Clase grabada

Recibe una cálida:

¡Bienvenida!

Te estábamos esperando 😁

English Always | alkemy

Clase grabada

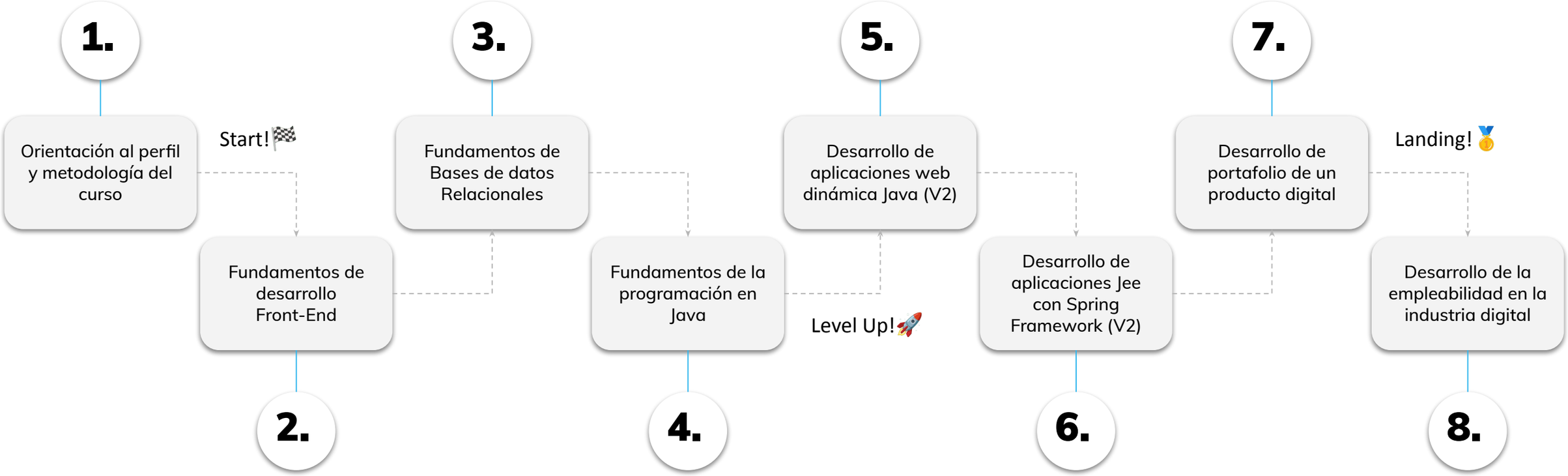
Pruebas Unitarias en Java

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 📚:

- ✔ *Principio de inversión de dependencias*

English Always | alkemy

LEARNING PATHWAY

4.7 Start!🏁

Pruebas Unitarias en Java → Características: Ventajas y limitaciones → Testeando los Test

Características de las pruebas unitarias. Ventajas y limitaciones.

English Always | alkemy

OBJETIVOS DE APRENDIZAJE

¿Qué aprenderemos?

- *Conocer las características de los test unitarios en Java*
- *Comprender las ventajas y limitaciones de los test unitarios en Java*

Rompehielo🥶

Respondan en el chat o levantando la mano: 🙌

1. ¿Cómo podríamos probar que el método sumar funciona correctamente?
2. ¿Qué beneficios obtendremos de escribir estas pruebas?
3. ¿En qué casos sería más complicado escribir pruebas automáticas?
4. ¿Creen que vale la pena el esfuerzo de construir estas pruebas? ¿Por qué?

```
public class Calculadora {

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

}
```

Test Unitarios:
Características

Test Unitarios

¿Qué son?:

Los test unitarios en Java son una parte fundamental de la práctica de pruebas en el desarrollo de software. Son **pequeños fragmentos de código diseñados para evaluar el comportamiento de una unidad de código aislada**, por lo general una función o método, en un entorno controlado. Estas pruebas verifican si la unidad de código (como una clase o método) se comporta como se espera.

Test Unitarios

Características principales

- **Independencia**: Las pruebas unitarias deben ser independientes entre sí, lo que significa que cada prueba debe poder ejecutarse de forma aislada sin depender de otras pruebas o del entorno externo.

- **Automatización**: Las pruebas unitarias se diseñan para ser ejecutadas de forma automatizada. Se utilizan frameworks como JUnit, TestNG o Mockito para facilitar la creación y ejecución de las pruebas.

- **Aislamiento**: Cada prueba unitaria debe probar una única funcionalidad o comportamiento específico. Para lograr esto, se utilizan mocks o stubs para aislar la unidad en prueba de sus dependencias externas.

Test Unitarios

Características principales

- **Cobertura o Coverage**: Las pruebas unitarias deben cubrir la mayor cantidad de código posible. El objetivo es asegurar que todas las funcionalidades y caminos del código sean probados, minimizando la posibilidad de errores.

- **Repetitividad**: Las pruebas unitarias deben ser repetibles y producir resultados consistentes. Esto significa que deben poder ejecutarse en cualquier entorno sin depender de factores externos.

- **Rapidez**: Las pruebas unitarias deben ser rápidas de ejecutar, ya que se espera que se puedan ejecutar frecuentemente, incluso durante el proceso de desarrollo.

- **Legibilidad**: Es importante que las pruebas unitarias sean claras y comprensibles para facilitar su mantenimiento y comprensión por parte de otros desarrolladores.

Test Unitarios: Ventajas

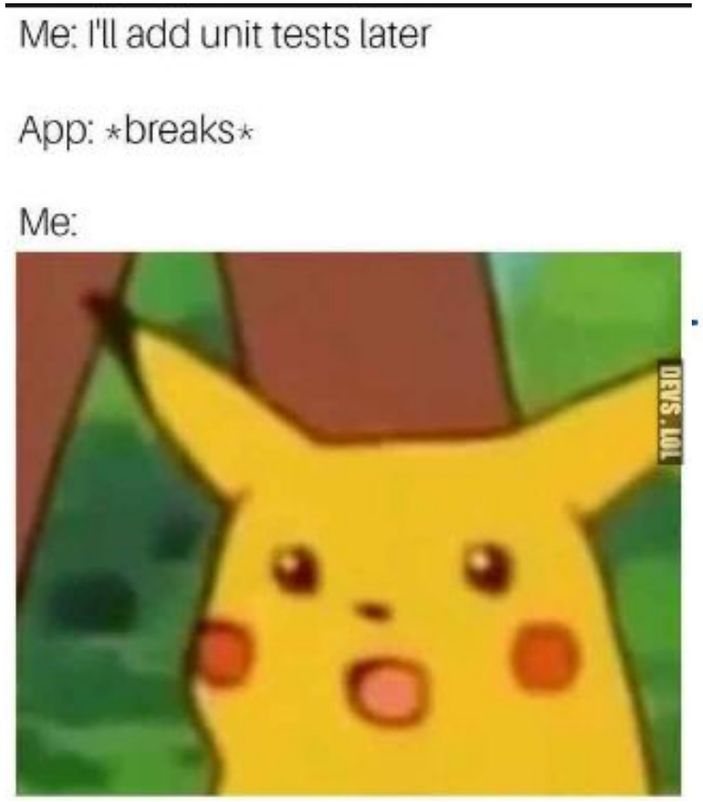
La principales ventajas de implementar test unitarios en nuestros programas son:

- **Detección temprana de errores**: Las pruebas unitarias te permiten identificar errores y problemas en el código de manera temprana. Al ejecutar las pruebas de forma regular, puedes capturar y corregir los errores en las primeras etapas del desarrollo, lo que facilita la solución de problemas y evita que se propaguen a otras partes del código.
- **Mejora de la calidad del código**: Las pruebas unitarias fomentan el desarrollo de un código de alta calidad. Al escribir pruebas, debes pensar en la funcionalidad de forma aislada y en cómo probarla adecuadamente. Esto conduce a un código más modular, desacoplado y mantenible.

Test Unitarios: Ventajas

La principales ventajas de implementar test unitarios en nuestros programas son:

- **Facilita el proceso de refactorización**: Las pruebas unitarias actúan como una red de seguridad al realizar refactorizaciones en el código. Siempre que realices cambios en tu código, puedes ejecutar las pruebas unitarias para asegurarte de que las modificaciones no hayan introducido nuevos errores.
- **Documentación y comprensión del código**: Las pruebas unitarias bien escritas sirven como documentación viva de cómo se espera que funcione el código. Al leer las pruebas, otros desarrolladores pueden comprender rápidamente el comportamiento esperado de una determinada funcionalidad o clase.

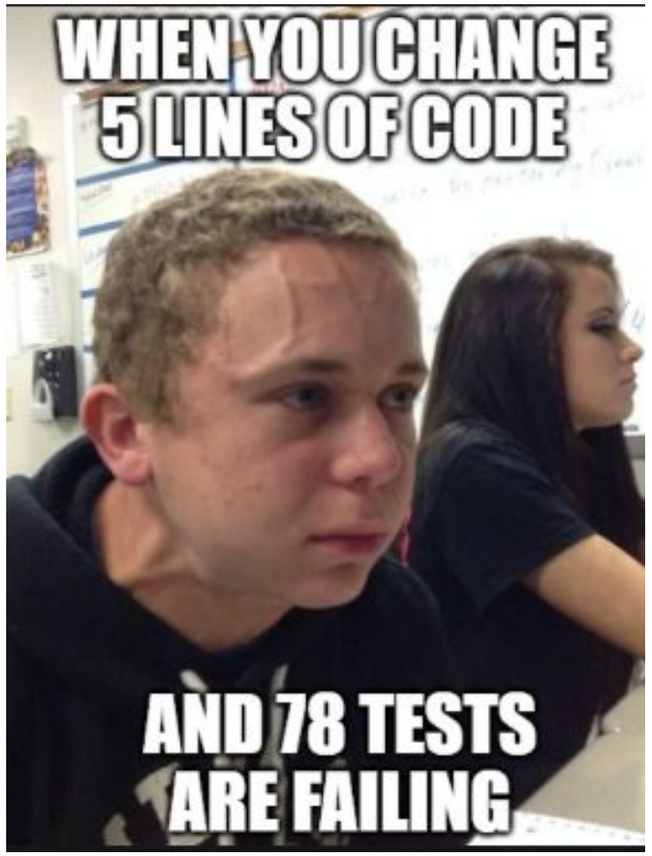


Test Unitarios: Ventajas

La principales ventajas de implementar test unitarios en nuestros programas son:

- **Facilita el trabajo en equipo**: Las pruebas unitarias permiten a los desarrolladores trabajar en paralelo y de manera colaborativa. Cada desarrollador puede escribir pruebas para las funcionalidades que está implementando y ejecutarlas para asegurarse de que no haya conflictos con otras partes del código.

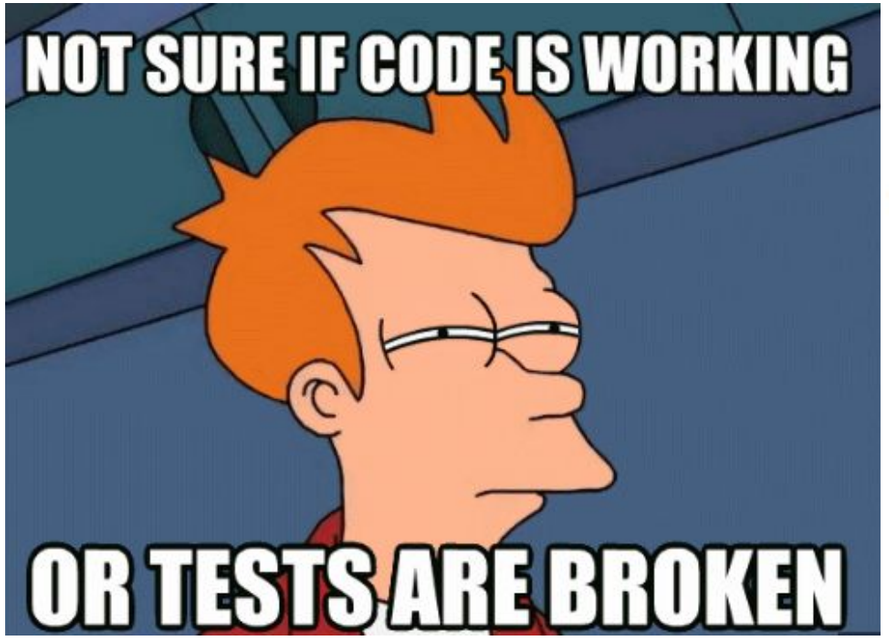
- **Mejora la productividad**: Aunque escribir pruebas unitarias requiere tiempo adicional, a largo plazo, puede mejorar la productividad del equipo. Al detectar y corregir errores tempranamente, se reducen los tiempos de depuración y se evitan problemas en etapas posteriores del desarrollo.



Test Unitarios: Ventajas

La principales ventajas de implementar test unitarios en nuestros programas son:

- **Facilita la integración continua y entrega continua**: Las pruebas unitarias son un componente clave de las prácticas de integración continua y entrega continua. Al tener un conjunto de pruebas sólidas, puedes automatizar su ejecución en los sistemas de integración continua, lo que garantiza que el código sea siempre estable y funcional.



Test Unitarios: Desventajas

Podemos encontrarnos con distintas escuelas o líneas de pensamiento que expongan "**Desventajas**" de las pruebas unitarias, pero la realidad es que **TODAS** las desventajas, **pueden ser reducidas**, logrando destacar así las ventajas.

Las principales **desventajas** a la hora de implementar Test unitarios en nuestro código y las posibles soluciones son:

- **Costo de desarrollo**: Es necesario invertir tiempo y recursos para escribir las pruebas y mantenerlas actualizadas a medida que el código evoluciona. Esto puede aumentar la carga de trabajo inicial y prolongar los plazos de entrega. Pero sabiendo que si son bien planteadas, otorgan la posibilidad de estabilidad de la app ante futuros cambios.

Test Unitarios: Desventajas

- **Curva de aprendizaje**: Para escribir pruebas unitarias efectivas, los desarrolladores deben familiarizarse con los frameworks y herramientas de pruebas, como JUnit o TestNG. Esto implica una curva de aprendizaje adicional para el equipo de desarrollo, especialmente si no están familiarizados con estas herramientas. Si bien es una desventaja, no es nada que no se pueda solucionar estudiando sobre Test Unitarios y practicando en nuestro dia a dia.
- **Mantenimiento continuo:** A medida que el código evoluciona, las pruebas unitarias también deben actualizarse y mantenerse. Si no se actualizan correctamente, las pruebas pueden volverse obsoletas y generar falsos positivos o falsos negativos. Esto requiere dedicar tiempo y esfuerzo para mantener el conjunto de pruebas actualizado y relevante.

Test Unitarios: Desventajas

- **Cobertura limitada**: Aunque las pruebas unitarias pueden cubrir una parte significativa del código, es posible que no logren cubrir todos los casos posibles. Algunos aspectos del sistema, como la interacción con bases de datos o servicios externos, pueden ser difíciles de probar mediante pruebas unitarias, lo que limita la cobertura total.

- **Sobrecarga de pruebas**: Si no se administra adecuadamente, el exceso de pruebas unitarias puede generar una sobrecarga en el proceso de desarrollo. Demasiadas pruebas pueden ralentizar el tiempo de compilación y ejecución, y puede ser difícil mantener todas las pruebas actualizadas y relevantes. Esto puede ser tranquilamente solucionado, planteando bien los diseños y arquitecturas de la app, con el fin de evitar monolitos saturados en múltiples módulos y tests.

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 DEMO

Ejemplo en vivo

Imaginemos el siguiente escenario y respondamos entre todos:

Antes de permitir la creación de una CuentaBancaria, debemos crear una función que valide si una persona es mayor de edad según su edad ingresada.

1. *¿Cómo podrían probar que la función funciona correctamente antes de usarla?*
2. *¿Qué ventajas les daría crear pruebas unitarias?*
3. *¿Pero también habría desventajas o limitaciones?*

Tiempo: 20 minutos

Ejercicio N° 1

Testeando los Test

Testeando los Test

¡Vamos a evaluar pros y contras!: 🙌

Debes definir cuáles de las siguientes características de los Test Unitarios son PROS y cuáles son CONTRAS.

En sala reducidas, respondan con pro o contra: ✍️

- Detecta errores temprano
- No prueban la UI ni la DB
- Requieren mantenimiento
- Facilitan el refactoring
- Aíslan el código bajo prueba
- Aumentan la carga de desarrollo inicial
- Automatización de las pruebas
- Simplicidad y rapidez de ejecución
- Sobrecarga si no se administran bien
- Requiere aprender frameworks de testing
- Independencia entre pruebas
- Complejidad de mockear dependencias

Al finalizar, tendremos una puesta en común en la sala principal

Tiempo🕐: *20 minutos*

¿Alguna consulta?

RESUMEN

¿Qué logramos en esta clase?

- ✓ *Comprender la definición y características de los Test Unitarios*
- ✓ *Reconocer las ventajas y desventajas de los Test Unitarios*

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 2 - 5*
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.