Decidimos realizarle pruebas a la clase ControlCarrera

Evidencia del Dataset con las descripciones de cada parte, la entrada y el resultado esperado

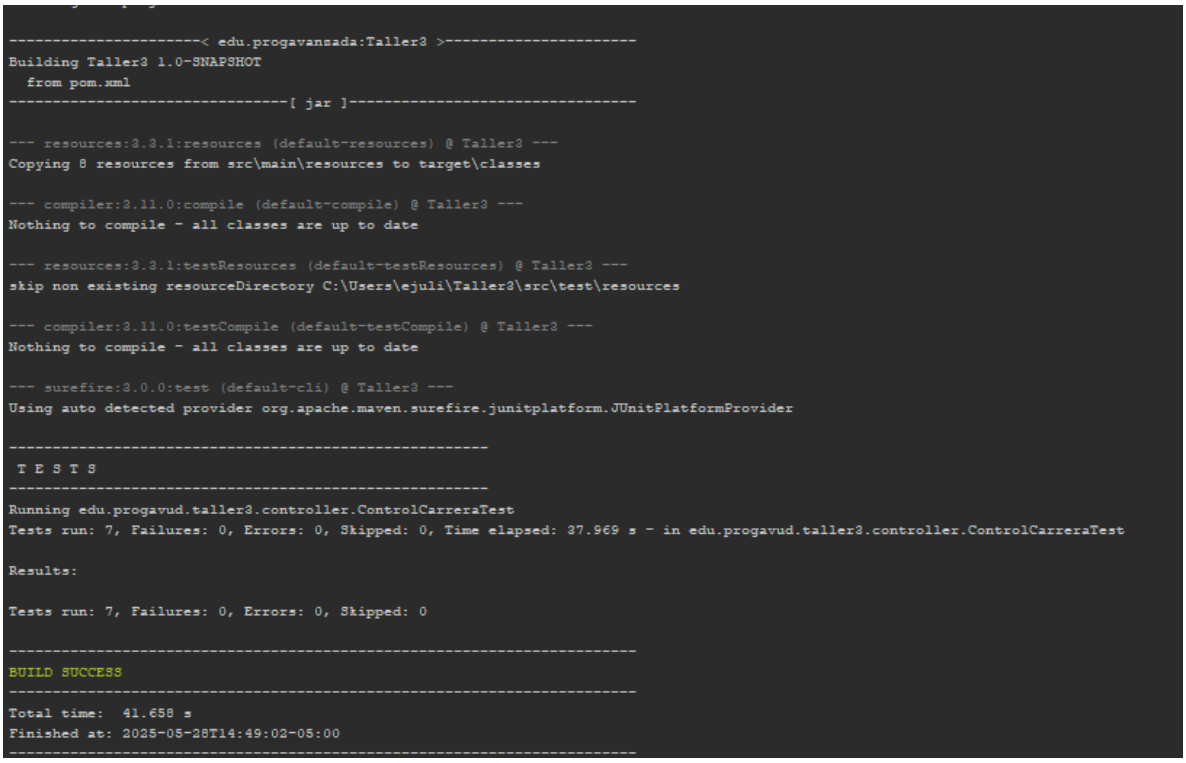
Descripción Test	Entrada	Resultado Esperado
Definir ganador único	• 3 competidores:	• Mensaje que contenga "Competidor2" y "1.2"
	- Competidor1, tiempo=1500ms	• No mencione "empate"
	- Competidor2, tiempo=1200ms	• Competidor2.getVictorias()==1
	- Competidor3, tiempo=1800ms	• Competidor1.getVictorias()==0, Competidor3.getVictorias()==0
Definir ganador con empate	• 3 competidores:	• Mensaje que contenga "empate", "CompetidorA", "CompetidorB" y "1000"
	- CompetidorA, tiempo=1000ms	• No incluya "CompetidorC"
	- CompetidorB, tiempo=1000ms	• CompetidorA.getVictorias()==1
	- CompetidorC, tiempo=1500ms	• CompetidorB.getVictorias()==1
		• CompetidorC.getVictorias()==0
Definir ganador absoluto	• 3 competidores:	• Mensaje que contenga:
	- Veterano con 3 victorias	- "Veterano" y "ganador Absoluto es: Veterano"
	- Novato con 1 victoria	- "Victorias: 3", "Victorias: 1", "Victorias: 0"
	- Principiante con 0 victorias	
Definir ganador absoluto con empate	• 3 competidores:	• Mensaje que contenga "empate", "EmpateA", "EmpateB" y "Victorias: 2"
	- EmpateA con 2 victorias	• No mencione "Perdedor" como ganador
	- EmpateB con 2 victorias	
	- Perdedor con 1 victoria	
Obtener distancia de carrera	— (no necesita lista de competidores)	• Retorna 100
Formato de mensaje de accidente	• 3 competidores en lista interna	Para cada llamada:
		• Empiece con "El competidor " y termine con " se accidento"
		• Número entre 1 y 3 (inclusive)
Formato de mensaje de impulso	• 2 competidores en lista interna	Para cada llamada:
		• Empiece con "El competidor: " y termine con " se impulso"
		• Número entre 1 y 2 (inclusive)

El siguiente análisis presenta los resultados de las pruebas unitarias diseñadas específicamente para evaluar los métodos de la clase ControlCarrera. Cada caso de prueba contempla:

- **Descripción del Test**: Indica el método de ControlCarrera que está siendo evaluado.
- **Entrada**: Valores concretos utilizados para la ejecución del método.

- **Resultado Esperado**: Comportamiento o salida que se espera del sistema una vez ejecutada la prueba.

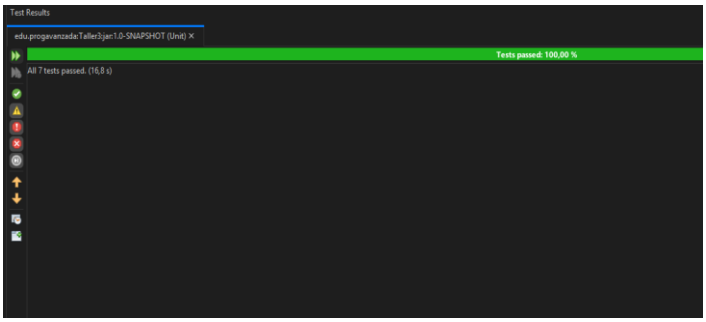
Imagen 1: Consola de construcción y prueba



En la consola se observa que:

- La **compilación del proyecto fue exitosa** (BUILD SUCCESS).
- Se **ejecutaron 7 pruebas unitarias** correspondientes a los distintos métodos de ControlCarreraTest.
- **Todas las pruebas pasaron exitosamente**, sin errores ni fallos (Failures: 0, Errors: 0).
- El **tiempo total de ejecución** fue de aproximadamente 41.658 segundos.
- Se incluye la **fecha y hora exacta** de ejecución, lo cual garantiza la trazabilidad del proceso de validación.

Imagen 2: Panel visual de resultados de pruebas



En la pestaña **Test Results** del IDE se confirma gráficamente que **el 100% de las pruebas fueron superadas con éxito** (7/7). Entre los métodos validados se encuentran:

- DefinirGanadorUnico()
- DefinirGanadorConEmpate()
- DefinirGanadorAbsoluto()
- DefinirGanadorAbsolutoConEmpate()
- GetDistanciaCarrera()
- AccidenteCompetidorFormato()
- ImpulsarCompetidorFormato()

Estos resultados respaldan el correcto funcionamiento del controlador ControlCarrera bajo múltiples escenarios de uso y validación.

Evidencia del uso de anotaciones de ciclo de vida en ControlCarreraTest

En esta clase de prueba se aprovechan las capacidades de JUnit 5 para organizar la configuración y garantizar que cada escenario de prueba parta de un estado controlado y reproducible. A continuación, se describen las principales anotaciones usadas:

- @BeforeEach
  El método setUp() anotado con @BeforeEach invoca internamente a crearControlCarreraParaPruebas(), una fábrica especializada que:
  1. Crea una subclase anónima de ControlCarrera en la que se vacían y asignan las listas internas (competidores y competidoresHilos) para evitar dependencias de UI o hilos reales.
  2. Fija por reflexión la propiedad distanciaCarrera a 100.

3. Sobrescribe moverCompetidorLabel(...) para que no realice ninguna operación gráfica.
   De este modo, antes de cada test disponemos de un objeto controlCarrera limpio y listo para recibir inyección de datos mediante reflexión.

- Ausencia de @BeforeAll y @AfterAll
  Debido a que cada prueba trabaja con un nuevo objeto de ControlCarrera y los datos se reinicializan en @BeforeEach, no es necesario compartir estado global ni configurar recursos costosos (como conexiones o datos persistentes) una sola vez. Por ello no se usan @BeforeAll ni @AfterAll.
- Ausencia de @AfterEach
  No se requiere limpieza explícita tras cada prueba: como el objeto controlCarrera se crea de cero en @BeforeEach y no abre recursos externos (archivos, sockets o hilos en ejecución real), basta con que el garbage collector libere la instancia al terminar cada método de test.