Clase grabada

Recibe una cálida:

¡Bienvenida!

Te estábamos esperando 😁

English Always | alkemy

Clase grabada

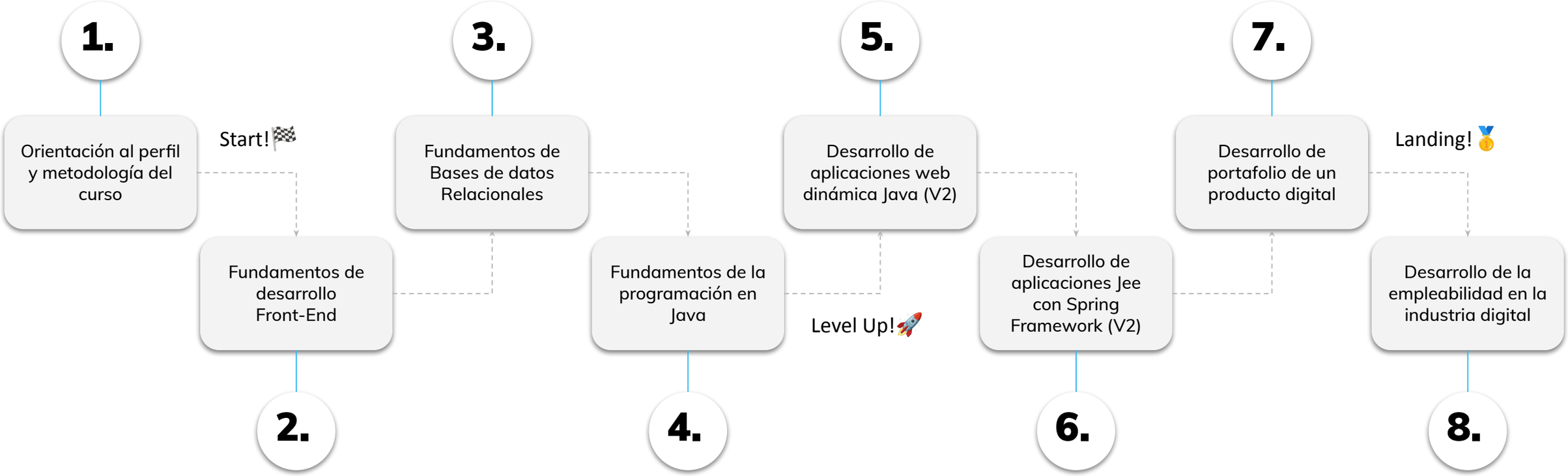
Pruebas Unitarias en Java

Plan formativo: Desarrollo de Aplicaciones Full Stack Java Trainee V2.0

English Always | alkemy

HOJA DE RUTA

¿Cuáles **skill** conforman el programa?



English Always | alkemy

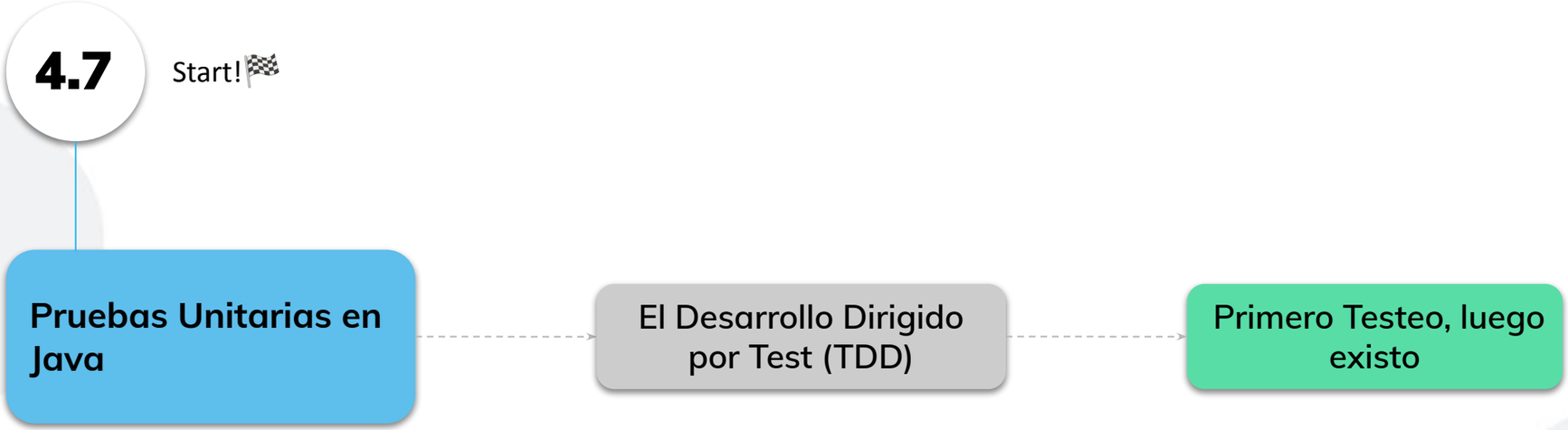
REPASO CLASE ANTERIOR

En la clase anterior trabajamos 📚:

✔ *Implementación de Suites de Pruebas*

English Always | alkemy

LEARNING PATHWAY



English Always | alkemy

OBJETIVOS DE APRENDIZAJE

¿Qué aprenderemos?

- ***Comprender el concepto e implementación del Desarrollo Dirigido por Test***

Desarrollo Dirigido por Test (TDD)

Desarrollo Dirigido por Test (TDD)

¿Qué es?:

Es una metodología de programación en la que **se escriben primero las pruebas**, normalmente unitarias, **y después se escribe el código fuente** de tal manera que supere dichos tests. Una vez llegados a este punto, **se refactoriza el código escrito**.

Al hacerlo así, se consigue mejorar la calidad del software final y, además, se reducen los costes de mantenimiento.

Desarrollo Dirigido por Test (TDD)

¿Cómo se desarrollan los test?

El ciclo que siguen los programadores en el test-driven development también recibe el nombre de **red-green-refactor** y describe cada una de las fases que deben cumplirse para alcanzar una mayor eficiencia:

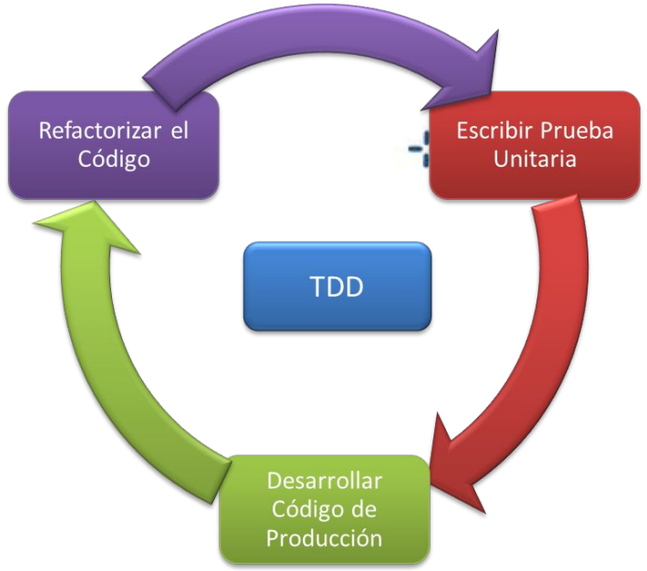
1. **Fase roja (red phase)**
2. **Fase verde (green phase)**
3. **Refactoring**

Desarrollo Dirigido por Test (TDD)

Fase roja (red phase)

En esta fase hay que ponerse en los zapatos del usuario, que quiere poder usar el código de forma sencilla.

Se redacta, por lo tanto, **un test que contenga componentes que aún no hayan sido implementados**, para luego decidir qué elementos son realmente necesarios para que el código funcione.

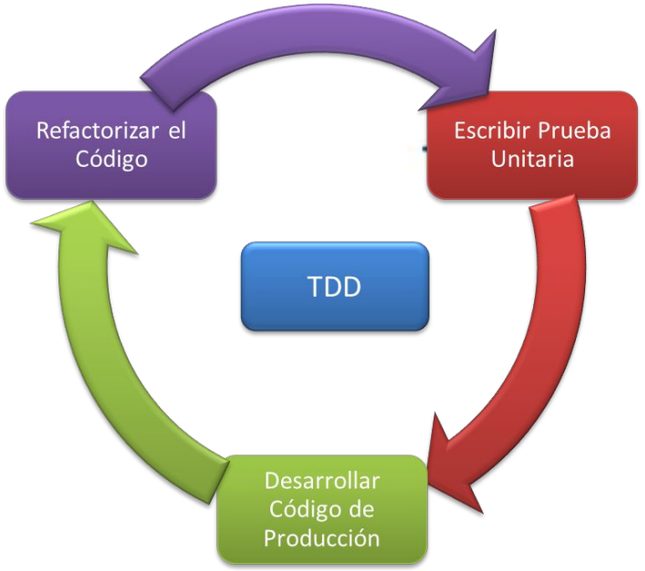


Desarrollo Dirigido por Test (TDD)

Fase verde (green phase)

Suponiendo que el test falle y se marque en rojo, se adopta entonces el papel de programador y se intenta encontrar una solución simple.

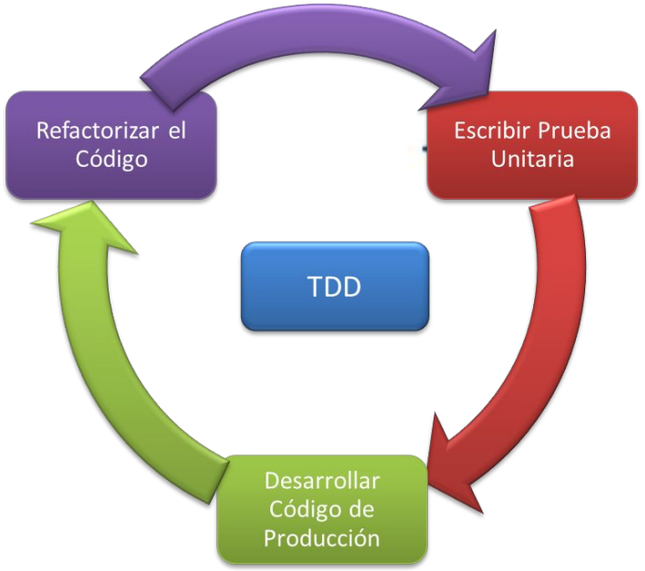
Es muy importante **redactar únicamente la cantidad de código que sea necesaria**. El código redactado se integra luego en el código productivo, de forma que el test quede marcado en verde.



Desarrollo Dirigido por Test (TDD)

Refactoring

En este paso, **el código productivo se pasa a limpio y se perfecciona su estructura**. Podría decirse que ahora se completa y reestructura de manera que resulte elegante y comprensible para los desarrolladores. Entre otras cosas, se eliminan los duplicados en el código, y se vuelve más profesional.



Desarrollo Dirigido por Test (TDD)

Ventajas del TDD

- **El diseño software se vuelve modular**: los desarrolladores se centran en una única característica cada vez. Así, no se pasa a la siguiente fase hasta que se supera la prueba unitaria asociada.
- **El código es más fácil de mantener**: la tarea de centrarse en trozos de código más pequeños y digeribles requiere menos esfuerzo por parte de los desarrolladores.
- **La cobertura de las pruebas es muy alta**: siguiendo esta metodología, no debería existir ningún fragmento de código escrito que no tenga pruebas asociadas.
- **Mejora de la calidad del código**: el foco de desarrollo se pone en la escritura de código para pasar las pruebas descritas.

Desarrollo Dirigido por Test (TDD)

¿Cómo se crea un ejemplo de TDD en un proyecto real?

Supongamos que el cliente nos pide que desarrollemos una calculadora que sume números.

Acordamos con el cliente que el criterio de aceptación sería que si se introducen en la calculadora dos números y se selecciona la operación de suma, la calculadora muestre el resultado de la suma en la pantalla.

Teniendo en cuenta este criterio, se comienza a definir el funcionamiento del algoritmo de suma y convertimos el criterio de aceptación en una prueba concreta, por ejemplo, un algoritmo que si se introduce un 3 y un 5 te devuelva un 8.

En la imagen podemos ver cómo definimos el código de criterio de aceptación dentro de un test.

```
public void testSuma() { assertEquals(8,
Calculadora.suma(3,5)); }
```

Desarrollo Dirigido por Test (TDD)

¿Cómo se crea un ejemplo de TDD en un proyecto real?

Luego ya es más sencillo poder definir el código productivo, que sería el que vemos en la imagen.

Al haber hecho estos pasos, verificamos si nuestro test lo logró. Al ejecutar, nos debería dar el flag verde.

```
public class Calculadora { public static
int b) { return a+b; } }
```

Desarrollo Dirigido por Test (TDD)

¿Cómo se crea un ejemplo de TDD en un proyecto real?

Un test tiene tres partes, que se identifican con las siglas AAA en inglés: **Arrange** (Preparar), **Act** (Actuar), **Assert** (Afirmar).

Podemos aplicar un refactor al código en el test. Veamos un ejemplo aplicando el "AAA" en la imagen:

```
public void testSuma() {

    //Arrange
    int a = 3;
    int b = 5;

    //Act
    int resultado = Calculadora.suma(a,b);

    //Assert
    assertEquals(8, resultado);
}
```

Evaluación Integradora ✨

¿Listos para un nuevo desafío? En esta clase comenzamos a construir nuestro....

Trabajo Integrador del Módulo 💪🏽

Iremos completándolo progresivamente clase a clase.

English Always | alkemy

LIVE CODING

Ejemplo en vivo

Vamos a practicar el TDD:

Como hemos visto, en TDD, primero escribimos las pruebas y luego implementamos el código para que las pruebas pasen. En este ejercicio, comenzaremos escribiendo las pruebas y luego implementaremos la clase CuentaBancaria.

1. *Primero, escribimos una prueba que describa el comportamiento esperado de nuestra clase CuentaBancaria. Usaremos el framework JUnit para escribir la prueba.*

Tiempo: 30 minutos

LIVE CODING

Ejemplo en vivo

Veamos un ejemplo de Test a implementar:

En este código, hemos escrito tres pruebas para CuentaBancaria.

- *La primera prueba verifica que el saldo inicial sea cero.*
- *La segunda prueba verifica que el método depositar aumente el saldo correctamente.*
- *La tercera prueba verifica que el método retirar reduzca el saldo correctamente.*

```java
@Test
public void saldoInicialDeberiaSerCero() {
    CuentaBancaria cuenta = new CuentaBancaria();
    assertEquals(0, cuenta.getSaldo());
}

@Test
public void depositarDeberiaAumentarElSaldo() {
    CuentaBancaria cuenta = new CuentaBancaria();
    cuenta.depositar(100);
    assertEquals(100, cuenta.getSaldo());
}

@Test
public void retirarDeberiaReducirElSaldo() {
    CuentaBancaria cuenta = new CuentaBancaria(200);
    cuenta.retirar(50);
    assertEquals(150, cuenta.getSaldo());
}
```

LIVE CODING

Ejemplo en vivo

2. *Implementar la clase CuentaBancaria para que las pruebas pasen.*

3. *Ejecutar las pruebas JUnit y asegurarse de que todas pasen correctamente. Si alguna prueba falla, deberemos corregir el código de la clase CuentaBancaria hasta que todas las pruebas pasen sin errores.*

```java
public class CuentaBancaria {
    private int saldo;

    public CuentaBancaria() {
        this.saldo = 0;
    }

    public CuentaBancaria(int saldoInicial) {
        this.saldo = saldoInicial;
    }

    public int getSaldo() {
        return saldo;
    }

    public void depositar(int cantidad) {
        if (cantidad > 0) {
            saldo += cantidad;
        }
    }

    public void retirar(int cantidad) {
        if (cantidad > 0 && cantidad <= saldo) {
            saldo -= cantidad;
        }
    }
}
```

Ejercicio N° 1

Primero Testeo, luego existo

Primero Testeo, luego existo

Es hora de practicar: 🙌

Para poner a prueba lo aprendido vamos a aplicar el Test Driven Development en un ejercicio simple.

Consigna: ✍️

Implementar una función **esPar(int num)** que reciba un número entero y retorne **true si es par o false si es impar**.

Recuerda que debes seguir los pasos del TDD: primero escribir un test que falle, luego el código mínimo (función esPar) para que pase, y finalmente refactorizar el código a necesidad.

Tiempo🕐: *30 minutos*

¿Alguna consulta?

RESUMEN

¿Qué logramos en esta clase?

- ✓ ***Comprender el concepto de TDD (Desarrollo Dirigido por Test)***
- ✓ ***Reconocer la estructura de implementación de los TDD***

English Always | alkemy

#WorkingTime

Continuemos ejercitando

¡Antes de cerrar la clase! Te invitamos a: 👇👇👇

1. Repasar nuevamente la grabación de esta clase
2. Revisar el material compartido en la plataforma de Moodle (lo que se vio en clase y algún ejercicio adicional)
   a. *Lectura Módulo 4, Lección 7: páginas 16 - 17*
3. Traer al próximo encuentro, todas tus dudas y consultas para verlas antes de iniciar nuevo tema.

¡Muchas Gracias!

Nos vemos en la próxima clase 👋🏻

English Always | alkemy

Momento:

Time-out!

⏰5 min.