ITSSAT

LISTA DE COTEJO PARA INVESTIGACION

INTITUTO TECNOLOGICO SUPERIOR DE SAN ANDRES TUXTLA CARRERA: INGENIERIA INFORMÁTICA		
DATOS GENERALES DEL PROCESO DE EVALUACION		
Nombre(s) del alumno(s): JEREZANO JARA CARLOS MARTIN		Firma del alumno(s):
Producto: Investigación Unidad I	Nombre del Proyecto: Investigación:Características de la programación orientada a objetos	Fecha:
Asignatura: Programación Avanzada	Grupo: 311- A	Semestre: Tercero
Nombre del Docente: MTI Lorenzo de Jesús Organista Oliveros		Firma del Docente:

INSTRUCCIONES				
Revisar las actividades que se solicitan y marque en los apartados "SI" cuando la evidencia se cumple; en caso contrario marque "NO". En la columna OBSERVACIONES indicaciones que puedan ayudar al alumno a saber cuáles son las condiciones no cumplidas, si fuese necesario.				
Valor del reactivo	*Características a cumplir (Reactivo)*	*CUMPLE*		*OBSERVACIONES*
		SI	*NO*	
1%	**Presentación.** El trabajo cumple con los requisitos de : a. Buena presentación	X		
1%	b. No tiene faltas de ortografía	X		
1%	c. Mismo formato (letra arial 14, títulos con negritas)	X		
1%	d. Misma calidad de hoja e impresión	X		
1%	e. Maneja el lenguaje técnico apropiado	X		
2%	**Introducción y Objetivo.** La introducción y el objetivo dan una idea clara del contenido del trabajo, motivando al lector a continuar con su lectura y revisión.	X		
5%	**Sustento Teórico.** Presenta un panorama general del tema a desarrollar y lo sustenta con referencias bibliográficas y cita correctamente a los autores.	X		
2%	**Desarrollo.** Sigue una metodología y sustenta todos los pasos que se realizaron al aplicar los conocimientos obtenidos, es analítico y bien ordenado.	X		
2%	**Resultados.** Cumplió totalmente con el objetivo esperado, tiene aplicaciones concretas.	X		
2%	**Conclusiones.** Las conclusiones son claras y acordes con el objetivo esperado.	X		
2%	**Responsabilidad.** Entregó el reporte en la fecha y hora señalada.	X		
20%	**CALIFICACIÓN:**	**20%**		

INSTITUTO TECNOLÓGICO SUPERIOR DE SAN ANDRÉS TUXTLA

INGENIERÍA MECATRÓNICA

GRUPO 311-A

MATERIA:

PROGRAMACION AVANZADA

ACTIVIDAD:

INVESTIGACIÓN UNIDAD 1

INTRODUCCION

DOCENTE:

LORENZO DE JESUS ORGANISTA OLIVEROS

ALUMNO:

CARLOS MARTIN JEREZANO JARA

SAN ANDRES TUXTLA, VER. Fecha: 20 de septiembre de 2025

Índice

1. Componentes Principales en la POO

1.1. Clases y objetos

Las **clases** son moldes que definen atributos y métodos. Los **objetos** son instancias de esas clases.
Ejemplo en Java:

```java
class Persona {
    String nombre;
    int edad;
}
public class Main {
    public static void main(String[] args) {
        Persona p = new Persona();
        p.nombre = "Carlos";
        p.edad = 20;
    }
}
```

1.2. Métodos y atributos

- **Atributos:** almacenan el estado de un objeto.
- **Métodos:** definen el comportamiento.

Ejemplo en Python:

```python
class Coche:
    def __init__(self, marca, modelo):
        self.marca = marca
        self.modelo = modelo

    def arrancar(self):
        print(f"El {self.marca} {self.modelo} ha arrancado.")

c1 = Coche("Toyota", "Corolla")
c1.arrancar()
```

1.3. Constructores y destructores

Los constructores inicializan objetos. Los destructores liberan recursos.
Ejemplo en C++:

```cpp
#include <iostream>
using namespace std;

class Persona {
public:
    Persona() { cout << "Objeto creado\n"; }
    ~Persona() { cout << "Objeto destruido\n"; }
};

int main() {
    Persona p;
}
```

1.4. Interfaces y clases abstractas

- **Clases abstractas:** definen métodos que deben implementarse en clases hijas.
- **Interfaces:** definen contratos que aseguran consistencia.

Ejemplo en Java:

```
interface Animal {
    void hacerSonido();
}
class Perro implements Animal {
    public void hacerSonido() {
        System.out.println("Guau guau!");
    }
}
```

2. Ventajas y Limitaciones de la POO

2.1. Productividad y mantenimiento

La POO incrementa la productividad al permitir reutilización y modularidad.
Ejemplo: una clase `Empleado` puede extenderse para crear `Gerente`, sin reescribir código base.

2.2. Complejidad inicial de aprendizaje

Al inicio, conceptos como herencia, polimorfismo y encapsulamiento pueden resultar difíciles.
Ejemplo en Python:

```
class Figura:
    def area(self):
        pass

class Circulo(Figura):
    def __init__(self, radio):
        self.radio = radio
    def area(self):
        return 3.14 * self.radio**2
```

Este ejemplo muestra herencia y polimorfismo, que requieren tiempo para dominar.

2.3. Rendimiento en comparación con otros paradigmas

La creación de múltiples objetos puede consumir más memoria que la programación estructurada. Sin embargo, las ventajas en diseño suelen compensarlo.

Ejemplo de instanciación masiva en C++:

```
for(int i=0; i<1000000; i++) {
    Persona *p = new Persona();
    delete p;
}
```

3. Componentes Principales en la POO

3.1. Clases y objetos

Las **clases** son moldes que definen atributos y métodos. Los **objetos** son instancias de esas clases.
Ejemplo en Java:

```java
class Persona {
    String nombre;
    int edad;
}
public class Main {
    public static void main(String[] args) {
        Persona p = new Persona();
        p.nombre = "Carlos";
        p.edad = 20;
    }
}
```

3.2. Métodos y atributos

- **Atributos:** almacenan el estado de un objeto.
- **Métodos:** definen el comportamiento.

Ejemplo en Python:

```python
class Coche:
    def __init__(self, marca, modelo):
        self.marca = marca
        self.modelo = modelo

    def arrancar(self):
        print(f"El {self.marca} {self.modelo} ha arrancado.")

c1 = Coche("Toyota", "Corolla")
c1.arrancar()
```

3.3. Constructores y destructores

Los constructores inicializan objetos. Los destructores liberan recursos.
Ejemplo en C++:

```cpp
#include <iostream>
using namespace std;

class Persona {
public:
    Persona() { cout << "Objeto creado\n"; }
    ~Persona() { cout << "Objeto destruido\n"; }
};

int main() {
    Persona p;
}
```

3.4. Interfaces y clases abstractas

- **Clases abstractas:** definen métodos que deben implementarse en clases hijas.
- **Interfaces:** definen contratos que aseguran consistencia.

Ejemplo en Java:

```java
interface Animal {
    void hacerSonido();
}
class Perro implements Animal {
    public void hacerSonido() {
        System.out.println("Guau guau!");
    }
}
```

4. Ventajas y Limitaciones de la POO

4.1. Productividad y mantenimiento

La POO incrementa la productividad al permitir reutilización y modularidad.
Ejemplo: una clase `Empleado` puede extenderse para crear `Gerente`, sin reescribir código base.

4.2. Complejidad inicial de aprendizaje

Al inicio, conceptos como herencia, polimorfismo y encapsulamiento pueden resultar difíciles.
Ejemplo en Python:

```python
class Figura:
    def area(self):
        pass

class Circulo(Figura):
    def __init__(self, radio):
        self.radio = radio
    def area(self):
        return 3.14 * self.radio**2
```

Este ejemplo muestra herencia y polimorfismo, que requieren tiempo para dominar.

4.3. Rendimiento en comparación con otros paradigmas

La creación de múltiples objetos puede consumir más memoria que la programación estructurada. Sin embargo, las ventajas en diseño suelen compensarlo.
Ejemplo de instanciación masiva en C++:

```cpp
for(int i=0; i<1000000; i++) {
    Persona *p = new Persona();
    delete p;
}
```

5. Comparación con Otros Paradigmas

5.1. Programación estructurada

Se basa en procedimientos y funciones, con enfoque en la secuencia de instrucciones.
Ejemplo en C:

```c
#include <stdio.h>

int suma(int a, int b) {
    return a + b;
```

```
}

int main() {
    int r = suma(3, 4);
    printf("Resultado: %d", r);
    return 0;
}
```

5.2. Programación funcional

Se centra en funciones puras y ausencia de estado.
Ejemplo en Haskell:

```
suma :: Int -> Int -> Int
suma a b = a + b

main = print (suma 3 4)
```

5.3. Programación orientada a eventos

Se organiza alrededor de eventos y manejadores de eventos.
Ejemplo en JavaScript:

```
document.getElementById("btn").addEventListener("click", () => {
    alert("  Botn   presionado!");
});
```

5.4. Diferencias clave

- La **estructurada** favorece la claridad en secuencias.
- La **funcional** enfatiza inmutabilidad y expresividad matemática.
- La **orientada a eventos** responde dinámicamente a entradas del usuario.
- La **POO** modela el mundo real con objetos y relaciones.

6. Conclusión

La **Programación Orientada a Objetos** se ha consolidado como paradigma dominante en la industria del software, gracias a su capacidad para modelar sistemas complejos, fomentar la reutilización y facilitar el mantenimiento.

Hoy en día, aunque la programación funcional moderna (ej. en `Scala`, `Kotlin`, `Python`) ha ganado fuerza, la POO sigue siendo indispensable en aplicaciones empresariales, videojuegos, inteligencia artificial y sistemas embebidos.

En el futuro, es probable que surja un **paradigma híbrido** donde convivan la POO y la programación funcional, integrando lo mejor de ambos enfoques.

7. Bibliografía

- Booch, G. (1994). *Object-Oriented Analysis and Design*. Addison-Wesley.
- Lafore, R. (2002). *Object-Oriented Programming in C++*. Sams Publishing.
- Eckel, B. (2006). *Thinking in Java*. Prentice Hall.

- Documentación oficial:
  - Python
  - Java
  - C++
  - C#

ITSSAT

GUIA DE OBSERVACIÓN PARA RESOLUCIÒN DE EJERCICIOS PRACTICOS

NOMBRE DE LA ASIGNATURA: Programación Avanzada

NOMBRE DE LA UNIDAD: Introducción

ALUMNO: JEREZANO JARA CARLOS MARTIN

INSTRUCCIONES

Revisar los documentos o actividades que se solicitan y marque en los apartados “SI” cuando la evidencia a evaluar se cumple; en caso contrario marque “NO”. En la columna “OBSERVACIONES” ocúpela cuando tenga que hacer comentarios referentes a lo observado.

Valor del reactivo	***Características a cumplir (Reactivo)***	***CUMPLE***		***OBSERVACIONES***
		Si	***NO***	
8%	¿Identifico el problema planteado?	X		
4%	¿Identifico los datos de entrada del problema?	X		
4%	¿Identifico los datos de salida del problema?	X		
8%	¿Generó la solución del problema en forma clara y comprensible (orden)?	X		
12%	¿Elaboró el programa respetando la sintaxis del lenguaje de programación (orden)?	X		
4%	Comprobó los resultados esperados a través de pruebas de escritorio?	X		
40%	***CALIFICACIÓN:***	**40%**		

```
/* PROGRAMACION AVANZADA
UNIDAD 1
ING.MECATRONICA. GRUPO:311-A
ALUMNO: CARLOS MARTIN JEREZANO JARA
PROGRAMA 7
Utiliza vectores y metodos para el llenado de datos
*/



#include <iostream>
using namespace std;

int vec[10];

void LlenaVector()
{
for (int x = 0; x < 10 ; x++)
    {
        cout <<"TECLEA LOS NUMERO \n";
        cin >> vec[x];
    }
}

void ImprimeVector()
{
for (int x = 0; x < 10; x++)
    {
        cout << "Numero [" << x << "]" << vec[x] << "\n";
    }
}



int main()
{
LlenaVector();

ImprimeVector();

}
```

Título*

Evaluación - Unidad I

*Obligatorio

Instrucciones (opcional)

-Realizar un programa que cree 5 métodos, cada uno tendrá una función diferente [Los métodos serán indicados por el docente] con las siguientes indicaciones:

- El programa incluirá ciclo Do-While y Switch [Incluir el menú de los métodos] []

1.- Subir el código fuente [EvaluacionUI.cpp]

2.- Una imagen [captura de pantalla del funcionamiento del programa]

Adjuntar

Drive YouTube Crear Subir Enlace NotebookLM Gem

```
/* PROGRAMACION AVANZADA - UNIDAD 1
ING. MECATRONICA GRUPO:311-A
ALUMNO: CARLOS MARTIN JEREZANO JARA

ESTE PROGRAMA PIDE EL NUMERO DE FILAS E IMPRIME * INCIANDO POR EL NUMERO ASIGNADO Y
DISMINUYENDO PROGRESIVAMENRTE
*/

#include <iostream>
using namespace std;

int main()
{
    int n;
      cout << " Teclee un numero:\n";
      cin >> n;

    for(int i=n; i>=1; i--)
       {
            for (int j=1;j<=i; j++) //j es igual a 1, siempre que j sea igual o
menor que i j va aunmentar de 1 en uno e imprime *
        {
            cout << "*";
        }
        cout <<"\n";
       }

}
```