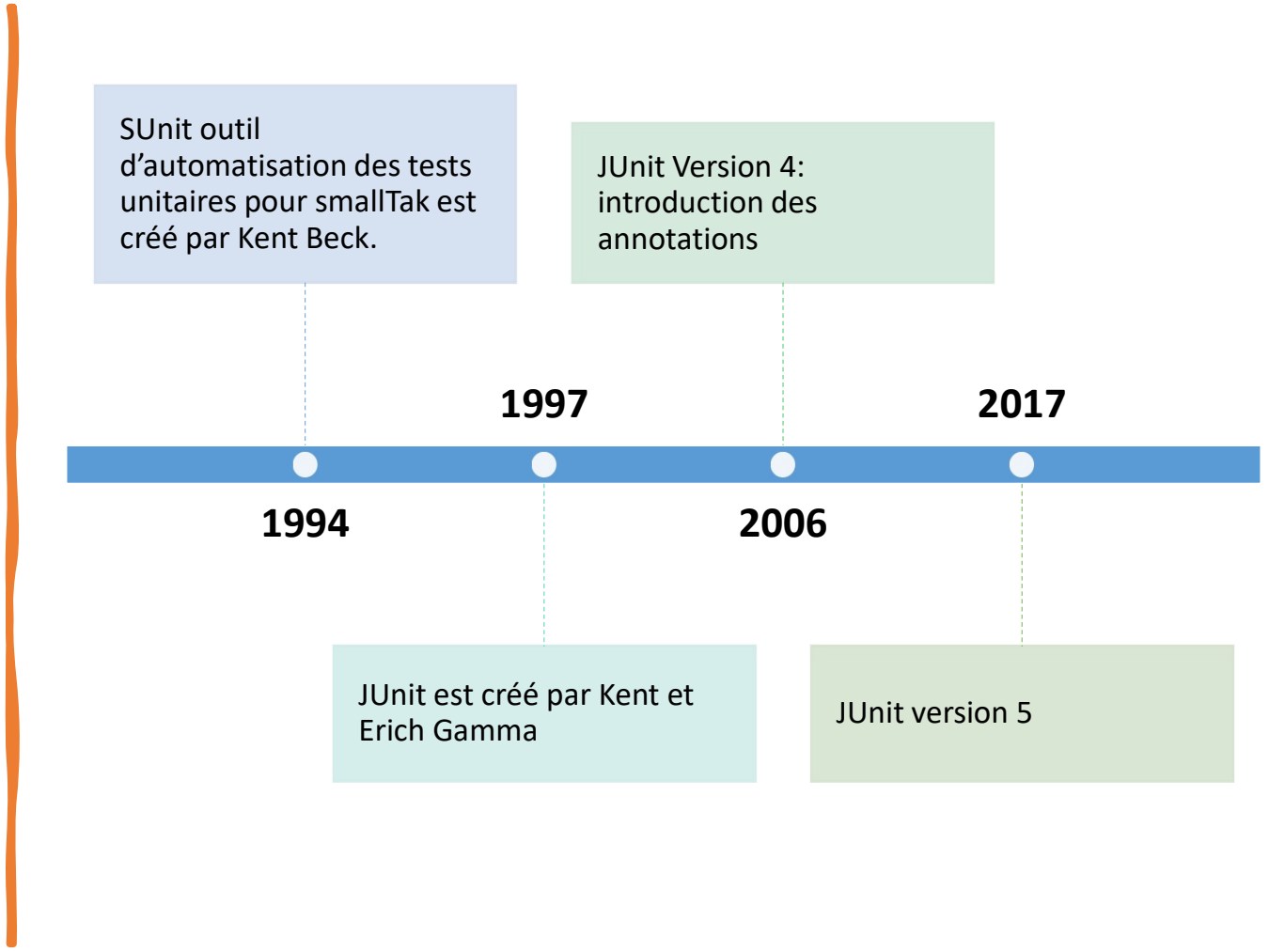
JUnit

Tests unitaires

Types de tests

- Test unitaires
- Tests d’intégration
- Tests d’acceptation (du point du vue utilisateur)
- Styles de tests
  - Boîte noire: Tester le comportement
  - Boîte blanche

Historique


SUnit outil d'automatisation des tests unitaires pour smallTak est créé par Kent Beck.
JUnit Version 4: introduction des annotations
1997
2017
1994
2006
JUnit est créé par Kent et Erich Gamma
JUnit version 5

Tests

- Tests unitaires
  - Xunit: SUnit, Nunit CppUnit, Eunit , PerlUnit, PHPUNit, xUnit.net, XUnit.jsMocha , AVA, Jasmine, Tape, minitest, FSTest,pytest
- Tests de l’interface utilisateur: Selenium, SOATest.
- Tests système: Simian Army (ensemble d’outils de tests des systèmes dans le cloud, réalisé par Netflix), pour vérifier la disponibilité , la fiabilité, la sécurité ,

Etapes d’exécution d’un test

Initialisation (arrange): préparation des données et de l’environnement nécessaire à l’exécution du test.

Exercice (Act): exécution de la fonction à tester

Vérification (Assert): vérification du succès ou de l’échec de l’exécution

Désactivation (Annihilation): destruction éventuelle des objets temporaires ou libération des ressources utilisés lors du test.

Assertions

Une assertion est une condition booléenne qui entraîne soit l'échec du test soit son succès.

Méthodes d'assertion

- * Version 5 uniquement

assertEquals	assertNotEquals
assertFalse	assertTrue
assertNull	assertNotNull
assertSame	assertNotSame
assertAll	fail
assertThrows	assertTimeout
assertTimeoutPreemtively	assertThat (v 4 uniquement)

Junit 5

- Composé de 3 modules séparés:
  - Junit Platform: API d'exécution des tests
  - Junit Jupiter: API d'écriture des tests
  - Junit vintage: supporte l'exécution des tests écrits dans les versions 3 et 4.
- Supporte Java 8, notamment les expressions Lambda

Différences entre Junit 4 et Junit 5: annotations

Junit 5	Junit 4
@BeforeAll	@BeforeClass
@AfterAll	@AfterClass
@BeforeEach	**@Before**
@AfterEach	**@After**
@Disable	**@Ignore**

Différences entre Junit 4 et Junit 5: assumptions (hypothèses)

Junit 5	Junit 4
assumeFalse	assumeFalse
	assumeNoException
	assumeNotNull
assumeThat	assumeThat
assumeTrue	assumeTrue

Exemple JUnit 5: classe à tester

```
package cours.test;

public class Calcul implements ICalcul {

public int somme(int a, int b) {
return a + b;
}

public double division(double a, double b) {

return a / b;
}

public double division(int a, int b) {
return a / b;
}

public int somme(int n) {
int s = 0;
for (int i = 0; i < n;) {
s += i;

}
return s;

}

}
```

Exemple JUnit 5: classe de test

```java
package cours.test;

import static org.junit.Assert.assertNotEquals;
import static org.junit.jupiter.api.Assertions.assertNotNull;
import static org.junit.jupiter.api.Assertions.assertAll;
import static org.junit.jupiter.api.Assertions.assertTimeoutPreemptively;
import static org.junit.jupiter.api.Assertions.assertEquals;
import static org.junit.jupiter.api.Assumptions.assumeTrue;

import java.time.Duration;

import org.junit.jupiter.api.AfterAll;
import org.junit.jupiter.api.AfterEach;
import org.junit.jupiter.api.BeforeAll;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Disabled;
import org.junit.jupiter.api.DisplayName;
import org.junit.jupiter.api.RepeatedTest;
import org.junit.jupiter.api.Test;
import org.junit.jupiter.api.TestInstance;
import org.junit.jupiter.api.TestInstance.Lifecycle;

@TestInstance(Lifecycle.PER_METHOD) // Par défaut.
@DisplayName("Tests de la classe Calcul")
public class CalculTest {

    public CalculTest() {
        System.out.println("Constructeur");
    }

    // doit être static si Lifecycle.PER_METHOD
    @BeforeAll
    static void setUpAll() {
        System.out.println("Before All");
    }

    @BeforeEach
    void setUp() {
        System.out.println("Before Each");
    }

    // doit être static si Lifecycle.PER_METHOD
    static @AfterAll void tearDownAll() {
        System.out.println("After All");
    }

    @AfterEach
    void tearDown() {
        System.out.println("After Each");
    }

    @Test
    @DisplayName("Somme")
    @Disabled
    void sommeTest() {
        // Test fixture: transient Fresh, Persistence Fresh,Persistence Shared
        Calcul c = new Calcul();
        // Actions
        int d = c.somme(40, 60);

        // assertEquals(2,d,"la fonction somme est incorrecte");
        assertEquals(2, d, () -> {
            return "la fonction somme est incorrecte";
        });
    }

    @DisplayName("Division")
    @RepeatedTest(value = 5, name = "{displayName} -> {currentRepetition}/{totalRepetitions}")
    void divisionTest() {
        Calcul c = new Calcul();

        double d = c.division(20l, 1l);
        assertEquals(Double.POSITIVE_INFINITY, d);

    }

    Calcul calcul;
```

```java
    Calcul calcul;
    // AssertAll
    @Test
    void sommeTestAssertAll1() {
        calcul = new Calcul();
        int d = calcul.somme(40, 60);

        // Si un test échoue, les autres ne sont pas exécutés
        assertNotNull(calcul);
        assertNotEquals(100, d);
        assertEquals(20, d, () -> {
            return "la fonction somme est incorrecte";
        });

    }

    @Test

    void sommeTestAssertAll2() {
        calcul = new Calcul();
        int d = calcul.somme(40, 60);

        assertAll("Erreurs de caclul de la somme", () -> assertNotNull(calcul), () -> assertNotEquals(100, d),
                () -> assertEquals(20, d, () -> {
                    return "la fonction somme est incorrecte";
                }));

        ;
    }
    // assertTimeOut
    @Test()
    void sommeTimeout() {
        calcul = new Calcul();
        int n = 1000000;
        assumeTrue(1 == 2);// le reste du code ne sera pas exécuté.
        int s = assertTimeoutPreemptively(Duration.ofMillis(4), () -> calcul.somme(n));
        System.out.println("Somme=" + s);
    }
}
```

Tests unitaires dans un projet Spring Boot

La dépendance boot suivante:

```
<dependency>
<groupId>org.springframework.boot</groupId>
<artifactId>spring-boot-starter-test</artifactId>
<scope>test</scope>
</dependency>
```

ajoute dans le projet les bibliothèques: junit,hamcrest et mockito.

Test unitaire pour IntervenantService

```java
package cours.spring.mvc.conf.services;

import static org.hamcrest.CoreMatchers.is;
import static org.hamcrest.MatcherAssert.assertThat;
import static org.mockito.Mockito.when;
import java.util.ArrayList;
import java.util.List;
import org.junit.Test;
import org.junit.runner.RunWith;
import org.mockito.InjectMocks;
import org.mockito.Mock;
import org.mockito.runners.MockitoJUnitRunner;

import cours.spring.mvc.conf.model.Intervenant;
import cours.spring.mvc.conf.repository.IntervenantRepository;

@RunWith(MockitoJUnitRunner.class)
public class IntervenantServiceTest {

    @Mock
    IntervenantRepository intervenantRepository;

    @InjectMocks
    IntervenantService service = new IntervenantServiceImpl();
```

```java
@Test
public void testerIntervenant() {

    // Given When Then -> BDD Behaviour Driven development
    Intervenant i = new Intervenant();
    i.setId(1);
    i.setNom("Inter1");
    when(intervenantRepository.findOne(1l)).thenReturn(i);
    assertThat(service.getIntervenant(1l).getNom(), is("Inter1"));

}

@Test
public void testerIntervenants() {
    List<Intervenant> liste = new ArrayList<>();
    liste.add(new Intervenant());
    liste.add(new Intervenant());
    liste.add(new Intervenant());
    liste.add(new Intervenant());
    liste.add(new Intervenant());
    when(intervenantRepository.findAll()).thenReturn(liste);
    assertThat(service.getIntervenants().size(), is(5));
}
```

Test d'intégration pour IntervenantService

```java
package cours.spring.mvc.conf.servicesintegration;

import static org.junit.Assert.assertEquals;

@RunWith(SpringRunner.class)
@SpringBootTest
public class IntervenantServiceIntegrationTest {

    @Autowired
    IntervenantService intervenantService;

    @Test
    public void testerIntervenant() {
        Intervenant i = intervenantService.getIntervenant(1);

        assertEquals(1, i.getId());
    }
}
```