

AWAKELA

#programmingbootcamp

PRUEBAS UNITARIAS



The image is a collage of screenshots from the video game DOOM. It features several different levels, including Hell, Helltrain, and Helltrain Station. The screenshots show various weapons like the Super Shotgun, BFG9000, and Plasma Gun. There are also numerous enemies like Hell Knights, Hell Dukes, and Hell Marine. The collage is set against a dark background with a large, stylized logo in the center.



PRUEBAS UNITARIAS

¿QUÉ VAMOS A VER?

- Características.
- Ventajas y Limitaciones.
- Introducción a JUnit.
- ¿Por qué aplicar Test?
- Características de JUnit.
- Caso de uso.





CARACTERÍSTICAS DE LAS PRUEBAS UNITARIAS

- Se aseguran de que el código no presente fallos, errores o cálculos inesperados, y que siempre nos retorne el valor que realmente estamos esperando.
- No solamente aplicamos los test sobre el funcionamiento de los métodos, también es posible verificar que los argumentos o parámetros sean del tipo correcto.
- **¿Las pruebas unitarias son perfectas?** Por supuesto que **no**. Pero si es un primer filtro clave para asegurarnos que se sigan correctamente las buenas prácticas de programación.



unit testing



CARACTERÍSTICAS DE LAS PRUEBAS UNITARIAS

- Para tener pruebas unitarias de calidad existen ciertos requisitos que se deben cumplir:
- **Automatizable:** No debería ser un requisito que las pruebas deban modificarse, a menos que sea puntuales de cada proyecto.
- **Completas:** Deben abarcar la mayor cantidad de código.
- **Reutilizables:** Lo ideal es no tener pruebas que solo puedan ser ejecutadas una vez.
- **Independientes:** La ejecución de una prueba no debe afectar jamás al procedimiento de otra.
- **Profesionales:** Debe ser tratadas con el mismo profesionalismo y buenas prácticas que se escribe el código.





CARACTERÍSTICAS DE LAS PRUEBAS UNITARIAS

VENTAJAS

- **Fomentan el cambio:** Facilitan que el programador pueda revisar el código y realizar los cambios correspondientes antes de dar por finalizado un módulo.
- **Simplifica la integración:** Pruebas aprobadas se traduce en una mayor seguridad a la hora de entregar el código.
- **Documenta el código:** Las propias pruebas son una especie de documentación, al leerlas es posible entender el funcionamiento de un código.

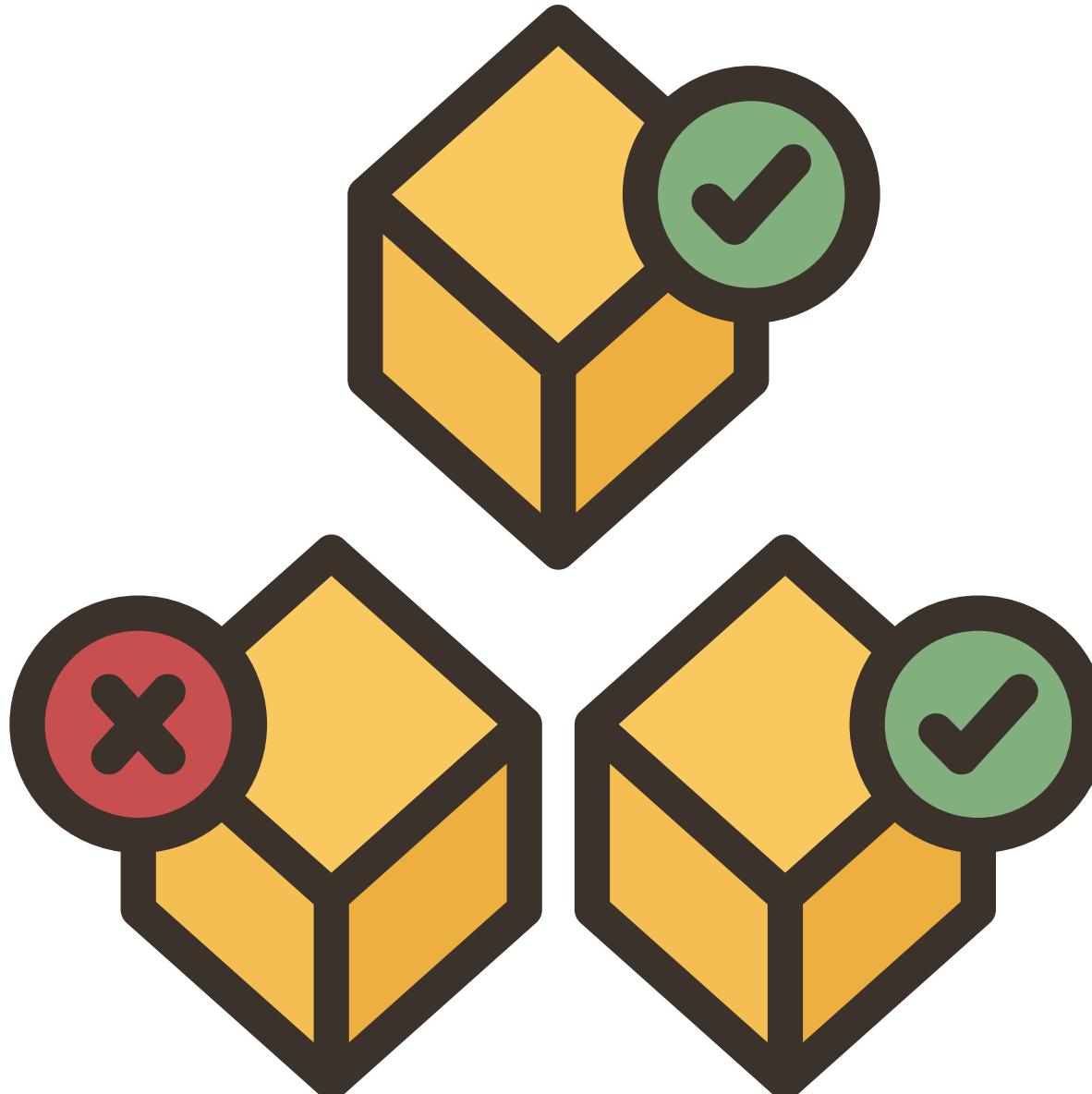




CARACTERÍSTICAS DE LAS PRUEBAS UNITARIAS

VENTAJAS

- **Separación de la interfaz y la implementación:** Dado que la interfaz no es parte del comportamiento lógico del código, es posible eliminarlas y/o modificarlas sin afectar a las pruebas unitarias, y viceversa.
- **Los errores son acotados y fáciles de localizar:** Al tener pruebas segmentadas y bien organizadas es mucho más fácil saber en qué punto nuestro código está presentando errores.





CARACTERÍSTICAS DE LAS PRUEBAS UNITARIAS

LIMITACIONES

- Es importante tener en cuenta que las pruebas unitarias no van a poder descubrir todos los errores que se puedan presentar en un código.
- Problemas de rendimiento, de integraciones externas o conjuntos que presenten un inconveniente mayor normalmente van más allá de una prueba unitaria.
- **¿Entonces realmente son efectivas?** Si, porque hablamos de una primera línea que puede atacar a varias problemáticas triviales o de lógica que ya se está esperando.





INTRODUCCIÓN A JUNIT

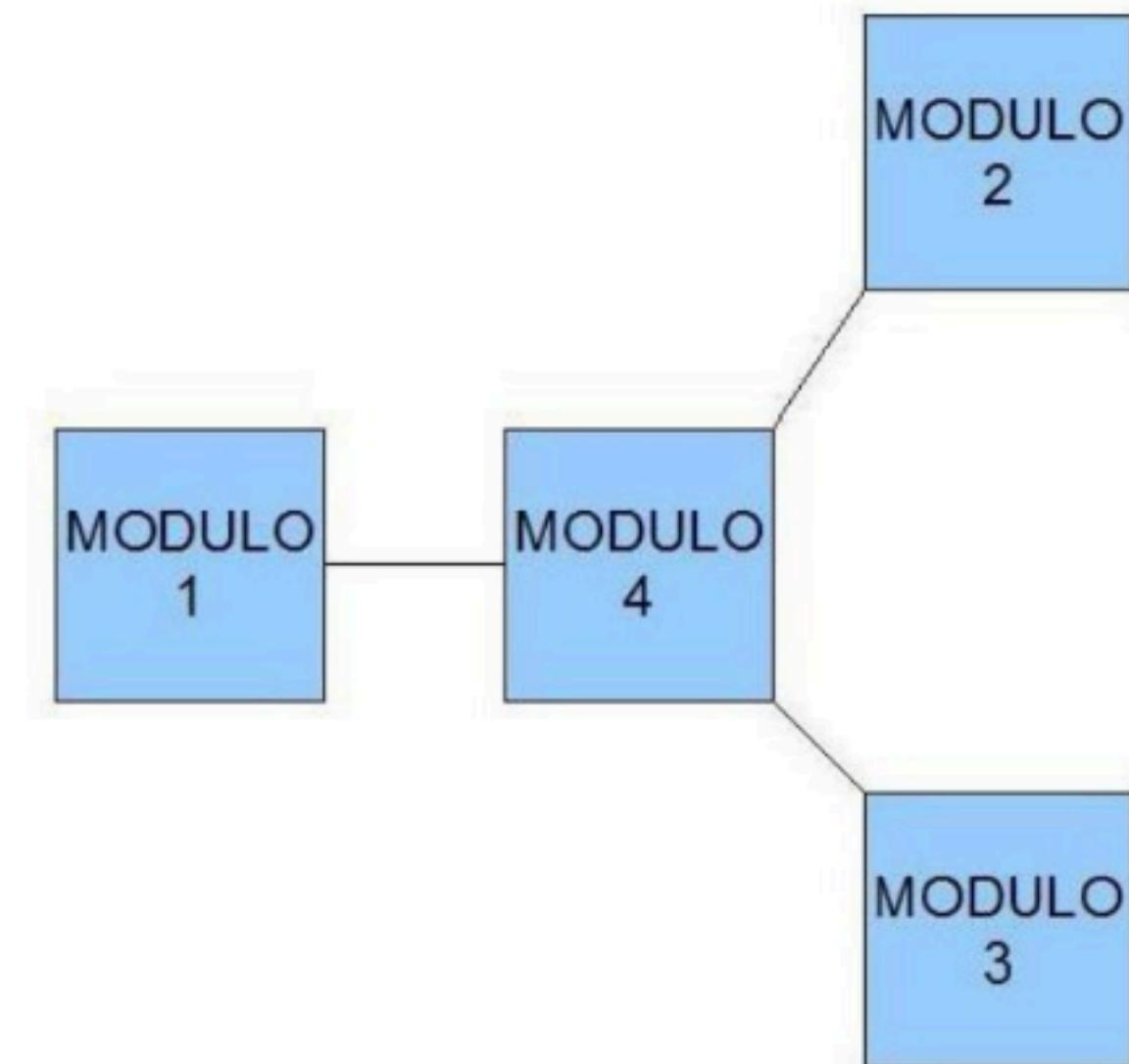
- Es una librería OpenSource.
- Fue desarrollada para probar el funcionamiento de las clases y métodos que componen una aplicación.
- Los casos de prueba de JUnit son módulos que disponen de diferentes métodos para testear una clase y su funcionamiento concreto.
- Mediante una Suite de JUnit se agrupan sus módulos, construyendo así un programa que puede aplicar pruebas sobre el software.
- Es importante tener en cuenta que cuando aplicamos un cambio en un módulo, las pruebas deben volver a ejecutarse.





¿POR QUÉ APLICAR TEST?

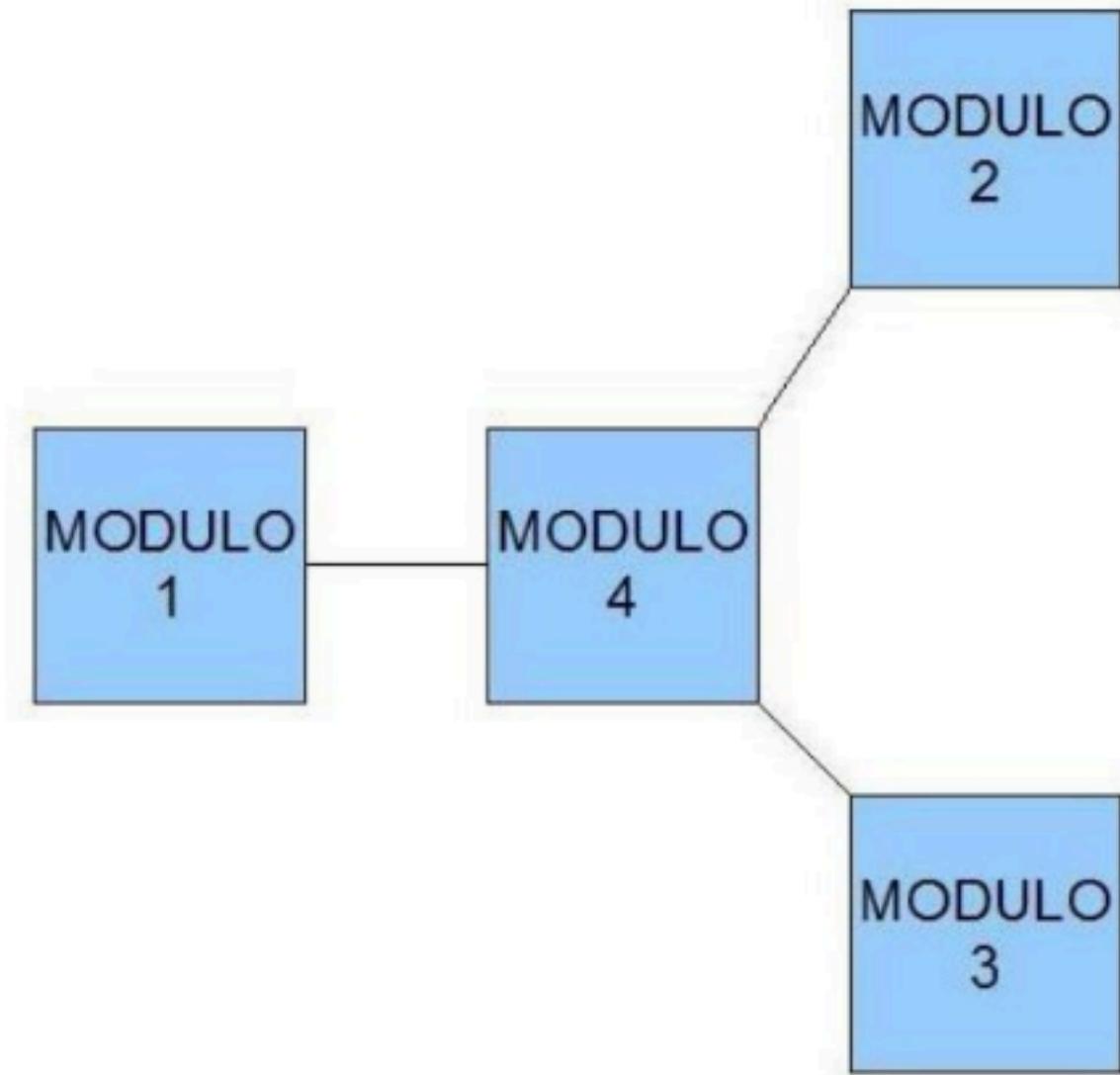
- Más allá de las ventajas que ya conocemos que nos entrega una prueba unitaria, vamos a pensar en una problemática real.
- Normalmente en un equipo nos vamos a encontrar con varios programadores trabajando simultáneamente sobre un mismo proyecto.
- Obviamente cada uno debería estar escribiendo o modificando código en un módulo independiente, pero estos suelen estar relacionados de alguna forma.





¿POR QUÉ APLICAR TEST?

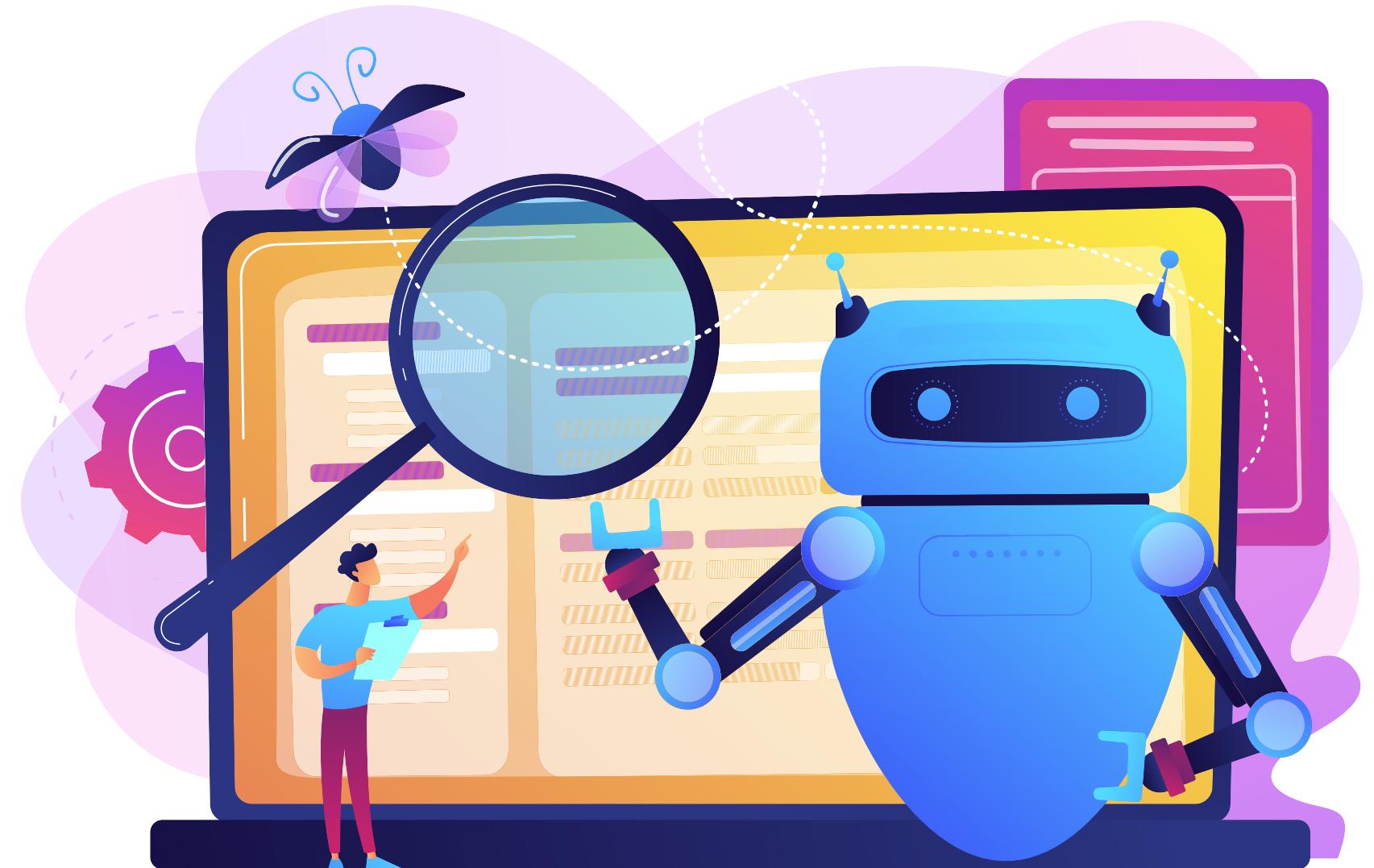
- En un principio tenemos funcionando perfectamente el módulo 1 y 4.
- ¿Qué pasa si ahora queremos agregar nuevos módulos? (2 y 3)
- Se desarrollan e integran perfectamente al módulo 4, a simple vista y con las nuevas pruebas creadas todo funciona... PERO... tuvimos que hacer algunos cambios al módulo 4 para que las nuevas integraciones funcionen.
- **¿Eso nos asegura que el módulo 1 esté funcionando?**
Pues *no*, y lo ideal sería tener pruebas automatizadas sobre él para comprobar si sigue funcionando correctamente o es necesario también adaptarlo a los cambios generales.





CARACTERÍSTICAS DE JUNIT 5

- A diferencia de sus versiones anteriores, se compone de tres subproyectos diferentes.
- **JUnit Platform:** Sirve de base para marcos de prueba en JVM (Java Virtual Machine).
- **JUnit Vintage:** Proporciona un TestEngine para ejecutar pruebas basadas en JUnit 3 y JUnit 4.
- **JUnit Jupiter:** Incluye nuevas anotaciones que no se encontraban en su versión pasada JUnit 4.





CARACTERÍSTICAS DE JUNIT 5

ANOTACIONES MÁS UTILIZADAS

- **@Test:** Le indicamos a JUnit que el método que tengamos bajo la anotación será testeado.
- **@BeforeEach:** Quiere decir que el método que la posea se va a ejecutar antes de cada método de prueba en la clase. Su análogo anterior era *@Before*
- **@BeforeAll:** Se ejecutará una vez antes de todos los métodos de prueba que tenga la clase. Importante tener en cuenta que el método debe ser estático. Su análogo anterior era *@BeforeClass*

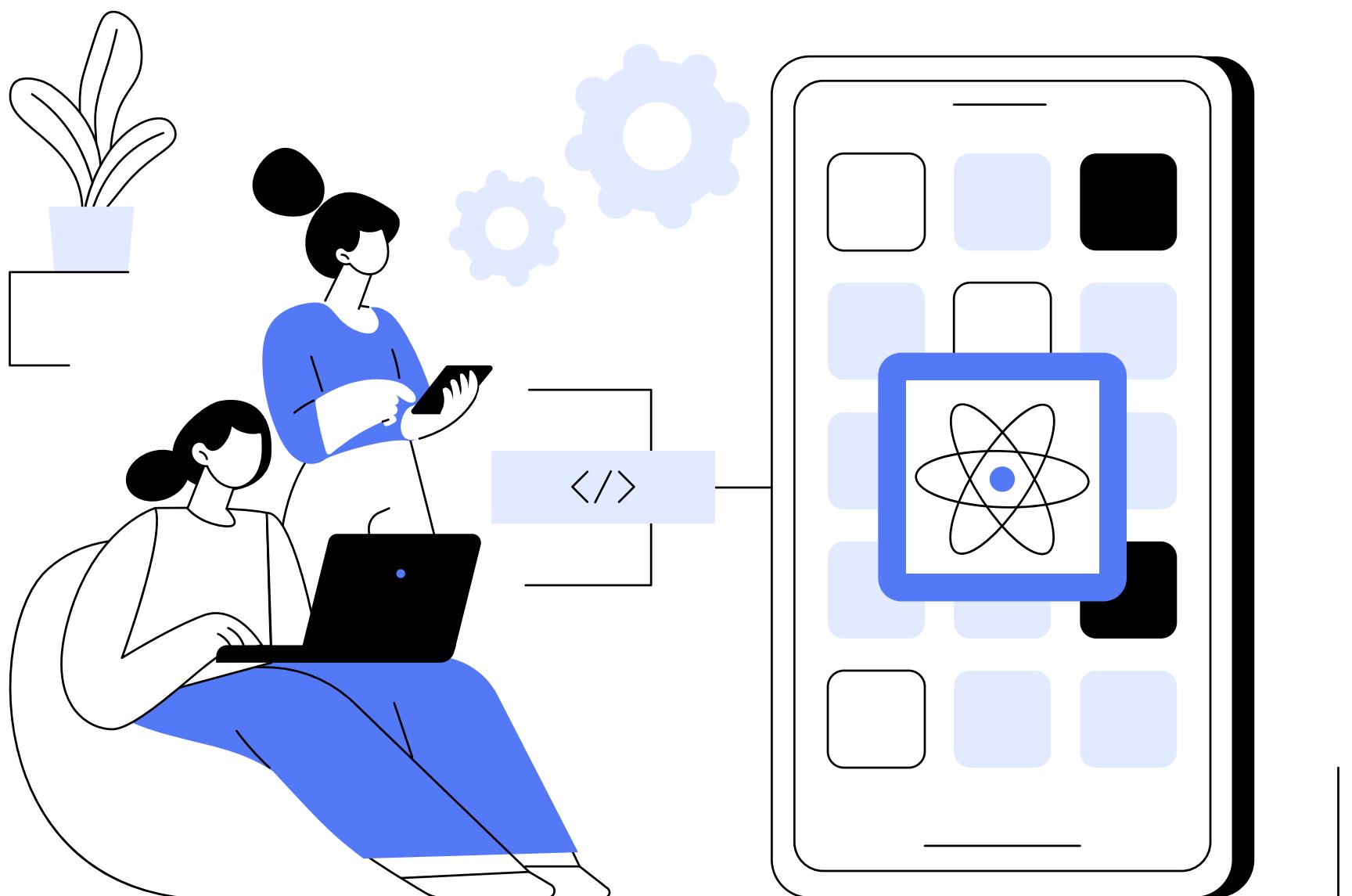




CARACTERÍSTICAS DE JUNIT 5

ANOTACIONES MÁS UTILIZADAS

- **@AfterEach:** Se ejecuta después de cada método de prueba. Su análogo era @After
- **@AfterAll:** Se ejecuta luego de que todos los métodos se hayan ejecutado. Es importante tener en cuenta que su método debe ser estático. Su análogo era @AfterClass

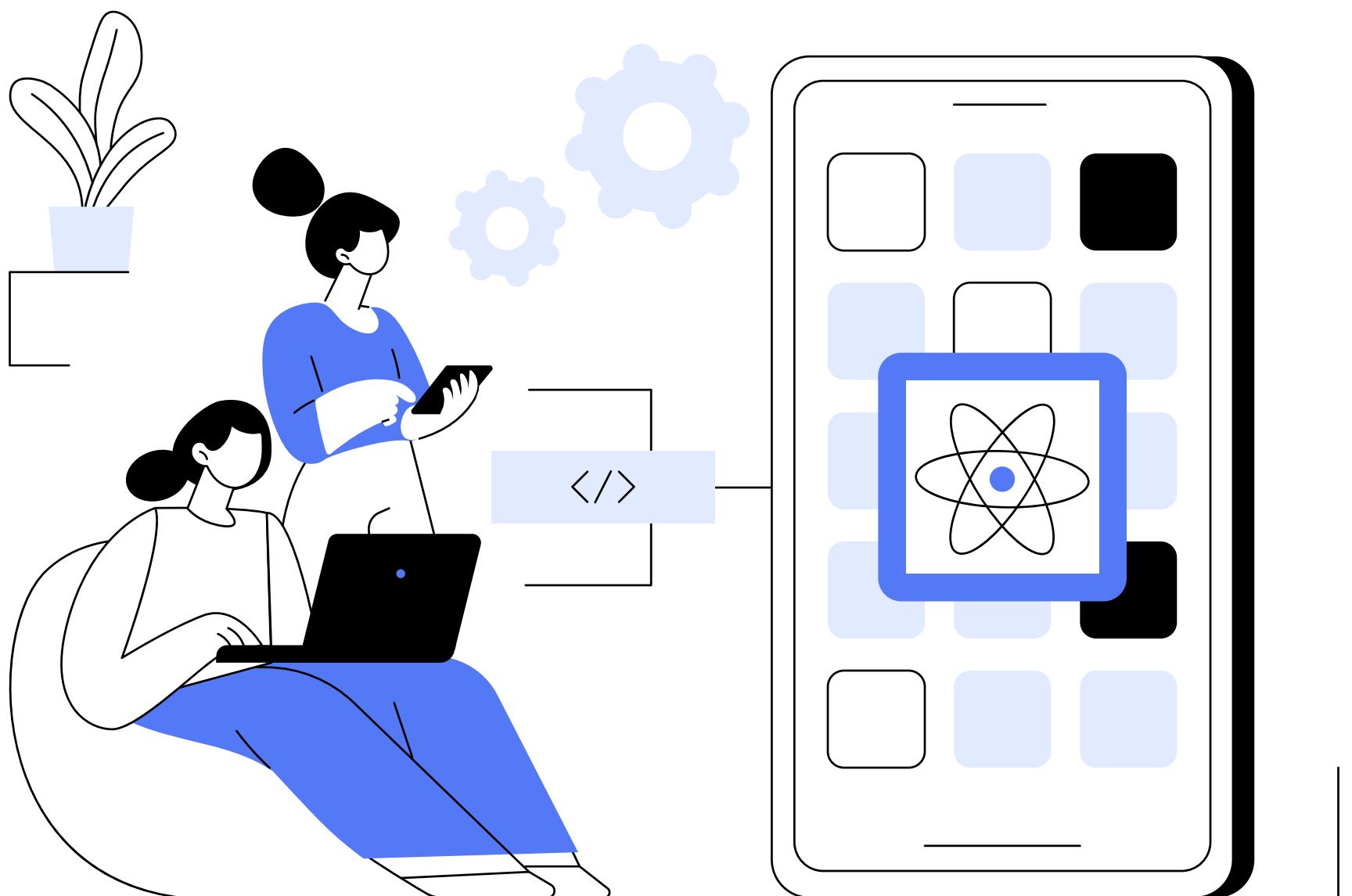




CARACTERÍSTICAS DE JUNIT 5

ANOTACIONES MÁS UTILIZADAS

- **@AfterEach:** Se ejecuta después de cada método de prueba. Su análogo era @After
- **@AfterAll:** Se ejecuta luego de que todos los métodos se hayan ejecutado. Es importante tener en cuenta que su método debe ser estático. Su análogo era @AfterClass





CARACTERÍSTICAS DE JUNIT 5

ANOTACIONES MÁS UTILIZADAS

- **Assertions:** Todas las afirmaciones en JUnit son métodos estáticos en la clase. Las más comunes a utilizar son `assertTrue()` y `assertEquals()`
- **Assumptions:** Las suposiciones se utilizan para ejecutar pruebas solo si se cumplen ciertas condiciones. Entre ellas las más comunes que podemos encontrar son `assumeTrue()` y `assumeFalse()`
- **Exceptions:** Se implementan a través del método `assertThrows()`





AWAKELAB

#programmingbootcamp

nodovirtual.awakelab.cl

 jELOU futurejob by  Chile