

	<p align="center">UNIVERSIDAD NACIONAL DE SAN AGUSTIN FACULTAD DE INGENIERÍA DE PRODUCCIÓN Y SERVICIOS ESCUELA PROFESIONAL DE INGENIERÍA DE SISTEMA</p>	
Formato: Guía de Práctica de Laboratorio / Talleres / Centros de Simulación		
Aprobación: 2022/03/01	Código: GUIA-PRLE-001	Página: 1








INFORME DE LABORATORIO

(formato estudiante)

INFORMACIÓN BÁSICA					
ASIGNATURA:	Tecnología de Objetos				
TÍTULO DE LA PRÁCTICA:	<i>La Evolución de la Programación Orientada a Objetos: Más que Clases y Objetos</i>				
NÚMERO DE PRÁCTICA:	01	AÑO LECTIVO:	2025	NRO. SEMESTRE:	VI
FECHA DE PRESENTACIÓN		HORA DE PRESENTACIÓN			
INTEGRANTE (s): Soncco Mamani, Max Junior Vera Zapacayo, Jhoer Luis Velasque Arcos, Mikhail Gabino				NOTA:	
DOCENTE(s): Ing. EDITH GIOVANNA CANO MAMANI					

SOLUCIÓN Y RESULTADOS
<p>I. SOLUCIÓN DE EJERCICIOS/PROBLEMAS</p> <p>1. Fase 1 – Investigación histórica</p> <p>Simula 67 fue creado en 1967 por Ole-Johan Dahl y Kristen Nygaard para resolver problemas de simulación de sistemas complejos, introduciendo por primera vez clases y objetos. Por su parte, Smalltalk surgió en los años setenta en Xerox PARC bajo la dirección de Alan Kay, con el propósito de enseñar conceptos computacionales y construir entornos gráficos interactivos. El principal aporte de Simula fue sentar las bases de la programación orientada a objetos mediante herencia y abstracción. Smalltalk llevó esas ideas más lejos al ser el primer lenguaje totalmente orientado a objetos donde todo es un objeto. Entre las limitaciones, Simula nunca tuvo gran adopción práctica y Smalltalk fue criticado por su lentitud y baja popularidad frente a lenguajes más comerciales. Sin embargo, ambos lenguajes marcaron la historia al influir directamente en Java, Python, Ruby y otros modernos.</p>

2. Fase 2 – Línea del tiempo crítica

AÑO	Lenguaje	Inventor(es)	Breve resumen	Icono visual	Pregunta crítica
1967	Simula 67	Ole-Johan Dahl y Kristen Nygaard	Primer lenguaje en implementar de forma completa el paradigma orientado a objetos.		¿Por qué se considera el primer lenguaje orientado a objetos?
1972	Smalltalk	Alan Kay, Dan Ingalls, Adele Goldberg	Introdujo el concepto 'todo es un objeto'.		¿Qué hizo especial a Smalltalk frente a otros lenguajes de su época?
1983	C++	Bjarne Stroustrup	Extiende el lenguaje C con clases y POO.		¿Por qué C++ es útil para programas grandes?
1995	Java	James Gosling y equipo Sun Microsystems	Lenguaje multiplataforma		¿Por qué Java funciona en distintos sistemas sin cambiar el código?
1991	Python	Guido van Rossum	Sintaxis simple y versatilidad.		¿Qué hace a Python fácil de aprender?
1995	Ruby	Yukihiro Matsumoto	Base de frameworks como Ruby on Rails para desarrollo web rápido		¿Por qué a los desarrolladores les gusta Ruby para páginas web?
2011	Kotlin	JetBrains	Lenguaje moderno interoperable con Java		¿Por qué Kotlin es bueno para hacer apps de Android?

	<p style="text-align: center;">UNIVERSIDAD NACIONAL DE SAN AGUSTIN FACULTAD DE INGENIERÍA DE PRODUCCIÓN Y SERVICIOS ESCUELA PROFESIONAL DE INGENIERÍA DE SISTEMA</p>	
<p style="text-align: center;">Formato: Guía de Práctica de Laboratorio / Talleres / Centros de Simulación</p>		
<p>Aprobación: 2022/03/01</p>	<p>Código: GUIA-PRLE-001</p>	<p>Página: 3</p>

En esta línea del tiempo se revela cómo los lenguajes de programación fueron evolucionando desde Simula 67 que inició en la orientación a objetos, hasta modernos como Kotlin orientados a proactividad y seguridad. Smalltalk consolidó la idea de que todo es objeto, marcando para entornos gráficos interactivos. C++ combina la eficiencia de C con la POO, convirtiéndose en un estándar para software de alto rendimiento. Java llevó la portabilidad a otro nivel con su multiplataforma, mientras Python y Ruby destacaron por su simplicidad y productividad. En la última década, lenguajes mixtos como Kotlin, Swift y Rust surgieron para equilibrar seguridad, rendimiento y facilidad de desarrollo. Esta progresión muestra una tendencia clara, cada nueva generación hereda conceptos de la anterior, pero busca resolver sus limitaciones, impulsando la innovación continua en el ecosistema de software.

3. Fase 3 – Debate guiado



Como grupo, estamos a favor de la programación orientada a objetos (POO), consideramos que Simula 67 y Smalltalk fueron esenciales para organizar programas complejos y modelar el mundo real en el software. Sin su aporte, lenguajes como Java, C++ o Python no habrían alcanzado la solidez que hoy tienen en la industria. Aunque es cierto que Smalltalk no tuvo gran éxito comercial y que hoy existen paradigmas alternativos como el funcional, su valor histórico e intelectual es indiscutible. La POO permitió pensar en términos de clases y objetos, una forma más natural y cercana a la realidad que simplificó la construcción de sistemas grandes. Además, los conceptos de herencia, encapsulamiento y polimorfismo siguen siendo la base en la mayoría de lenguajes modernos. Por ello, nuestra postura es que la POO fue y sigue siendo un pilar fundamental en la evolución del software, aun cuando conviva con otros paradigmas.

4. Fase 4 – Cierre (ensayo personal)

¿Es la programación orientada a objetos el mejor paradigma hoy en día?

Max: En mi opinión, la programación orientada a objetos sigue siendo muy importante, pero no creo que sea el mejor paradigma en todos los casos. Lenguajes como Java y C++ muestran su valor en proyectos grandes y complejos. Sin embargo, lenguajes como Python mezclan POO con programación funcional, lo que demuestra que se necesita más flexibilidad. En áreas como la inteligencia artificial, la programación funcional ofrece ventajas claras en el manejo de datos. Según la documentación de Oracle, hoy la industria apuesta por combinar paradigmas en lugar de usar solo uno. Por eso pienso que la POO es esencial, pero no exclusiva.

Jhoer: Yo considero que la programación orientada a objetos ya no es suficiente como el paradigma principal. Aunque Java y C# todavía se usan bastante, lenguajes como JavaScript y

	<p style="text-align: center;">UNIVERSIDAD NACIONAL DE SAN AGUSTIN FACULTAD DE INGENIERÍA DE PRODUCCIÓN Y SERVICIOS ESCUELA PROFESIONAL DE INGENIERÍA DE SISTEMA</p>	
<p style="text-align: center;">Formato: Guía de Práctica de Laboratorio / Talleres / Centros de Simulación</p>		
<p>Aprobación: 2022/03/01</p>	<p>Código: GUIA-PRLE-001</p>	<p>Página: 4</p>

Rust muestran formas diferentes y más prácticas de programar. Muchas veces la POO complica demasiado los proyectos más simples. En aplicaciones web modernas, los frameworks trabajan con estilos reactivos y declarativos que resultan más eficientes. Artículos de IEEE resaltan que hoy lo importante es mezclar distintos paradigmas según la necesidad. Por eso concluyo que la POO sigue siendo útil, pero ya no es la mejor opción actual.

Mikhail: Desde mi punto de vista, la programación orientada a objetos sigue siendo el mejor paradigma en la mayoría de contextos. Lenguajes como Java, Python y C++ lo demuestran, ya que son muy usados en la industria y en la enseñanza. Su forma de representar problemas con clases y objetos es clara y muy cercana a la vida real. Aunque existen otros enfoques, casi siempre se usan como complemento de la POO. Tanto Python.org como Oracle señalan que este paradigma continúa siendo el más difundido en todo el mundo. Por eso mi postura es que la POO mantiene su liderazgo.

RETROALIMENTACIÓN GENERAL

REFERENCIAS Y BIBLIOGRAFÍA

- <https://www.stroustrup.com/whatis.pdf>
- <https://www.youtube.com/watch?v=aYT2se94eU0&t=626s>
- <https://www.quora.com/What-did-Alan-Kay-mean-by-I-made-up-the-term-object-oriented-and-I-can-tell-you-I-did-not-have-C++-in-mind>
- https://docs.google.com/presentation/d/1y_IO5O0i5SPFWD8dQjhZ8m_uVDzTR98NzMZyTJ5aiHQ/edit?usp=sharing