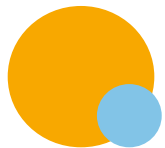


# Test Driven Development: ¡tu código a prueba de errores!

Miguel Martínez | VP Engineering





# ¿Quién soy?

- VP Engineering & Tech Lead en Next Digital
- Sectores: aerolíneas, logística, telecomunicaciones, ...
- Arquitectura de microservicios, APIs, automatización, development experience, buenas prácticas...



✉ [mmartinez@nextdigital.es](mailto:mmartinez@nextdigital.es)

✂ [@miguelms\\_es](https://twitter.com/miguelms_es)

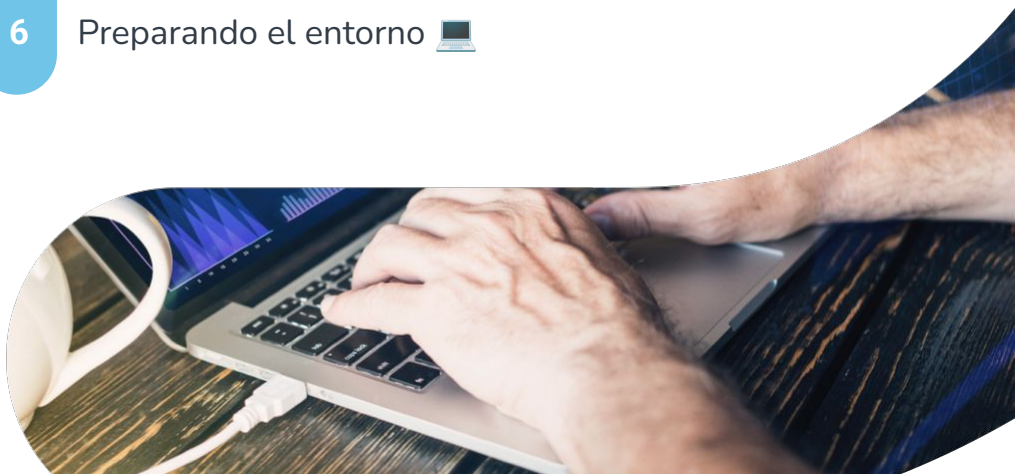
👤 [LinkedIn: Miguel Martínez Serrano](#)

💻 [miguelms.es](https://miguelms.es)

# Índice

## Parte 1

- 1 ¿Por qué testear?
- 2 Midiendo la calidad de código
- 3 ¿Cómo debe ser un test?
- 4 ¿Qué es TDD?
- 5 JUnit 5: conceptos
- 6 Preparando el entorno 🖥️



# Índice

## Parte 2

- 7 Kata: tenis 🎾
- 8 SonarQube en local 🧪
- 9 Kata: FizzBuzz 🎲





**01\_**

¿Por qué  
testear?

# ¿Por qué testear?

- Asegurar la robustez de software
  - Detectamos errores antes de que lleguen a producción
  - Garantizamos que el software funciona según lo esperado
- Mantenimiento de código
  - Los tests son un “seguro”, puedes realizar cambios con **confianza**
  - Fomenta diseño más limpio
- Pensamos *edge-cases* que no se nos ocurrirían de otra manera
  - Casos que son difíciles de recrear manualmente





# ¿Por qué testear? Confianza

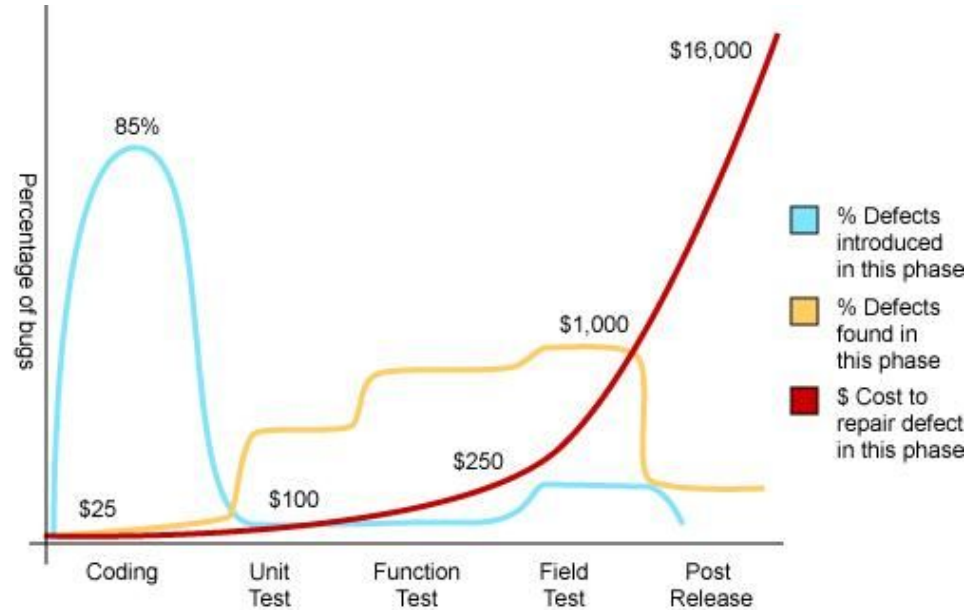
- Confianza para los usuarios (cliente)
- *Demo* de producto más robustas.
  - A priori no deberían detectar errores
- Confianza en el equipo de desarrollo
  - Podemos refactorizar con más libertad, para mantener el código limpio.
  - Sin miedo a introducir nuevo código o funcionalidades
  - Promueve cultura de responsabilidad compartida por la calidad
- Facilita iteraciones más rápidas y frecuentes en proyectos ágiles



# ¿Por qué testear? Coste y tiempo



- El coste y tiempo de resolución de un bug es mayor en producción
- **Objetivo:** detectar los bugs cuanto antes para ahorrar tiempo y costes a largo plazo



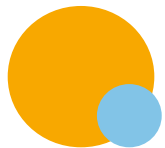
Source: Applied Software Measurement, Capers Jones, 1996





## 02\_

Midiendo la  
calidad de código



# Calidad de código

- **Análisis estático**
  - **Complejidad ciclomática**
  - **Cobertura de código**
  - **Tamaño de método/clases**
    - LOC (Lines of code)
  - **Código duplicado**
- **Herramientas: SonarQube, Jacoco...**
- **\* Principios SOLID, patrones de diseño...**

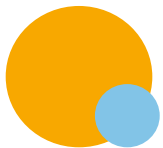


# Calidad de código: complejidad ciclomática



```
public String processOrder(int orderType, boolean isPriority, double amount) {
    String result = "";

    if (orderType == 1) {
        if (isPriority) {
            if (amount > 1000) {
                result = "Express processing for large priority order";
            } else {
                result = "Express processing for small priority order";
            }
        } else {
            if (amount > 500) {
                result = "Standard processing for large regular order";
            } else {
                result = "Standard processing for small regular order";
            }
        }
    } else if (orderType == 2) {
        switch (customerType) {
            case "VIP":
                if (amount > 2000) {
                    [....]
                }
                return result;
        }
    }
}
```



# Calidad de código: complejidad ciclomática

```
public String processOrder(int orderType, boolean isPriority, double amount, String
customerType) {
    if (orderType == 1) {
        return processStandardOrder(isPriority, amount);
    }
    if (orderType == 2) {
        return processVipOrder(customerType, amount);
    }
    return processBulkOrder(amount);
}
```



## Calidad de código: Cobertura

- Mide el porcentaje de código fuente que se ejecuta durante las pruebas
- ¿Qué nos aporta?
  - **Calidad del test**
  - **Identifica partes de código sin probar (identificamos escenarios por cubrir)**
- NO hacemos test para buscar alta cobertura de código, sino para probar funcionalidad.
- Una cobertura alta no garantiza calidad si los tests no validan los resultados correctamente. Es un indicador, no un objetivo.



# Calidad de código: SonarQube

- Lo veremos más adelante, nos proporciona información muy interesante sobre nuestra base de código. Ej:

42min

Debt

6

Code Smells

Maintainability

A



33.3%

Coverage on 12 Lines to cover

14

Unit Tests



0.0%

Duplications on 87 Lines

0

Duplicated Blocks



## 03\_

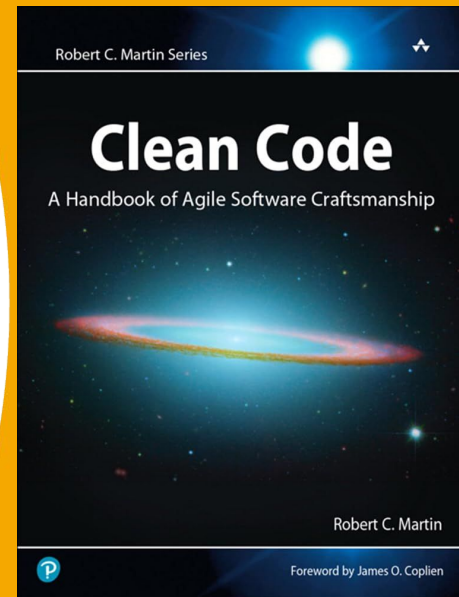
¿Cómo debe ser un test?

- Principios FIRST
- Patrón AAA

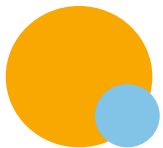


# Principios FIRST

- Fast (rápido)
  - Para ejecutar muchos en poco tiempo
- Independent (independiente)
  - Los test no pueden depender unos de otros
- Repeatable (repetible)
  - Mismo resultado, independiente del entorno
- Self-validating (auto evaluable)
  - Resultado claro: ok/ko
- Timely (oportuno)
  - Deben escribirse idealmente antes de codificar y siempre antes de subir a producción

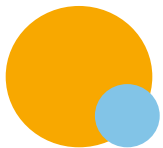






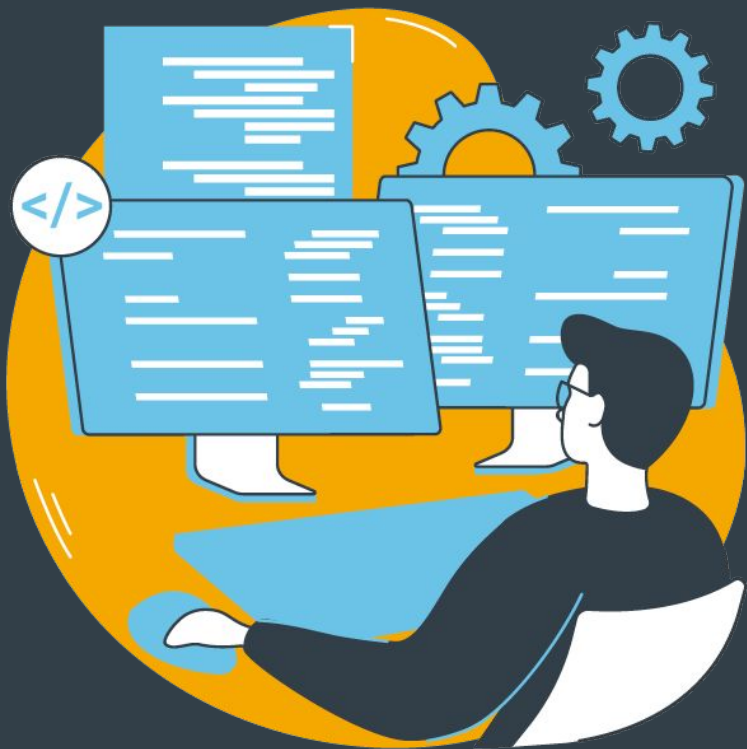
# Patrón AAA

- **Arrange**
  - Configura el **entorno** necesario para ejecutar la prueba: objetos requeridos, **estado inicial** de datos y define dependencias.
- **Act**
  - **Evento principal a probar**
- **Assert**
  - **Verifica que el resultado de la acción coincide con el comportamiento esperado.**



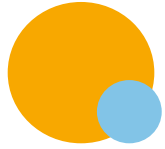
## Patrón AAA: ejemplo

```
@Test new *  
void shouldReturnDirectionWhenSchoolExists() {  
    // Arrange  
    School school = new School();  
  
    // Act  
    String direction = school.getDirection();  
  
    // Assert  
    assertEquals(expected: "Oviedo, Asturias", direction);  
}
```



04\_

TDD: ¿qué es?




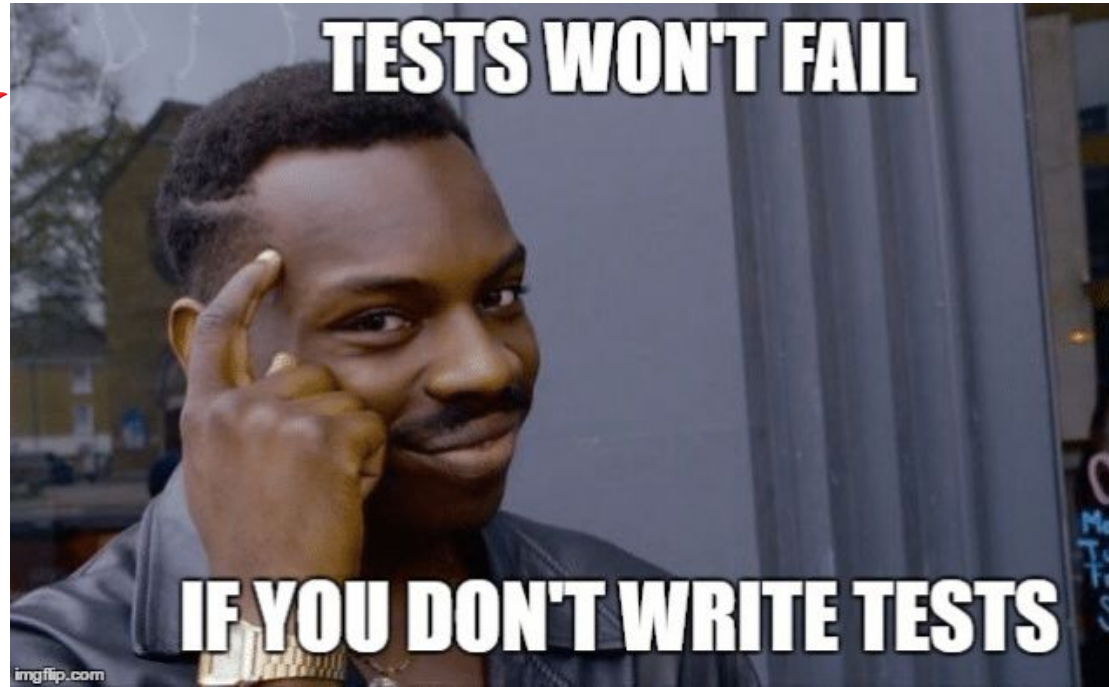
# TDD: ¿qué es?

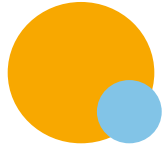
Metodología de desarrollo guiado por pruebas.

Cambia la manera de desarrollar:

Ciclo **Red**-**Green**-Refactor

- 
1. Escribe un **test que falle** 🚩
  2. Haz que el **test pase** ✅
  3. Refactor

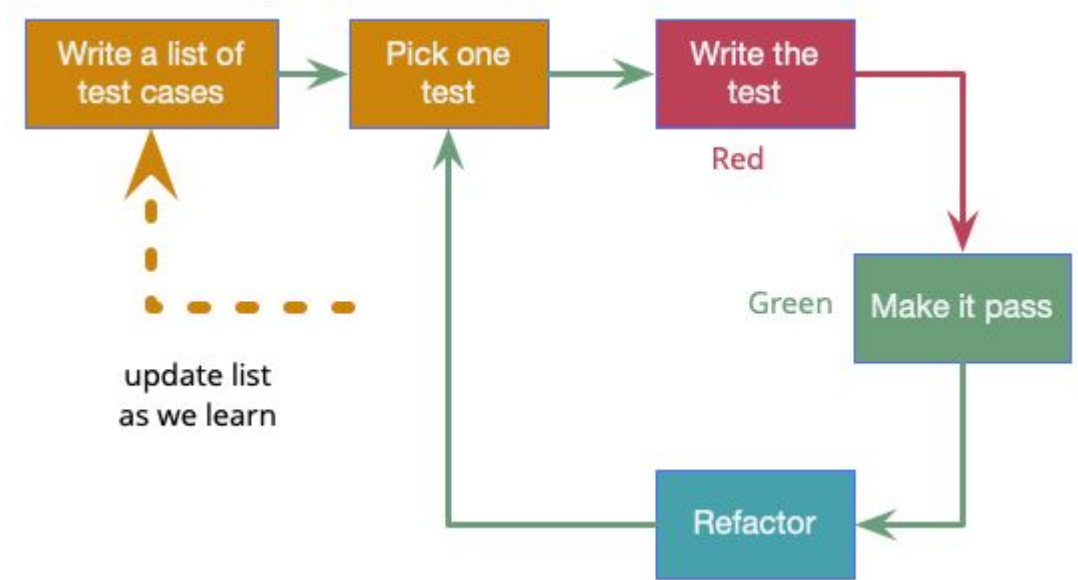




# TDD: ¿qué es?

Importantísimo:

- Identificar Test cases
- Refinar





## TDD: beneficios

- Usar los test al comienzo del desarrollo, nos ayuda a implementar un diseño desacoplado
- El código estará mucho más limpio
  - El refactor es parte del proceso
- El código tendrá alta cobertura de test
  - Como consecuencia de la metodología, no como meta
- Los tests actúan como documentación ejecutable
  - Explican cómo debería comportarse el código en diferentes escenarios.



**Solo escribimos código cuando tenemos un test que falle**



**05\_**

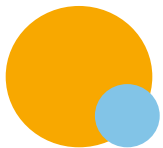
**Unit: conceptos**







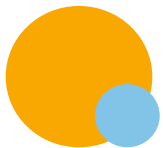
- Conjunto de librerías que nos ayudan en el desarrollo de pruebas, mediante anotaciones dotamos de comportamiento a nuestra batería de pruebas
  - `@Test`
  - `@BeforeEach`, `@BeforeAll`
  - `@AfterEach`, `@AfterAll`
  - `@Timeout`, `@Disable`
  - ...
- <https://junit.org/junit5/docs/current/user-guide/>



## JUnit: @Test

- Anotación que representa una función de test que se ejecutará

```
@Test new *  
void myRandomTest() {  
    // Some test to be executed  
}
```

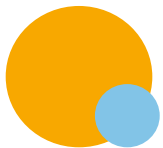


## JUnit: @BeforeEach, @BeforeAll

- Funciones que se ejecutan antes de toda la suite de tests y antes de cada uno en particular

```
@BeforeAll new *
static void setUpEnvironment(){
    // global initialization before execution
}

@BeforeEach new *
void setUpTest(){
    // test initialization stuff for test cases
}
```



## JUnit: @AfterEach, @AfterAll

- Funciones que se ejecutan después de toda la suite de tests y después de cada uno en particular

```
@AfterAll new *
static void cleanEnvironment(){
    // global cleanup after all execution
}

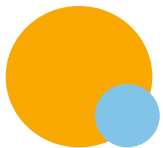
@AfterEach new *
void cleanTest(){
    // test cleanup for every test
}
```



# JUnit: @Timeout

*Table 1. Example timeout configuration parameter values*

Parameter value	Equivalent annotation
42	<code>@Timeout(42)</code>
42 ns	<code>@Timeout(value = 42, unit = NANoseconds)</code>
42 $\mu$ s	<code>@Timeout(value = 42, unit = MICROseconds)</code>
42 ms	<code>@Timeout(value = 42, unit = MILLIseconds)</code>
42 s	<code>@Timeout(value = 42, unit = SECONDS)</code>
42 m	<code>@Timeout(value = 42, unit = MINUTES)</code>
42 h	<code>@Timeout(value = 42, unit = HOURS)</code>
42 d	<code>@Timeout(value = 42, unit = DAYS)</code>



## JUnit: @Disable

```
@Disabled("my optional reason") new *
@Test
void shouldReturnDirectionWhenSchoolExists() {
    School school = new School();

    String direction = school.getDirection();

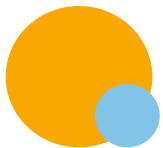
    assertEquals(expected: "Oviedo, Asturias", direction);
}
```

⊗ shouldReturnDirectionWhenSchoolExists()

✓ sampleTrue()

✓ myRandomTest()

10 ms

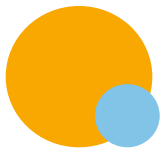


## JUnit: `assertTimeout()`

- Para validar el tiempo de ejecución de una función concreta

```
@Test new *
void checkTimeoutFunction() {
    School school = new School();

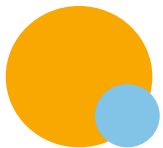
    assertTimeout(Duration.ofMillis(500), ()-> school.getDirection());
}
```



## JUnit: assertThrows() / assertDoesNotThrow()

```
@Test new *  
void checkException() {  
    assertThrows(NullPointerException.class, () -> {  
        System.out.print("testing exceptions!");  
        throw new NullPointerException();  
    });  
}
```



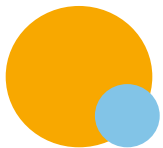


## JUnit: assertEquals() / assertNotEquals()

```
@Test
void shouldReturnDirectionWhenSchoolExists() {
    School school = new School();

    String direction = school.getDirection();

    assertEquals(expected: "Oviedo, Asturias", direction);
}
```

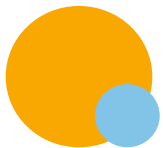


## JUnit: assertTrue() / assertFalse()

```
@Test new *
void shouldReturnDirectionNotEmptyWhenSchoolExists() {
    School school = new School();

    String direction = school.getDirection();

    assertFalse(direction.isEmpty());
}
```



## JUnit: @ParameterizedTest & @ValueSource

- Nos permite ejecutar el mismo test con diferentes inputs

```
@ParameterizedTest(name = "{0} is a valid number") new *  
@ValueSource(strings = {"1", "2", "3"})  
void checkNumbers(String textToCheck) {  
    assertDoesNotThrow( () -> Integer.parseInt(textToCheck));  
}
```

```
<dependency>
```

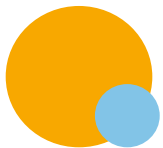
```
<groupId>org.junit.jupiter</groupId>
```

```
<artifactId>junit-jupiter-params</artifactId>
```

```
<version>5.11.4</version>
```

```
<scope>test</scope>
```

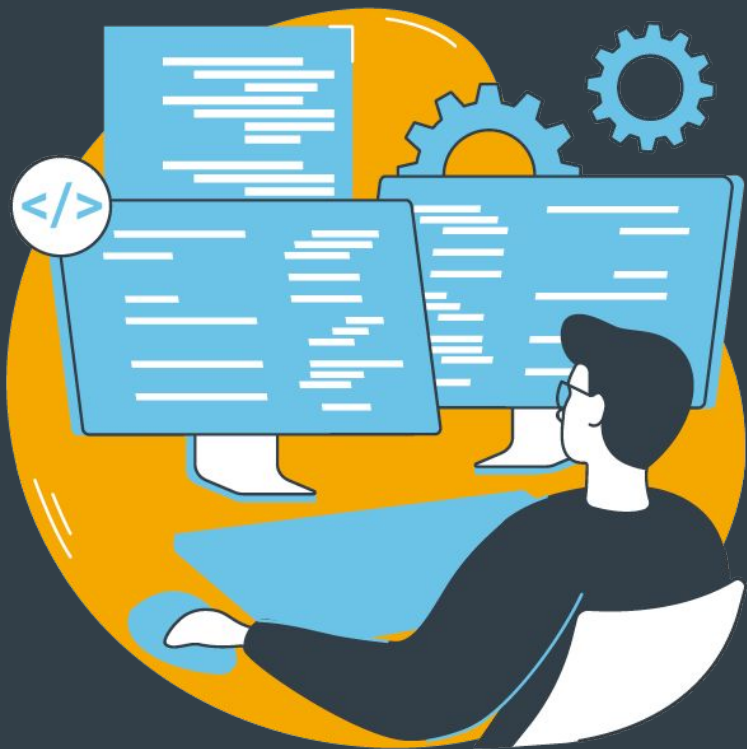
```
</dependency>
```



# JUnit: @ParametrizedTest & @MethodSource

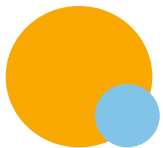
- Fuente de inputs desde un *Factory method*

```
static Stream<Integer> myMethodSource() {  
    return Stream.of(1, 2, 3, 4, 5);  
}  
  
@ParameterizedTest  
@MethodSource("myMethodSource")  
void testSquareFunction(int input) {  
    int expected = input * input;  
    assertEquals(expected, square(input));  
}  
  
private int square(int n) {  
    return n * n;  
}
```



06\_

Preparando el  
entorno 



# Preparando el entorno

- Java 21, Maven

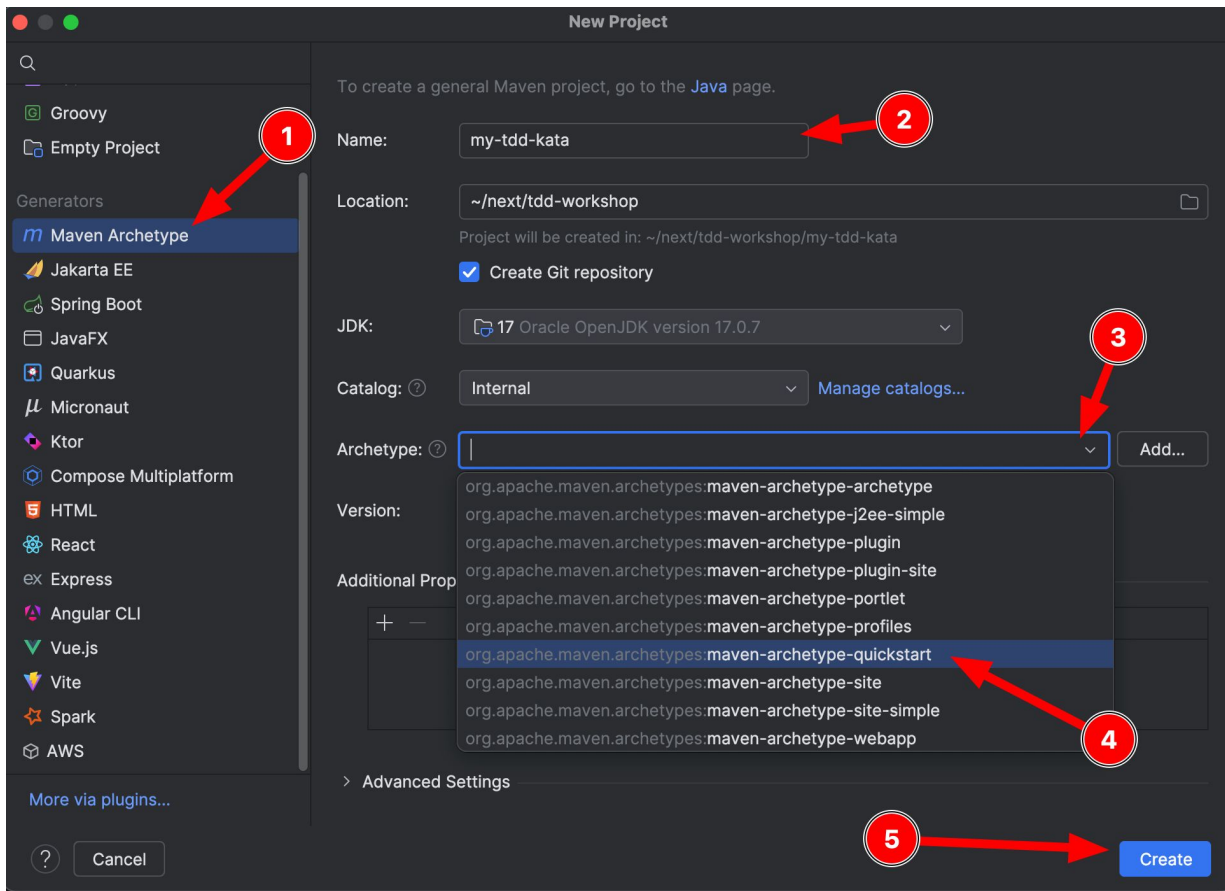
Lo primero: crear un proyecto Java+Maven básico con JUnit.

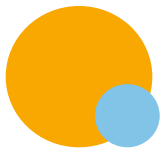
```
mvn archetype:generate
-DarchetypeGroupId=org.apache.maven.archetypes \
-DarchetypeArtifactId=maven-archetype-quickstart \
-DarchetypeVersion=1.5 \
-DgroupId=org.kata.tennis \
-DartifactId=tennis-tdd-workshop \
-DinteractiveMode=false
```

<https://maven.apache.org/archetypes/maven-archetype-quickstart/>  
<https://github.com/miguelms95/tdd-tennis-workshop>



# Preparando el entorno : maven-archetype-quickstart





## Preparando el entorno : dependencias

```
<dependency>
  <groupId>org.junit.jupiter</groupId>
  <artifactId>junit-jupiter-api</artifactId>
  <version>5.11.4</version>
  <scope>test</scope>
</dependency>
<dependency>
  <groupId>org.junit.jupiter</groupId>
  <artifactId>junit-jupiter-params</artifactId>
  <version>5.11.4</version>
  <scope>test</scope>
</dependency>
```





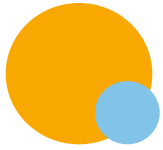
07\_

Kata: tenis 🎾



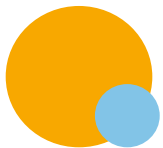
## Kata: tennis

- Puntuación de cada juego: 0, 15, 30, 40
- Si los dos jugadores tienen 40 puntos están en “deuce”
  - Si un jugador hace punto, estará en “ventaja”
  - Si el jugador con ventaja hace punto, gana el juego
  - Si el jugador sin ventaja hace punto, vuelven a “deuce”
- La nomenclatura de los puntos: “love”, “fifteen”, “thirty” y “forty”.
- El partido lo ganará el primero que gane 4 juegos y al menos 2 juegos más que el oponente



## Kata: Test Cases

0-0	love-love
1-0	fifteen-love
2-0	thirty-love
3-0	forty-love
0-1	love-fifteen
0-2	love-thirty
...	
3-3	deuce
4-3	Advantage player 1
3-4	Advantage player 2
5-3	player 1 wins
3-5	player 2 wins



# Kata: Test Cases

## Basic Scoring

- ✓ `shouldReturnLoveLove_OnStart()` → Score is "Love-Love" at start.
- ✓ `testPlayerOneScoresOnce_ShouldBeFifteenLove()` → "Fifteen-Love".
- ✓ `testPlayerTwoScoresOnce_ShouldBeLoveFifteen()` → "Love-Fifteen".
- ✓ `testBothPlayersScoreOnce_ShouldBeFifteenAll()` → "Fifteen-All".
- ✓ `testPlayerOneScoresTwice_ShouldBeThirtyLove()` → "Thirty-Love".
- ✓ `testPlayerTwoScoresTwice_ShouldBeLoveThirty()` → "Love-Thirty".

## Deuce and Advantage

- ✓ `testScoreIsFortyAll_ShouldBeDeuce()` → "Deuce".
- ✓ `testPlayerOneAdvantageAfterDeuce_ShouldBeAdvantagePlayerOne()` → "Advantage Player 1".
- ✓ `testPlayerTwoAdvantageAfterDeuce_ShouldBeAdvantagePlayerTwo()` → "Advantage Player 2".
- ✓ `testPlayerOneWinsAfterAdvantage_ShouldBeWinPlayerOne()` → "Player 1 wins".
- ✓ `testPlayerTwoWinsAfterAdvantage_ShouldBeWinPlayerTwo()` → "Player 2 wins".
- ✓ `testAdvantageLost_BackToDeuce()` → "Deuce" if the player with Advantage loses the next point.

## Winning the Game

- ✓ `testPlayerOneWinsByTwoPoints_ShouldBeWinPlayerOne()` → "Player 1 wins".
- ✓ `testPlayerTwoWinsByTwoPoints_ShouldBeWinPlayerTwo()` → "Player 2 wins".









# 08\_

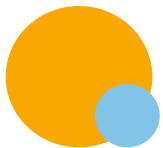
## SonarQube en local





# SonarQube en local

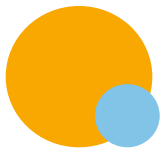
- SonarQube es una plataforma de análisis estático de código que detecta errores, vulnerabilidades y deuda técnica, ayudando a mantener código limpio y seguro.
- Características
  -  **Análisis Automático:** Detecta bugs, code smells y vulnerabilidades.
  -  **Cobertura de Pruebas:** Integra con JaCoCo para medir cobertura de tests.
  -  **Soporte Multi-Lenguaje:** Compatible con Java, JavaScript, Python, etc.
  -  **Integración CI/CD:** Funciona con Jenkins, GitHub Actions, GitLab CI/CD.
  -  **Dashboards Detallados:** Métricas de calidad en una interfaz intuitiva.
  -  **Reglas Personalizables:** Configura estándares de calidad según necesidades.



# SonarQube en local

## 1. Configura `sonar-project.properties`

```
sonar.projectKey=mi-proyecto
sonar.projectName=Mí Proyecto Java
sonar.host.url=http://localhost:9000
sonar.login=MI_TOKEN_DE_SONAR
sonar.language=java
sonar.sourceEncoding=UTF-8
sonar.tests=src/test/java
sonar.java.binaries=target/classes
sonar.test.inclusions=**/*Test.java
sonar.junit.reportPaths=target/surefire-reports
sonar.jacoco.reportPaths=target/jacoco.exec
```



# SonarQube en local

2. Levantar con [Docker](#) una instancia de Sonar y de PostgreSQL

- Se levanta SonarQube en <http://localhost:9000>
-  Usuario y contraseña por defecto: admin / admin

```
docker network create sonar-network
```

```
docker run -d --name sonar-db --network sonar-network \  
  -e POSTGRES_USER=sonar \  
  -e POSTGRES_PASSWORD=sonar \  
  -e POSTGRES_DB=sonarqube \  
  postgres:15
```

```
docker run -d --name sonar --network sonar-network -p 9000:9000 \  
  -e SONAR_JDBC_URL=jdbc:postgresql://sonar-db:5432/sonarqube \  
  -e SONAR_JDBC_USERNAME=sonar \  
  -e SONAR_JDBC_PASSWORD=sonar \  
  sonarqube:lts-community
```





## SonarQube en local

2. Levantar con [Docker](#) una instancia de Sonar y de PostgreSQL

- Se levanta SonarQube en <http://localhost:9000>
-  Usuario y contraseña por defecto: admin / admin

En docker-compose.yml

<https://github.com/miguelms95/tdd-tennis-workshop/blob/feat/sonar/docker-compose.yml>

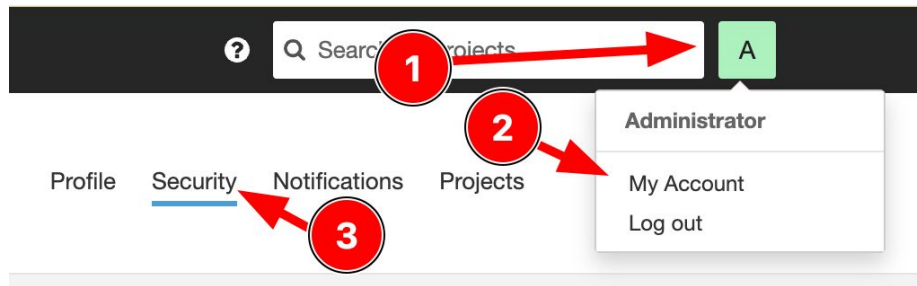
Ejecuta: **docker-compose up**



# SonarQube en local

## 3. Generar un token en SonarQube

- Ve a <http://localhost:9000>
- Inicia sesión con admin/admin
- Crea un nuevo token de acceso en "My Account" → "Security"
  - Nombre "test"
  - Tipo: "User token"



### Generate Tokens

Name

Type

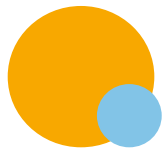
User Token ▼

Expires in

30 days ▼

**Generate**

- Copia el token generado (lo usarás en el siguiente paso).



## SonarQube en local

4. Pega el token en el fichero sonar-project.properties

```
sonar.login=TU_TOKEN_SONAR
```

5. Añade las dependencias de **Sonar** y **Jacoco** al proyecto para capturar la cobertura de código

# SonarQube en local

```
<build>
  <plugins>
    <plugin>
      <groupId>org.sonarsource.scanner.maven</groupId>
      <artifactId>sonar-maven-plugin</artifactId>
      <version>3.9.1.2184</version>
    </plugin>
    <plugin>
      <groupId>org.jacoco</groupId>
      <artifactId>jacoco-maven-plugin</artifactId>
      <version>0.8.10</version>
      <executions>
        <execution>
          <goals>
            <goal>prepare-agent</goal>
          </goals>
        </execution>
        <execution>
          <id>report</id>
          <phase>verify</phase>
          <goals>
            <goal>report</goal>
          </goals>
        </execution>
      </executions>
    </plugin>
  </plugins>
</build>
```



# SonarQube en local

`mvn clean verify sonar:sonar -Dsonar.login=TU_TOKEN_AQUI`

- Reporte de Jacoco en: <target/site/jacoco/index.html>
- Reporte de SonarQube: <http://localhost:9000>

42min

Debt

6

 Code Smells

Maintainability

**A**



33.3%

Coverage on 12 Lines to cover

14

Unit Tests

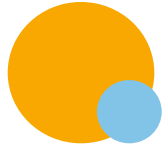


0.0%

Duplications on 87 Lines

0

Duplicated Blocks



:next DIGITAL

Somos una consultora especializada en **desarrollo de software,**  
**soluciones de analítica avanzada e**  
**inteligencia artificial.**

**Nuestras áreas principales de trabajo son:**



Cloud



Data & AI



Advanced Analytics



SW Development

[SirviendoCodigo.com](http://SirviendoCodigo.com)

[www.youtube.com/@SirviendoCodigo](http://www.youtube.com/@SirviendoCodigo)



# Gracias!

[aulaupm@nextdigital.es](mailto:aulaupm@nextdigital.es)

*La simplicidad es la máxima sofisticación*  
Leonardo Da Vinci

✉ [mmartinez@nextdigital.es](mailto:mmartinez@nextdigital.es)

✂ [@miguelms\\_es](https://twitter.com/miguelms_es)

👤 [LinkedIn: Miguel Martínez Serrano](#)

💻 [miguelms.es](https://miguelms.es)

