







Desafío 15


Desafío 15


 **Evaluador de Entregas de Programación**

 **Estudiante:** MILAGROS TERESITA POZZO FASINI


 **Centro:** CeRP del Suroeste

 **Curso:** PROGRAMACIÓN II: 02INPGR22025215525 Anual Semipresencial


 **Año:** 2025

 **Desafío 15**

Simula una carrera de Vehículo (subclases Auto y Moto) en la que cada vehículo avanza una distancia aleatoria en cada turno. Usa un arreglo polimórfico para los competidores y determina el ganador al final de 5 rondas.

 **Tu código de solución:**

Escribe tu código aquí...

 **Fundamentación de tu solución:**

Explica tu estrategia, qué pensaste para resolver el problema, etc.

Todos los vehículos de la carrera (autos y motos) tienen cosas en común: un **nombre** y una **distancia recorrida**. Por eso hice una clase Vehículo que guarda esos datos y que tiene un método abstracto avanzar(), que obliga a cada subclase a definir cómo se mueve. Después creé dos clases concretas: Auto y Moto. Cada una tiene su propia forma de avanzar: el auto avanza entre 5 y 15 metros, y la moto entre 3 y 12 metros, elegidos al azar en cada turno.

Para organizar la carrera, en el main armé un **arreglo polimórfico** con los competidores, es decir, un mismo arreglo que puede guardar autos y motos porque todos son del tipo Vehículo. Luego, hice un bucle de 5 rondas, como pide el enunciado. En cada ronda, cada vehículo avanza una distancia aleatoria (llamando a su propio avanzar()), se va sumando su total y se muestra por pantalla.

Al final de las 5 rondas, el programa busca qué vehículo recorrió más distancia. Si hay empate, muestra a todos los empatados. Si no, anuncia un único ganador. Esto hace que la simulación sea clara y que se vea bien cómo funciona el **polimorfismo**: aunque todos están en un mismo arreglo, cada uno ejecuta su propia versión del método avanzar().



Clase base `Vehiculo` con lo común (un nombre identificador y la distancia total recorrida) y un método abstracto `avanzar()` para obligar a que cada tipo concrete su forma de moverse; de ese modo, `Auto` y `Moto` heredan esos datos y solo se concentran en “cómo avanzan” en cada turno. Elegí usar números aleatorios con `Random` porque la consigna habla de una carrera con avance variable; para mantenerlo simple en nivel inicial, fijé rangos enteros fáciles de entender (por ejemplo, el auto entre 5 y 15 metros y la moto entre 3 y 12), y cada llamada a `avanzar()` devuelve cuánto avanzó en esa ronda y lo suma al acumulado. Para organizar la competencia quise explotar el polimorfismo: construí un arreglo de tipo `Vehiculo[]` donde guardo indistintamente autos y motos, y en el bucle de 5 rondas recorro ese arreglo llamando al mismo método `avanzar()` sin preguntar “qué es” cada objeto; en tiempo de ejecución, Java resuelve si debe ejecutar la versión de `Auto` o la de `Moto`. Mientras itero, voy mostrando por pantalla cuánto avanzó cada competidor y su total, lo que ayuda a seguir la carrera paso a paso. Al finalizar las 5 rondas, calculo la mayor distancia recorrida y después recorro otra vez a los competidores para identificar al ganador; si varios alcanzan la misma distancia máxima, los anuncio como empate para cubrir ese caso sin lógica complicada. Esta estructura mantiene el código claro y repetible: el estado de cada vehículo vive en su propio objeto, el flujo principal solo coordina las rondas y la comparación final, y el uso de una clase base con un método abstracto simplifica agregar nuevos tipos (por ejemplo, un “Camión”) sin tocar el motor de la carrera

Enlace a GitHub [https://github.com/MilagrosPozzo/Programacion-2/blob/main/U3 Lenguaje de Programacion Java/Unidad 3 Desaf%C3%ADo 15.java](https://github.com/MilagrosPozzo/Programacion-2/blob/main/U3%20Lenguaje%20de%20Programacion%20Java/Unidad%203%20Desaf%C3%ADo%2015.java)