# Verificación Orientada a Objetos: Contratos, Invariantes y Modularidad

Brayan Stiven Ramos

Zeneida Ramos portocarrero

Diana Ximena Ramos

# CONTENIDO


1. INTRODUCCIÓN

2.¿POR QUÉ ES IMPORTANTE?

3. OBJETIVOS

4. CONCEPTOS BÁSICOS

5. CARACTERÍSTICAS PRINCIPALES

6. VENTAJAS Y DESVENTAJAS

7 IMPORTANCIA Y APLICACIONES

8. CONCLUSIÓN

# INTRODUCCIÓN

- Este trabajo aborda la verificación orientada a objetos, un conjunto de técnicas y principios que permiten asegurar que los sistemas construidos con programación orientada a objetos funcionen correctamente, basándose en tres pilares fundamentales: contratos, invariantes y modularidad.





# ¿POR QUÉ ES IMPORTANTE?



**Garantiza confianza:** Asegura que el software se comporte como se espera, evitando fallos que pueden afectar a personas, empresas o sistemas críticos.

**Facilita el trabajo en equipo**: Al tener reglas claras (contratos), cada programador sabe qué esperar de cada parte sin confusión.

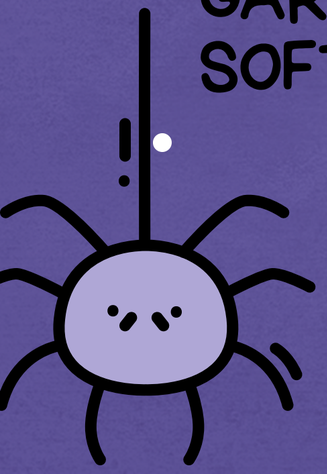
**Ahorra tiempo y dinero:** Detecta errores desde el inicio; corregir un fallo al principio cuesta mucho menos que cuando el sistema ya está en uso.

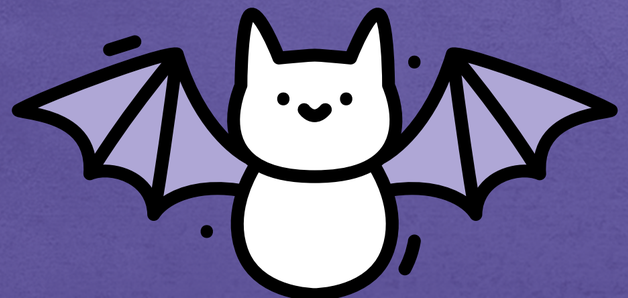
**Facilita el crecimiento del proyecto**: Gracias a la modularidad, se pueden agregar mejoras o cambiar partes sin desarmar todo el sistema.

**Es la base de los sistemas actuales**: La mayoría de los programas que usamos hoy se construyen con orientación a objetos; verificar bien su funcionamiento los hace seguros y duraderos.

# OBJETIVOS

- OBJETIVO GENERAL
- COMPRENDER Y EXPLICAR QUÉ ES LA VERIFICACIÓN ORIENTADA A OBJETOS, ASÍ COMO EL FUNCIONAMIENTO, LA UTILIDAD Y LA RELACIÓN ENTRE CONTRATOS, INVARIANTES Y MODULARIDAD PARA GARANTIZAR LA CALIDAD DEL SOFTWARE.





- OBJETIVOS ESPECÍFICOS
- DEFINIR LOS CONCEPTOS DE CONTRATO, INVARIANTE Y MODULARIDAD, Y SU PAPEL EN LA VERIFICACIÓN DE SOFTWARE.
- IDENTIFICAR LAS CARACTERÍSTICAS Y VENTAJAS DE APLICAR ESTOS PRINCIPIOS EN EL DESARROLLO DE SISTEMAS.
- MOSTRAR UN EJEMPLO PRÁCTICO DE APLICACIÓN Y DESCRIBIR EN QUÉ ÁREAS SE UTILIZAN ESTAS TÉCNICAS.

# CONCEPTOS BÁSICOS

| Concepto | Definición |
| --- | --- |
| **Verificación orientada a objetos** | Proceso de comprobar que un sistema cumple sus requisitos, aplicando métodos adaptados a la estructura de clases, objetos y herencia. |
| **Contrato** | Acuerdo entre un método y quien lo usa: establece condiciones de entrada (precondiciones), salida (postcondiciones) y garantías de comportamiento. |
| **Invariante** | Propiedad que siempre se mantiene verdadera para un objeto, sin importar qué operaciones se realicen sobre él. |
| **Modularidad** | Organización del sistema en piezas independientes (módulos o clases) que cumplen funciones definidas, con baja dependencia entre sí. |

# EJEMPLO

**Precondición: Lo que debe cumplirse antes**
**Postcondición: Lo que se cumple al terminar EJ: retirar (m)  pre: 0 < m ≤ saldo**

**Precondición: Lo que debe cumplirse antes**
**Definen cuando un objeto esta bien formado EJ: saldo ≥ 0**

**Cada clase se verifica por separado, asumiendo los contratos de las demás**
**Liskov: Subclase con pre más débiles y post más fuertes.**
**Verificación escalable y reutilizable**

# CARACTERÍSTICAS PRINCIPALES



**Basada en confianza y claridad**: Los contratos dejan explícitas las reglas, evitando usos incorrectos del código.
**Consistencia permanente**: Los invariantes protegen el estado de los objetos, impidiendo que lleguen a valores inválidos.
**Separación de responsabilidades**: La modularidad permite que cada parte se diseñe, pruebe y entienda por separado.
**Detección temprana de fallos:** Las violaciones de contrato o invariante se identifican en tiempo de desarrollo o prueba.
**Facilidad de mantenimiento:** Al tener piezas claras y reglas definidas, los cambios afectan menos al resto del sistema.









## Ventajas

- Mayor fiabilidad: Las reglas explícitas reducen comportamientos inesperados.
- Documentación viva: Los contratos explican qué hace cada parte, sirviendo de guía para otros programadores.
- Pruebas más sencillas: Las condiciones claras permiten diseñar casos de prueba de forma directa.





## DESVENTAJAS

- MAYOR TIEMPO INICIAL: ESCRIBIR CONTRATOS E INVARIANTES REQUIERE PLANIFICACIÓN Y ESFUERZO AL INICIO.
- EXCESO DE RESTRICCIONES: SI LAS REGLAS SON DEMASIADO ESTRICTAS, PUEDE DIFICULTAR LA REUTILIZACIÓN DEL CÓDIGO.
- CARGA EN EJECUCIÓN: EN ALGUNOS CASOS, COMPROBAR TODAS LAS CONDICIONES PUEDE HACER EL PROGRAMA UN POCO MÁS LENTO.







# IMPORTANCIA Y APLICACIONES

¿Para qué sirve?
Prevenir errores difíciles de detectar.
Facilitar el trabajo en equipo: cada programador conoce lo que cada módulo ofrece y exige.
Simplificar actualizaciones: se puede mejorar un módulo siempre que siga cumpliendo su contrato.

DÓNDE SE UTILIZA?
SISTEMAS BANCARIOS Y FINANCIEROS.
SOFTWARE MÉDICO O DE CONTROL INDUSTRIAL.
DESARROLLO DE PLATAFORMAS GRANDES (REDES SOCIALES, COMERCIO ELECTRÓNICO).
PROYECTOS QUE DEBEN MANTENERSE Y AMPLIARSE DURANTE MUCHOS AÑOS.



# CONCLUSIÓN







En este recorrido aprendimos que la verificación orientada a objetos no es solo una técnica de revisión, sino una forma de diseñar software con responsabilidad. Los contratos definen acuerdos claros, los invariantes cuidan la integridad de cada objeto y la modularidad permite crecer al sistema sin perder el control. La idea principal es: cuanto más explícitas y claras sean las reglas desde el principio, más sólido, seguro y fácil de mantener será el sistema.

# ¡MUCHAS GRACIAS!



¡Feliz noche de Halloween!