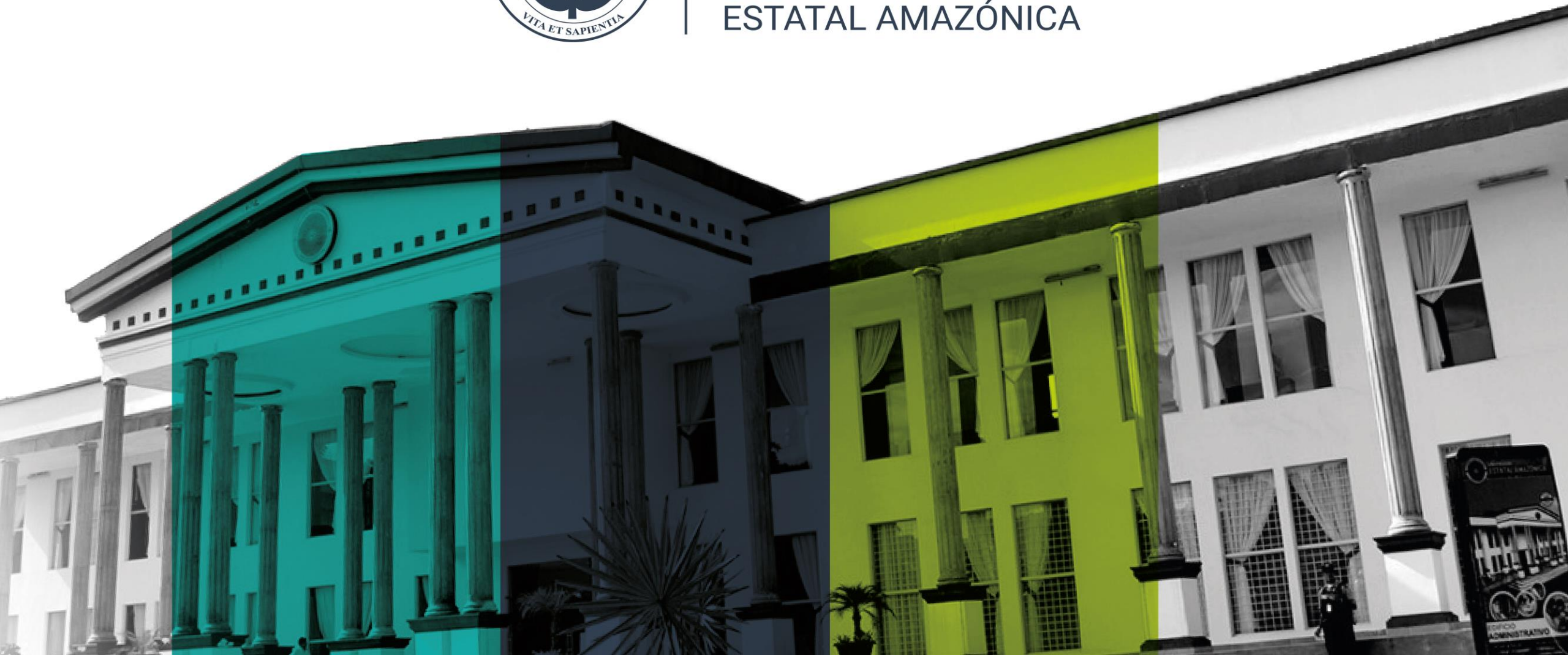
UEA
UNIVERSIDAD
ESTATAL AMAZÓNICA



UEA
UNIVERSIDAD
ESTATAL AMAZÓNICA

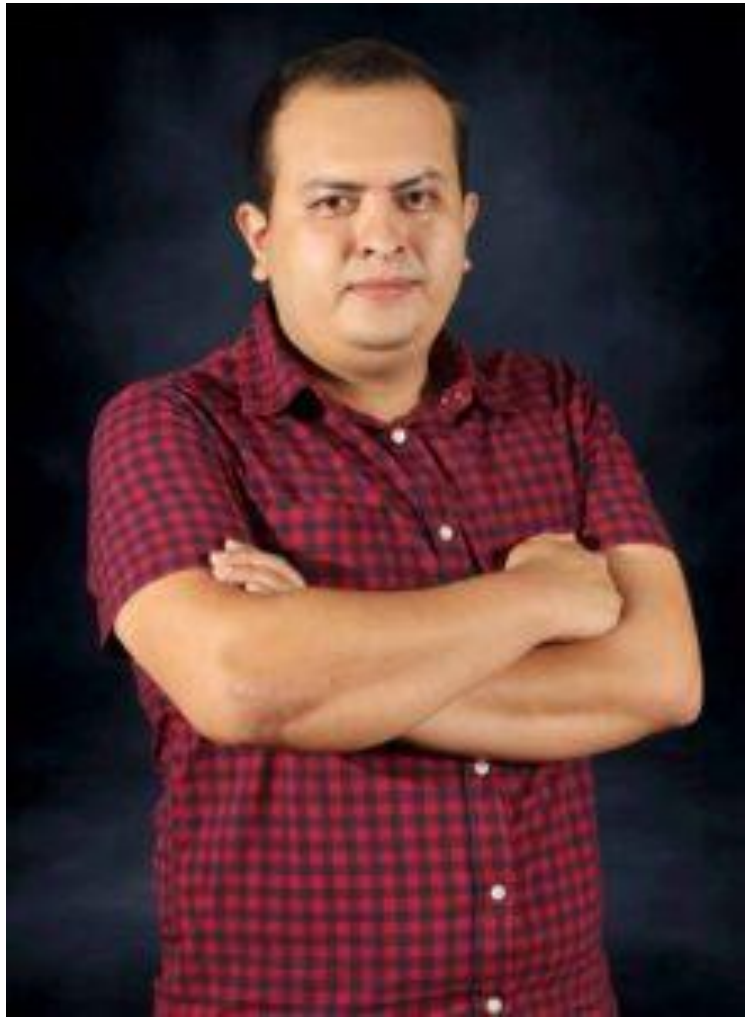
ASIGNATURA

PROGRAMACIÓN ORIENTADA A OBJETOS

Transformamos el mundo desde la Amazonía

Ing. Edwin Gustavo Fernández Sánchez, Mgs.
DOCENTE - PERSONAL ACADÉMICO NO TITULAR OCASIONAL
DIRECTOR DE GESTIÓN DE TECNOLOGÍAS INFORMACIÓN Y COMUNICACIÓN

UNIVERSIDAD ESTATAL AMAZÓNICA



UEA
UNIVERSIDAD
ESTATAL AMAZÓNICA

SEMANA
3

DESARROLLO DE LA SEMANA 3: DEL LUN. 09 AL DOM. 15 DE JUNIO/2025

Resultado de aprendizaje: Comprender los conceptos fundamentales de la programación orientada a objetos (POO) para el desarrollo de soluciones computacionales de complejidad media.

CONTENIDOS

UNIDAD I : Fundamentos de la programación orientada a objetos

- Tema 2: Conceptos orientados a objetos
  - Subtema 2.1:La POO frente a la programación tradicional

UEA
UNIVERSIDAD
ESTATAL AMAZÓNICA

Unidad 1

Fundamentos de la programación orientada a objetos

Tema 2

Conceptos orientados a objetos

UEA
UNIVERSIDAD
ESTATAL AMAZÓNICA

Subtema 2.1: La POO frente a la programación tradicional

PROGRAMACIÓN ORIENTADA A OBJETOS
(UEA-L-UFB-030)

UEA
UNIVERSIDAD
ESTATAL AMAZÓNICA

La POO frente a la programación tradicional

Analizaremos la evolución de la programación y dos paradigmas fundamentales:

- La Programación Orientada a Objetos (POO)
- La Programación Tradicional

La POO ha transformado la manera en que concebimos y construimos software.

UEA
UNIVERSIDAD
ESTATAL AMAZÓNICA

La POO frente a la programación tradicional

Es crucial comprender este paradigma, que se distingue por su enfoque innovador en la **organización del código**.

La POO se basa en el concepto central de "**objeto**", entidades que encapsulan datos y funciones. Al comprender esta definición, sentaremos las bases para explorar las aplicaciones prácticas y los beneficios que la POO aporta al desarrollo de software moderno

UEA
UNIVERSIDAD
ESTATAL AMAZÓNICA

Programación Tradicional

En este enfoque, la lógica del programa se estructuraba principalmente de manera procedural, centrándose en la ejecución secuencial de instrucciones.

La modularidad se lograba mediante funciones y subrutinas, y el código se organizaba alrededor de las tareas a realizar.

Sin embargo, esta aproximación enfrentaba limitaciones en términos de reutilización de código y mantenimiento a medida que los programas crecían en complejidad.

La **falta de encapsulamiento** y la dependencia en exceso de las variables globales eran comunes.

Estas características, aunque fueron la norma en su tiempo, demostraron ser desafíos significativos a medida que las exigencias de desarrollo de software evolucionaron.

UEA
UNIVERSIDAD
ESTATAL AMAZÓNICA

La transición hacia la Programación Orientada a Objetos (POO)

Fue una respuesta estratégica a las limitaciones inherentes de la Programación Tradicional a medida que los proyectos de desarrollo de software se volvieron más grandes y complejos, la necesidad de gestionar eficientemente la creciente complejidad se volvió apremiante.

La POO abordó estas limitaciones al introducir conceptos fundamentales como la encapsulación, la herencia y el polimorfismo.

UEA
UNIVERSIDAD
ESTATAL AMAZÓNICA

Transcisión

La encapsulación permitió ocultar detalles internos de los objetos, fomentando así la modularidad y reduciendo la dependencia de variables globales.

La herencia facilitó la creación de nuevas clases basadas en clases existentes, fomentando la reutilización del código.

Por último, el polimorfismo permitió que objetos de diferentes clases respondieran de manera uniforme a un mismo conjunto de operaciones.

Esta transición no solo buscaba resolver desafíos técnicos, sino también mejorar la eficiencia y la colaboración en equipos de desarrollo.

La POO se convirtió en una respuesta integral a las demandas cambiantes del desarrollo de software, proporcionando una estructura más flexible y escalable para abordar proyectos cada vez más sofisticados.

Este cambio de paradigma marcó un hito crucial en la evolución de la programación, transformando la forma en que concebimos y construimos software en la actualidad.

UEA
UNIVERSIDAD
ESTATAL AMAZÓNICA

Ejemplo

Este ejemplo muestra cómo gestionar una cuenta bancaria tanto en programación tradicional como en programación orientada a objetos (POO) en Python.

La versión de POO utiliza una clase BankAccount con métodos para depositar, retirar y calcular el interés, lo que resulta en un código más modular y orientado a objetos.

ChatGPT

python Copy code

```python
# Programación Tradicional
# Ejemplo: Gestión de una cuenta bancaria

# Definición de variables globales
balance = 0
interest_rate = 0.05

# Función para depositar dinero en la cuenta
def deposit(amount):
    global balance
    balance += amount

# Función para retirar dinero de la cuenta
def withdraw(amount):
    global balance
    balance -= amount

# Función para calcular el interés y actualizar el saldo
def calculate_interest():
    global balance, interest_rate
    interest = balance * interest_rate
    balance += interest

# Uso de las funciones en la programación tradicional
deposit(1000)
withdraw(500)
calculate_interest()

# Imprimir el saldo final
print("Balance (Traditional):", balance)
```

UEA
UNIVERSIDAD
ESTATAL AMAZÓNICA

Ejemplo

Este ejemplo muestra cómo gestionar una cuenta bancaria tanto en programación tradicional como en programación orientada a objetos (POO) en Python.

La versión de POO utiliza una clase BankAccount con métodos para depositar, retirar y calcular el interés, lo que resulta en un código más modular y orientado a objetos.

```python
# Programación Orientada a Objetos (POO)
# Ejemplo: Gestión de una cuenta bancaria

class BankAccount:
    def __init__(self, initial_balance=0, interest_rate=0.05):
        self.balance = initial_balance
        self.interest_rate = interest_rate

    def deposit(self, amount):
        self.balance += amount

    def withdraw(self, amount):
        self.balance -= amount

    def calculate_interest(self):
        interest = self.balance * self.interest_rate
        self.balance += interest

# Crear una instancia de la clase BankAccount
account = BankAccount()

# Uso de los métodos en la programación orientada a objetos
account.deposit(1000)
account.withdraw(500)
account.calculate_interest()

# Imprimir el saldo final
print("Balance (OOP):", account.balance)
```

UEA
UNIVERSIDAD
ESTATAL AMAZÓNICA

Beneficios de Adoptar la POO

La POO brinda beneficios significativos que mejoran la eficiencia y la calidad del código.

1. **Reutilización de Código:** La POO fomenta la reutilización de código a través del concepto de clases y objetos. Las clases sirven como plantillas para crear objetos, promoviendo la escritura de código eficiente y modular.

2. **Modularidad**: La modularidad es una característica fundamental de la POO. El código se organiza en módulos independientes (clases), lo que facilita la comprensión, el mantenimiento y la actualización del software.

3. **Mantenimiento Eficiente**: La POO simplifica el mantenimiento del código. Los cambios en una clase no afectan necesariamente otras partes del programa, reduciendo el riesgo de errores y agilizando las actualizaciones.

UEA
UNIVERSIDAD
ESTATAL AMAZÓNICA

Beneficios de Adoptar la POO

La POO brinda beneficios significativos que mejoran la eficiencia y la calidad del código.

4. **Flexibilidad:** La POO permite una mayor flexibilidad en el diseño del software. Las clases y objetos modulares ofrecen la capacidad de adaptarse y evolucionar con los requisitos cambiantes del proyecto. Esta flexibilidad facilita la incorporación de nuevas funcionalidades y la modificación de las existentes.

5. **Escalabilidad**: La estructura modular de la POO favorece la escalabilidad del software. Al dividir el código en unidades autónomas (clases), es más sencillo gestionar y expandir proyectos a medida que crecen en complejidad. Esta característica es fundamental para proyectos a gran escala.

UEA
UNIVERSIDAD
ESTATAL AMAZÓNICA

Consejos para la Transición

1. **Entender los Principios Básicos de POO**: Destacamos la importancia de comprender los conceptos fundamentales de la POO, como clases, objetos, herencia y polimorfismo. Familiarizarse con estos principios es esencial para una transición exitosa.

2. **Práctica Continua**: Sugerimos la práctica constante mediante la creación de pequeños proyectos en POO. La experiencia práctica fortalece la comprensión y habilidades en este paradigma.

UEA
UNIVERSIDAD
ESTATAL AMAZÓNICA

Consejos para la Transición

3. **Refactorización Gradual**: Recomendamos una transición gradual, refactorizando el código tradicional para incorporar gradualmente conceptos de POO. Esto minimiza la curva de aprendizaje y facilita la adaptación.

4. **Recursos de Aprendizaje**: Proporcionamos recursos y referencias útiles, como libros, tutoriales en línea y comunidades, para apoyar el proceso de aprendizaje y facilitar la transición.

UEA

UNIVERSIDAD
ESTATAL AMAZÓNICA