() {
return operation;
}
public int getTotal() {
return total;
}
private final int amount;
private final String operation;
private final int total;
public HistoryItem(int id, int amount, String operation, int total) {
super();
this.id = id;
this.amount = amount;
this.operation = operation;
this.total = total;
}

}
```

TEST & QUALITÉ LOGICIELLE

LES TESTS UNITAIRES AVEC JUNIT

- Les Fonctionnalités de base de JUnit



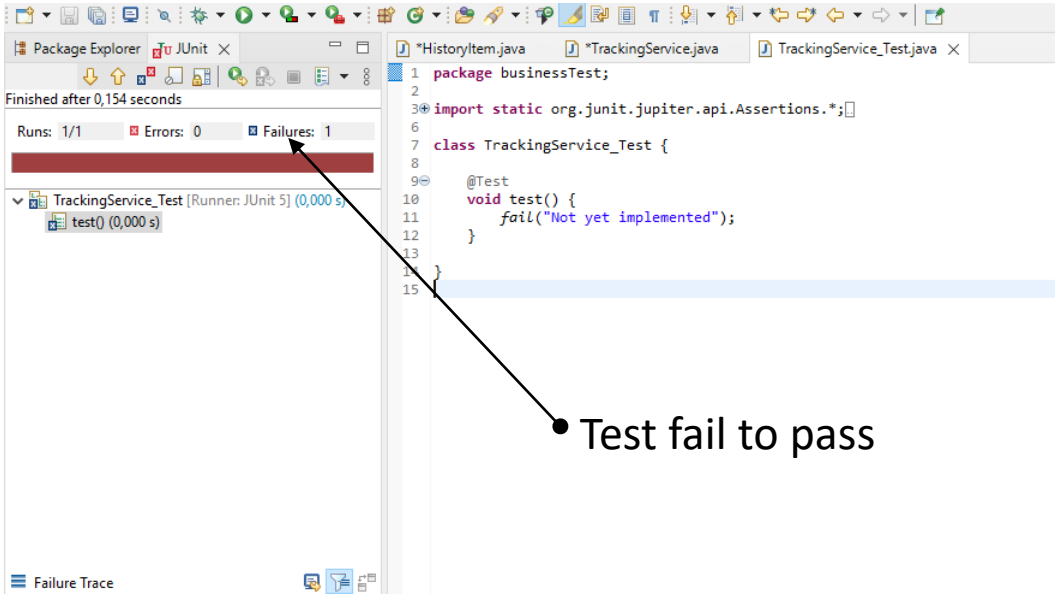
- Créer un nouveau projet java : ProteineTest_Test
- Créer un répertoire businessTest
- Ajouter un Junit test case : TrackingService_Test

```
package businessTest;

import static org.junit.jupiter.api.Assertions.*;

import org.junit.jupiter.api.Test;

class TrackingService_Test {

@Test
 void test() {
    fail("Not yet implemented");
  }
}
```

TEST & QUALITÉ LOGICIELLE

LES TESTS UNITAIRES AVEC JUNIT

- Les Fonctionnalités de base de JUnit

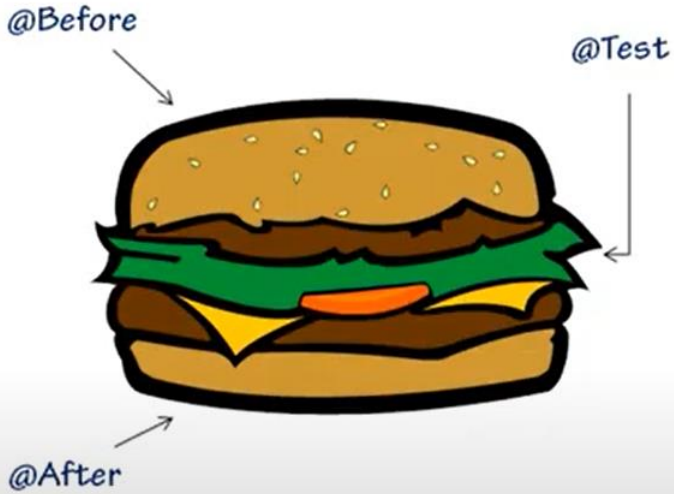


LES TESTS UNITAIRES AVEC JUNIT

- Les Fonctionnalités de base de JUnit

JUnit Annotations (Basic)

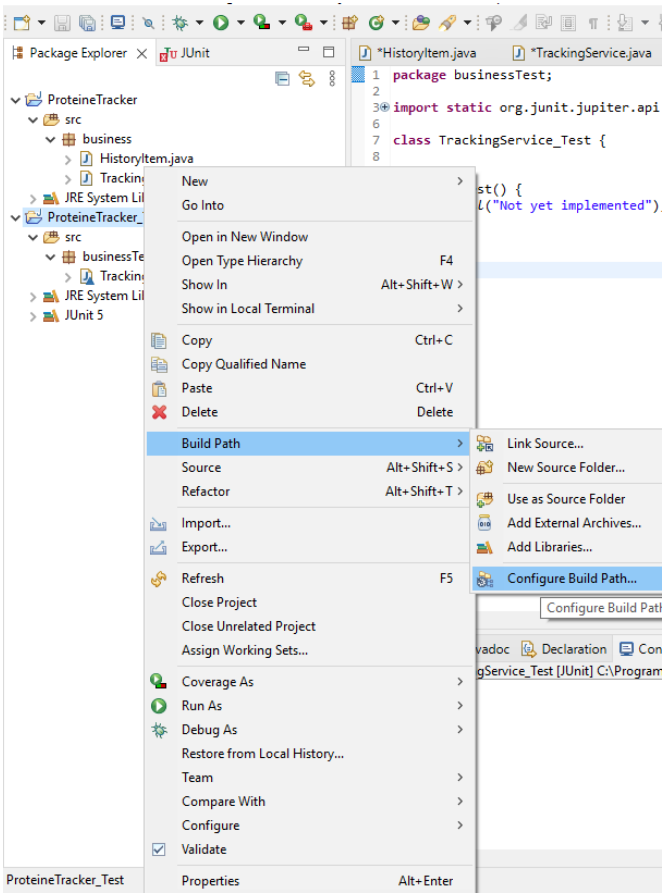
- @Test
- @Before
- @After
- @BeforeClass
- @AfterClass
- @Ignore
- @Test(expected = Exception.class)
- @Test(timeout = 100)

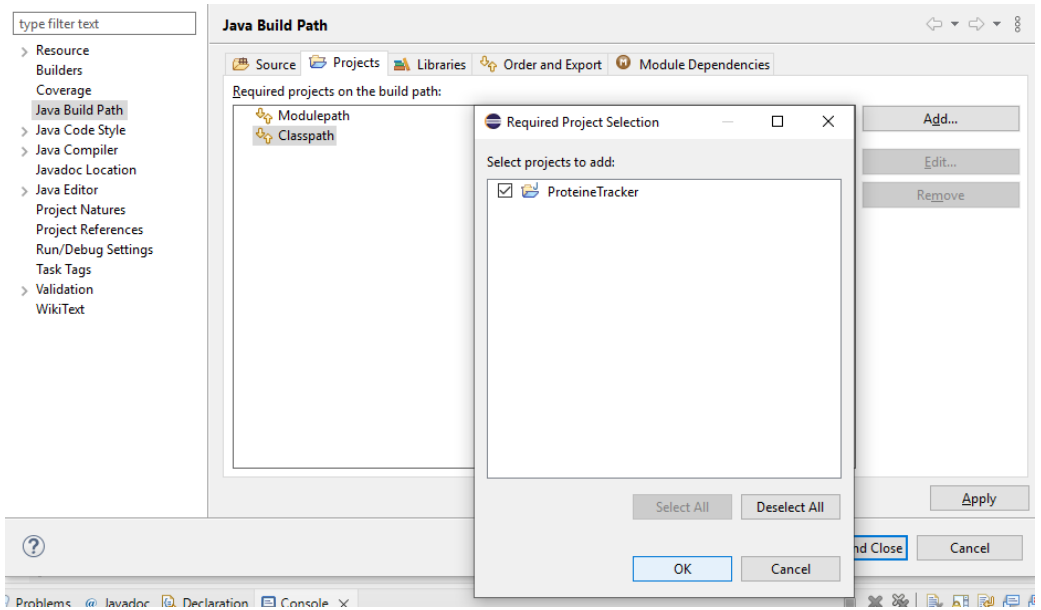


TEST & QUALITÉ LOGICIELLE

LES TESTS UNITAIRES AVEC JUNIT

- Les Fonctionnalités de base de JUnit





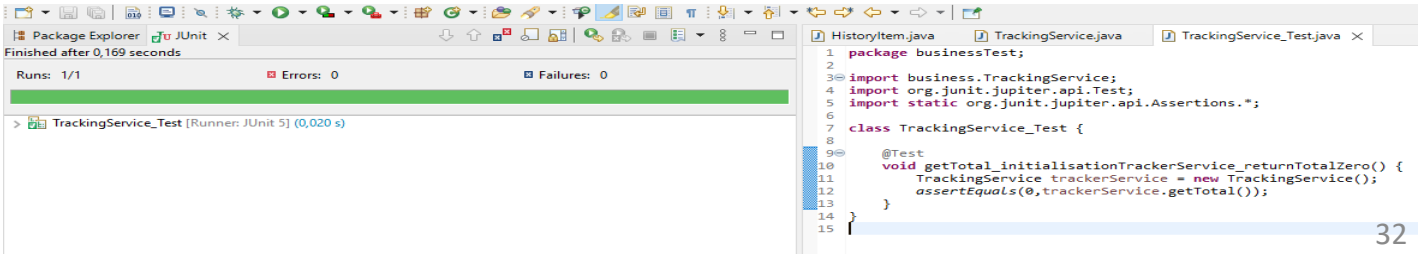
TEST & QUALITÉ LOGICIELLE

LES TESTS UNITAIRES AVEC JUNIT

- Les Fonctionnalités de base de JUnit

- Tester si le total est initialisé à zéro lorsque la classe TrackingService est créée.

```
package businessTest;

import business.TrackingService;
import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.*;

class TrackingService_Test {

@Test
void getTotal_initialisationTrackerService_TotalIsSetToZero() {
 TrackingService trackerService = new TrackingService();
 assertEquals(0,trackerService.getTotal());
}
}
```



TEST & QUALITÉ LOGICIELLE

LES TESTS UNITAIRES AVEC JUNIT

JUnit

- Les Fonctionnalités de base de JUnit

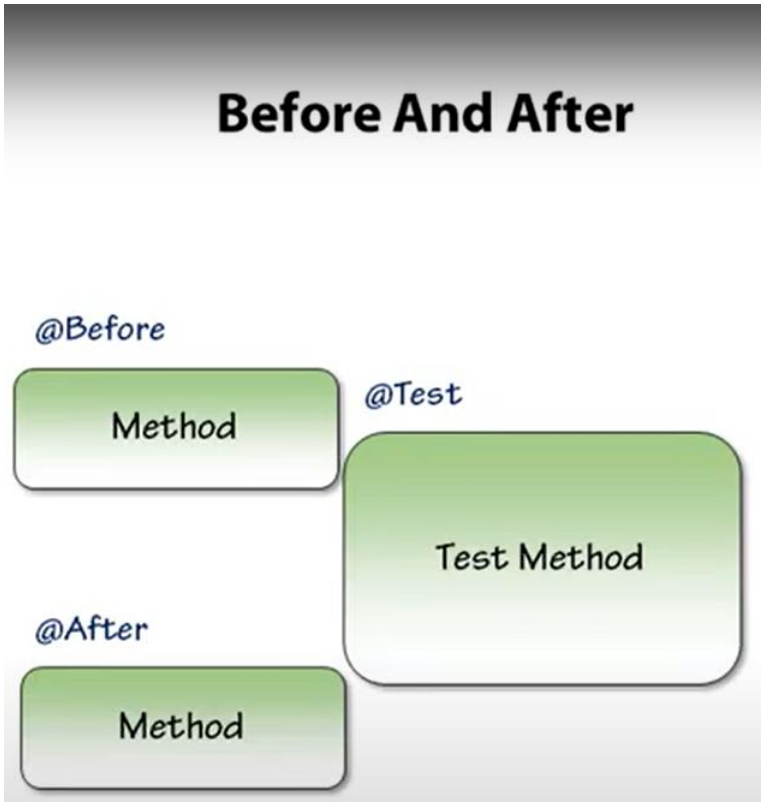
- Tester si le total augment avec la même quantité du protéine ajoutée
- Tester si le total reste égal à zéro si on diminue la protéine par une valeur > à zero

```
package businessTest;

import business.TrackingService;
import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.*;

class TrackingService_Test {
...
@Test
void getTotal_AddingAnAmountOfProteinToTotal_TotalIncreaseWithThatAmount() {
 TrackingService trackerService = new TrackingService();
 trackerService.addProteine(30);
 assertEquals(30,trackerService.getTotal());
}
@Test
void getTotal_RemovingAnAmountOfProteinToTotal_TotalStillZerot() {
 TrackingService trackerService = new TrackingService();
 trackerService.removeProteine(30);
 assertEquals(0,trackerService.getTotal());
}
```

TEST & QUALITÉ LOGICIELLE

LES TESTS UNITAIRES AVEC JUNIT

- Les Fonctionnalités de base de JUnit



- Before et After sont deux annotation très utiles dans le test unitaire assurant la suppression des répétition dans les tests

- Une méthode *public*, retournant un *void* et décorée par ***@Before***, respectivement ***@After***, sera exécutée avant, respectivement après, chaque exécution d'une méthode de test,

TEST & QUALITÉ LOGICIELLE

LES TESTS UNITAIRES AVEC JUNIT

- Les Fonctionnalités de base de JUnit

```
class TrackingService_Test {

	private TrackingService trackerService;
	@BeforeEach
	public void setUp()
	{
	System.out.println("Before");
	trackerService = new TrackingService();}

	@Test
	void getTotal_initialisationTrackerService_TotalIsSetToZero() {
	assertEquals(0,trackerService.getTotal());
	}
	@Test
	void getTotal_AddingAnAmountOfProteinToTotal_TotalIncreaseWithThatAmount() {
	 trackerService.addProteine(30);
	 assertEquals(30,trackerService.getTotal());
	}
	@Test
	void getTotal_RemovingAnAmountOfProteinToTotal_TotalStillZerot() {
	 trackerService.removeProteine(30);
	 assertEquals(0,trackerService.getTotal());
	}}
```

TEST & QUALITÉ LOGICIELLE

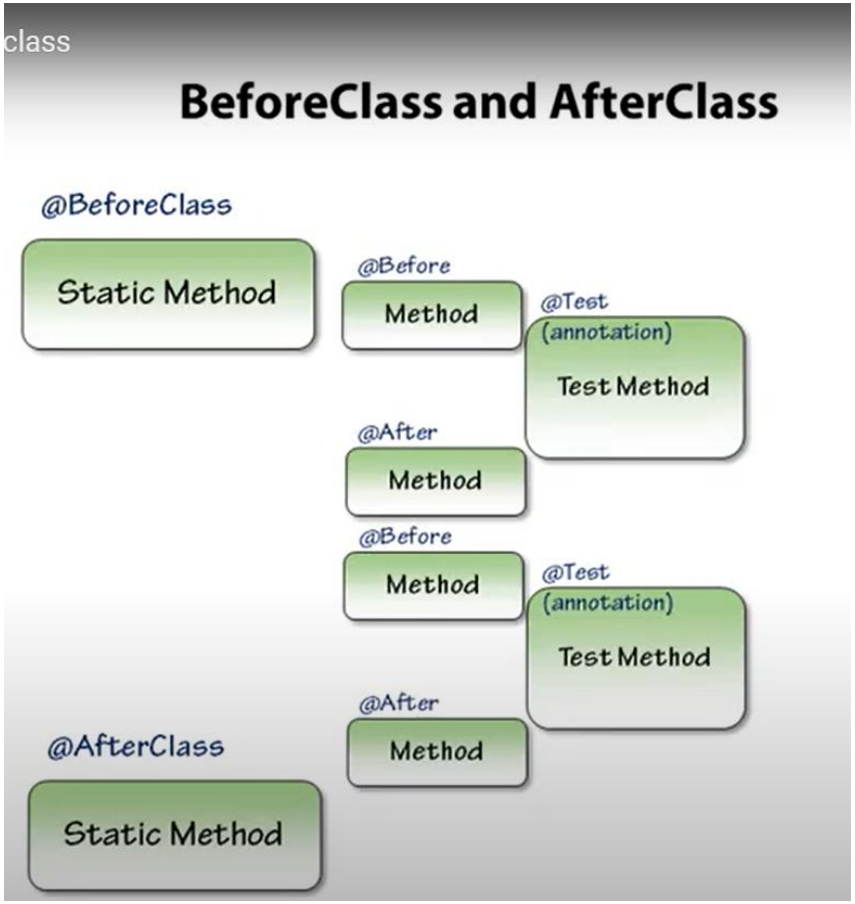
LES TESTS UNITAIRES AVEC JUNIT

- Les Fonctionnalités de base de JUnit

```
class TrackingService_Test {

	private TrackingService trackerService;
	@BeforeEach
	public void setUp()
	{
	System.out.println("Before");
	trackerService = new TrackingService();
	}

	@AfterEach
	public void tearDown()
	{
	System.out.println("After");
	}
	...
	@Test
	void getTotal_RemovingAnAmountOfProteinToTotal_TotalStillZerot() {
	 trackerService.removeProteine(30);
	 assertEquals(0,trackerService.getTotal());
	}}
```

LES TESTS UNITAIRES AVEC JUNIT

- Les Fonctionnalités de base de JUnit

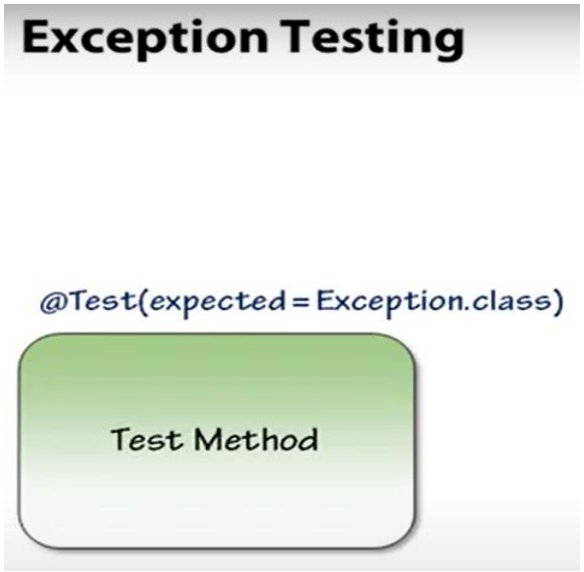


- Contrairement à *@Before* et*@After*, les méthodes annotées par **@BeforeAll,** respectivement **@AfterAll,** s'exécutent une fois seulement au début, respectivement à la fin des tests.
- Les méthode annotées par *@BeforeAll* et *@AfterAll* doivent être ***statiques***.
- Très utiles dans **les tests d'intégration**

TEST & QUALITÉ LOGICIELLE

LES TESTS UNITAIRES AVEC JUNIT

- Les Fonctionnalités de base de JUnit



- Junit a développé et supporté des techniques nous permettant d'écrire des tests qui vérifient si une exception est générée par une méthode ou pas.

```
public void setGoal(int goal) throws
InvalidGoalException {
if (goal<=0)
throw new InvalidGoalException();
this.goal = goal;
}
```

```
@Test
public void
setGoal_GoalLessThenZero_throwInvalidGoalException(
)
{
Assertions.assertThrows(InvalidGoalException.class,
()->trackerService.setGoal(0));
}
```

LES TESTS UNITAIRES AVEC JUNIT

- Les Fonctionnalités de base de JUnit

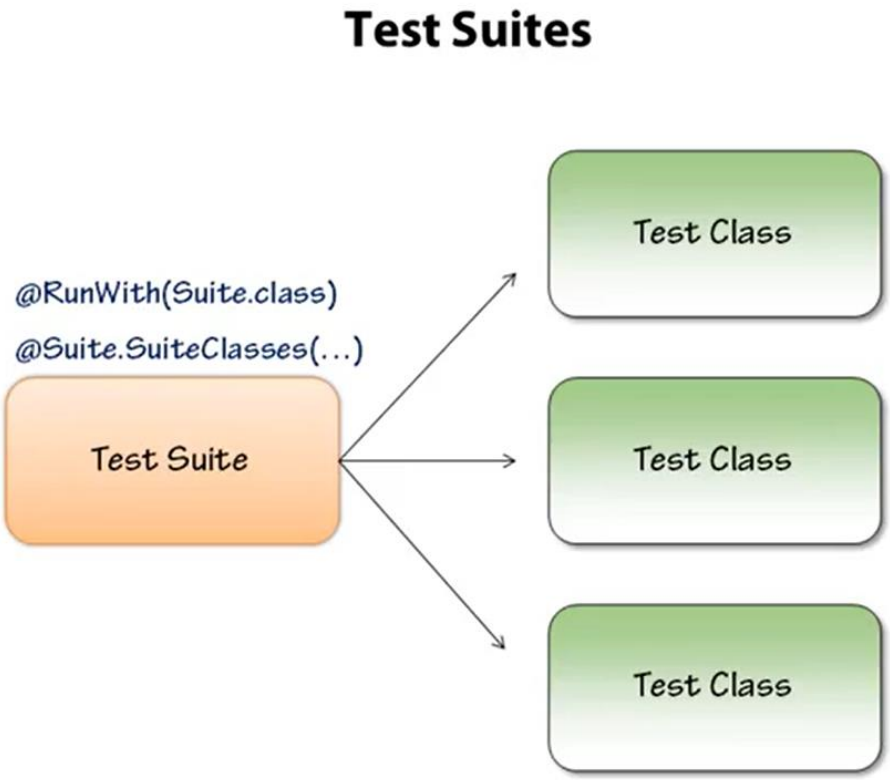
Assertions (Basic)

- assertArrayEquals
- assertEquals
- assertTrue
- assertFalse
- assertNull
- assertNotNull
- assertSame
- assertNotSame
- fail

TEST & QUALITÉ LOGICIELLE

LES TESTS UNITAIRES AVEC JUNIT

- Les Fonctionnalités de base de JUnit



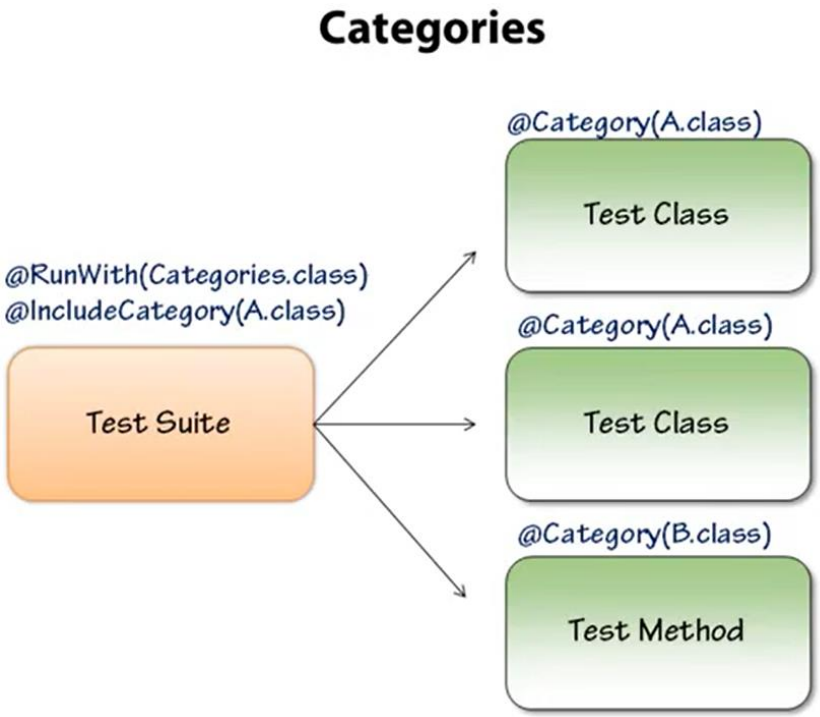
- **@RunWith(Suite.class),@Suite.SuiteCasses(listeOfClasses)** permettent d'exécuter plusieurs test class simultanément,

```
@RunWith(JUnitPlatform.class)

@SelectClasses(DummyTest.class)

@SelectPackages("businessTest")

public class ProteineTraking_Suite {

}
```

TEST & QUALITÉ LOGICIELLE

LES TESTS UNITAIRES AVEC JUNIT

- Les Fonctionnalités de base de JUnit



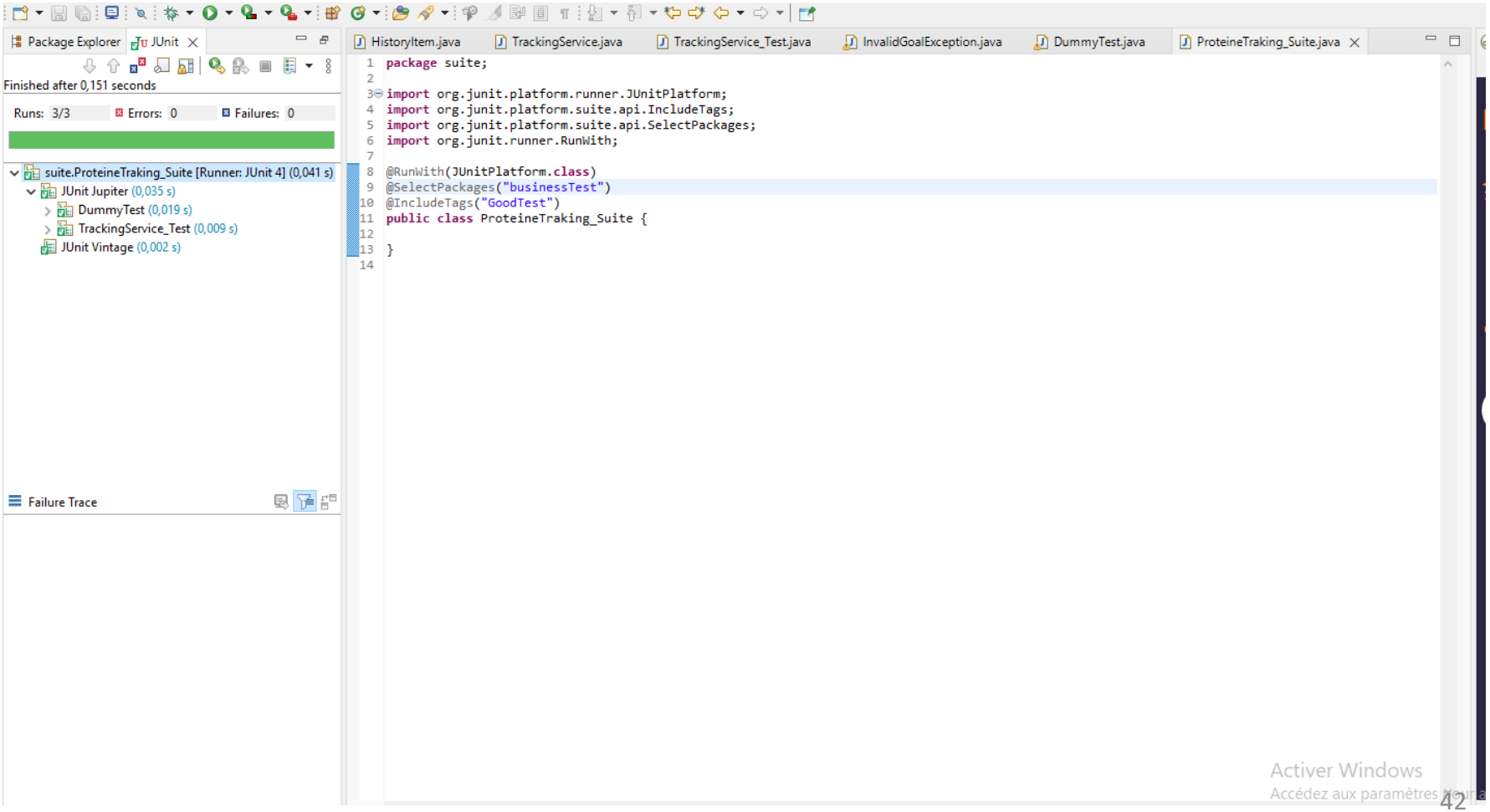
- **@Tag** permet de tagger une méthode. Cette dernière sera testée si son tag est spécifé dans **JuinitPlatForm (Suite JUnit4)**
- **IncludeTags("tag")** permet au Runner d'exécuter les classes marquée par le Tag "tag".

```
RunWith(JUnitPlatform.class)
@SelectPackages("com.businessTest")
@IncludeTags("GoodTest")
public class ProteineTraking_Suite {}
```

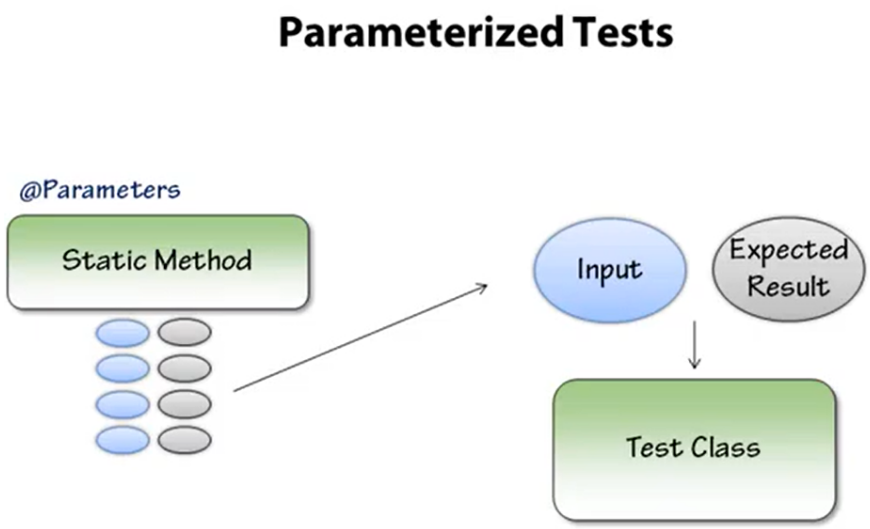
```
public class DummyTest {
@Test
@Tag("GoodTest")
void test() {}
}
```

TEST & QUALITÉ LOGICIELLE

LES TESTS UNITAIRES AVEC JUNIT



LES TESTS UNITAIRES AVEC JUNIT



- Permet d'éviter l'écriture du même test pour plusieurs valeurs différentes

- Pour ***Juinit 5*** :
- Créer une fonction *static* qui retourne une liste d'argument ***(Stream<Arguments>)*** à utiliser par toute fonction de test recommandant ce flux à travers les annotations

@ParameterizedTest
@MethodSource("data")

TEST & QUALITÉ LOGICIELLE

LES TESTS UNITAIRES AVEC JUNIT

```
public class ParamaterizedTests {

  private static TrackingService trackingService=new TrackingService();

   public static Stream<Arguments> data()
      {
            return Stream.of(
            Arguments.arguments(5,5),
            Arguments.arguments(5,10),
            Arguments.arguments(-12,0),
            Arguments.arguments(50,50)
           );}
   @ParameterizedTest
   @MethodSource("data")
  void  getTotal_AddingAnAmountOfProteinToTotal_TotalIncreaseWithThatAmount(int input,int expected)
  {
        if (input>0)
        trackingService.addProteine(input);
        else
        trackingService.removeProteine(-input);
        assertEquals(expected,trackingService.getTotal());
  }
}
```

TEST & QUALITÉ LOGICIELLE

LES TESTS UNITAIRES AVEC JUNIT

- Integrating Junit : Run Junit outside IDE

- **Junit** est un Framework performant pour créer rapidement et efficacement des tests unitaires.
- **Juint Runner** buit-in eclipse est idéale pour exécuter des tests en mode développement, mais cela ne présente pas la meilleure option pour automatiser des tests,
- L'ultime objectif de JUnit est de pouvoir automatiser les tests dans un building process, tel que Ant ou Maven.

TEST & QUALITÉ LOGICIELLE

LES TESTS UNITAIRES AVEC JUNIT

Integrating Junit : Maven

- Créer un projet Maven
- Le projet est scindé en deux parties
  - Src/main/java : contenant le code source de votre projet
  - Src/test/java :contenant les tests relatifs au code source. Par convention toutes les classes tests doivent se terminées par « TEST »
- *Ajouter les dépendances nécessaires au fichier pom.xml. M**aven** se préoccupera du téléchargement des dépendances. Installer chromeDriver compatible avec la version du chrome.*
- *Maven peut être utilisé pour l'intégration continue avec Jenkins*

```
<dependency>
      <groupId>org.junit.jupiter</groupId>
      <artifactId>junit-jupiter-engine</artifactId>
      <version>${junit.jupiter.version}</version>
      <scope>test</scope>
   </dependency>
<dependency>
      <groupId>org.junit.jupiter</groupId>
      <artifactId>junit-jupiter-api</artifactId>
      <version>${junit.jupiter.version}</version>
      <scope>test</scope>
</dependency>
```

```
<properties>
   <maven.compiler.source>11</maven.compiler.source>
   <maven.compiler.target>11</maven.compiler.target>
   <junit.jupiter.version>5.4.0</junit.jupiter.version>
</properties>
```

TEST & QUALITÉ LOGICIELLE

LES TESTS UNITAIRES AVEC Selenium

- Selenium est outils open source puissant permettant d'automatiser les navigateurs web.
- Selenium peut être développée facilement avec plusieurs outils y compris Junit.

- Au niveau du votre projet ***maven***, ajouter une nouvelle classe SeleniumTest dans le package

```
class SeleniumTest {
  @Test
   void testSelenium() {
      System.setProperty("webdriver.chrome.driver", "g:\\chromedriver.exe");
      WebDriver driver=new ChromeDriver();
      driver.get("http://www.google.com");
      WebElement element = driver.findElement(By.name("q"));
      element.sendKeys("Tunisia!\n");
      element.submit();
    }
}
```

TEST & QUALITÉ LOGICIELLE

LES TESTS UNITAIRES AVEC Selenium

- Selenium est outils open source puissant permettant d'automatiser les navigateurs web.
- Selenium peut être développée facilement avec plusieurs outils y compris Junit.

- Au niveau du votre projet ***maven***, ajouter une nouvelle classe SeleniumTest2 dans le package

```
public class ChromeSeleniumTest {
 private static WebDriver driver;

@BeforeAll
  public static void setUp(){
     System.setProperty("webdriver.chrome.driver", "/Chemin du chrome driver");
     driver = new ChromeDriver(); }
  @Test
   public void testChromeSelenium() {
      driver.get("https://memorynotfound.com/"); }
  @AfterAll
    public static void cleanUp(){
        if (driver != null) {
                driver.close();
                driver.quit();
         }
} }
```

TEST & QUALITÉ LOGICIELLE

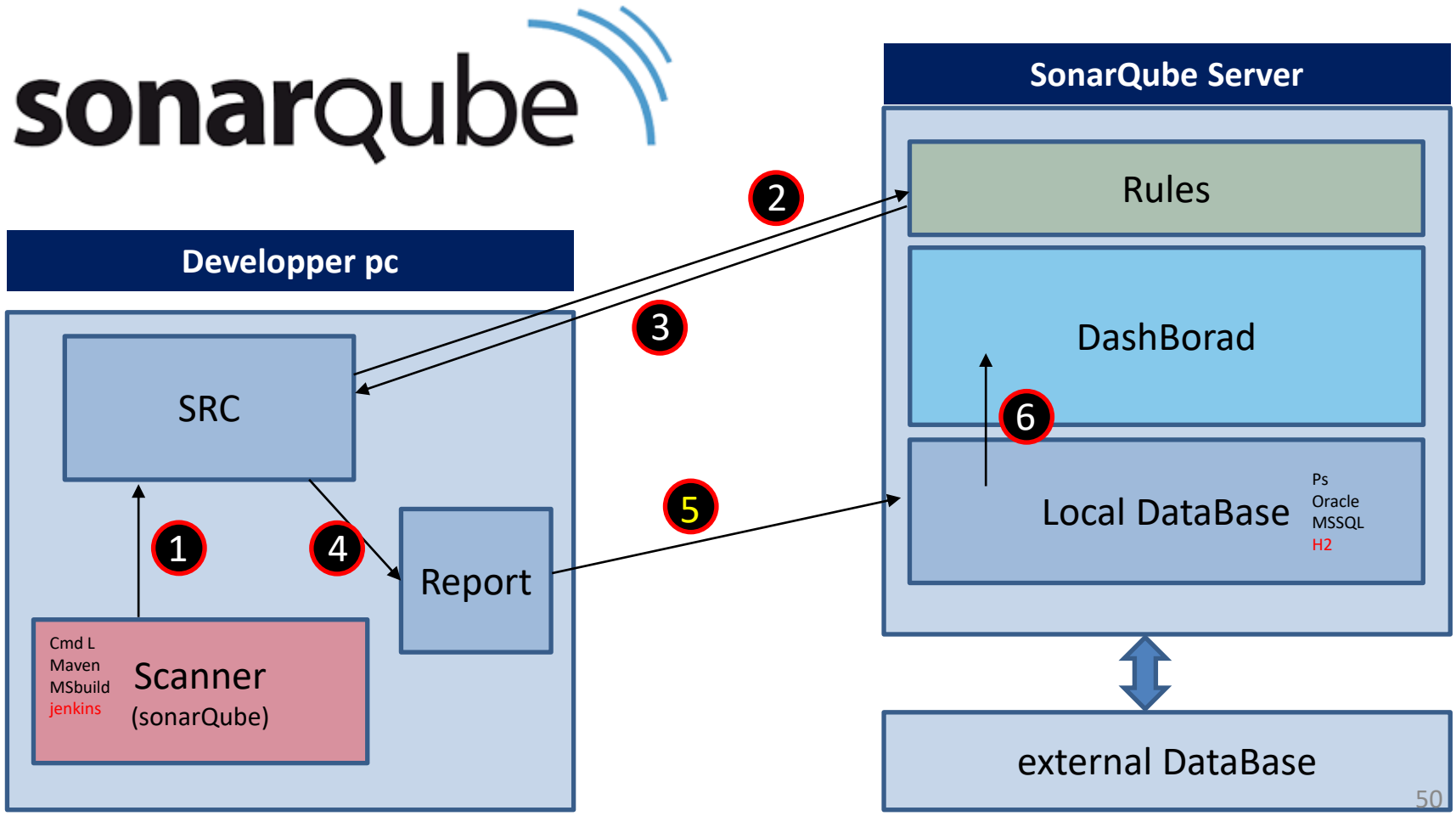
LES TESTS STATIQUES

- SonarQube est une plateforme open source pour le contrôle et l'inspection de la qualité du code source.
- Développé avec un ultime objectif de rendre la gestion de la qualité du code source accessible par toute l'équipe de développement

- SonarQube assure l'automatisation des révisons du code (Test Statique du code)
-

TEST & QUALITÉ LOGICIELLE

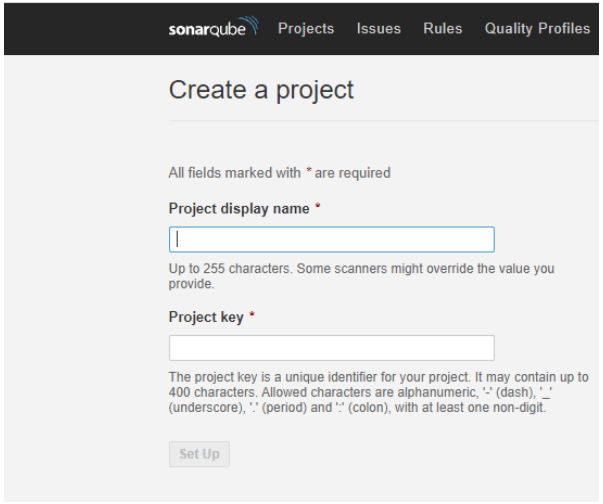
LES TESTS STATIQUES

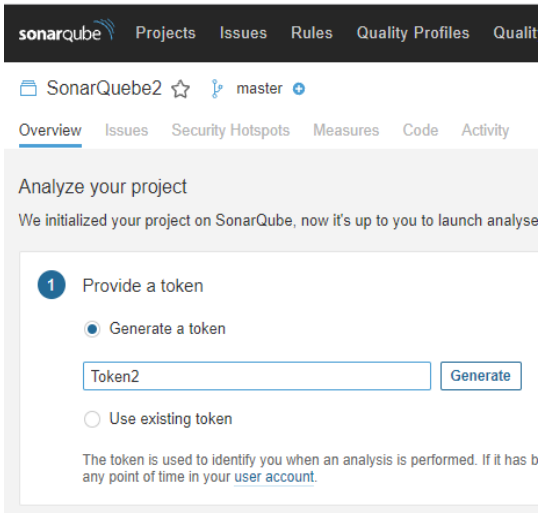


TEST & QUALITÉ LOGICIELLE

LES TESTS STATIQUES

- Installer la dernière version de SonarQube
- Lancer la command StartSonar.bat pour lancer le serveur
- Lancer le Danshbord du serveur : localhost:9000
- Pour votre première connexion, réinitialiser votre username et pw
- Créer un nouveau projet Maven et développer votre premier test
- Créer un nouveau projet dans le Dashbord de sonarQue pourtant le même nom
- Créer votre token d'identification, et copier la commande Maven à exécuter dans le répertoire de votre projet






```
mvn clean verify sonar:sonar

-Dsonar.projectKey=SonarQuebe2

-Dsonar.host.url=http://localhost:9000

-Dsonar.login=
8ae2d4824e91d7b428dbafafb047f20521e35626
```