REPÚBLICA DEL ECUADOR

Ministerio de Educación

UNIDAD EDUCATIVA "BAÑOS"

Baños de Agua Santa – Tungurahua

PROYECTO INTERMODULAR

"Temas de Segundo de Bachillerato Currículo del Módulo Formativo Nº3: Programación y Base de Datos"

Figura Profesional: Informática

Elaborado por: Anahi Sailema

Pamela Aponte

Elkyn Zambrano

Curso: Segundo "H"

Módulo formativo:

Módulo 3: Programación y Base de Datos

Docentes Técnicos: Mg. Diana Fuentes

Año Lectivo: 2024-2025

EL NUEVO ECUADOR

Índice

Índice 2
1. Programación orientada a objetos 3
1.1 Elementos 3
1.2 Características 4
1.3 Propiedades y ventajas de la programación orientada a objetos 5
2. **Estructuras de control:** 7
2.1 Clases y funciones 7
2.2 Librerías 7
2.3 Código fuente 7
2.4 Desarrollo de programas 7
2.5 Documentación de programas 7
3. Análisis Estructurado de Sistemas 8
3.1 Objetivos: 8
3.2 Entradas: 8
3.3 Salidas: 8
3.4 Fases: 9
3.5 Modelización de funciones y procesos 9
3.6 Modelización de datos y análisis entidad-relación 10
3.5 Programación con sistemas gestores de bases de datos relacionales: 10

1. Programación orientada a objetos

La Programación Orientada a Objetos (POO) es un paradigma de programación, es decir, un modelo o un estilo de programación que nos da unas guías sobre cómo trabajar con él. Se basa en el concepto de clases y objetos. Este tipo de programación se utiliza para estructurar un programa de software en piezas simples y reutilizables de planos de código (clases) para crear instancias individuales de objetos. (Martínez Canelo, 2020)

1.1 Elementos: La programación orientada a objetos (POO) es un paradigma de programación que se basa en la creación de objetos que representan entidades del mundo real. Estos objetos tienen propiedades y comportamientos que se definen mediante una serie de elementos que se describen a continuación:

Elementos de la POO

- Clases: plantillas o moldes que definen las propiedades y comportamientos de un objeto**.**

- Objetos: instancias de una clase que tienen propiedades y comportamientos definidos en la clase.

- Propiedades (Atributos): características de un objeto que se almacenan en variables.

- Métodos: acciones que un objeto puede realizar.

- Herencia: capacidad de una clase para heredar las propiedades y comportamientos de otra clase.

- Polimorfismo: capacidad de un objeto para tomar diferentes formas.

- Encapsulación: capacidad de un objeto para ocultar sus propiedades y comportamientos internos y solo exponer una interfaz pública.

- Abstracción: capacidad de un objeto para representar conceptos y entidades del mundo real de manera simplificada.

La programación orientada a objetos o POO consiste básicamente en el manejo de clases organizadas para generar un programa, estas tienen atributos básicos y pueden generar objetos con métodos y atributos específicos.

Pero observamos varios términos como Clases, Objetos, atributos y métodos, a continuación, explicaremos cada uno de ellos y su relación

Atributo:

Los atributos son las propiedades o estados de un elemento Clase u objeto, los atributos se declaran como variables del elemento y ayudan a estructurar el objeto de la clase.

Método:

Los métodos son el conjunto de funciones que pueden tener los elementos de un POO clase u objeto, pueden ser funciones aritméticas, comparación, medición, etc. Estos métodos pueden ser privados o públicos.

Clase:

Una clase en POO es una plantilla para la creación de objetos, esta plantilla es un elemento genérico o básico que tiene las características generales, comportamientos, atributos del objeto que se quiera crear.

Objeto e instancia:

Es una unidad de programa que consta de atributos o propiedades y de funciones o métodos, esta unidad de programa se desarrolla a partir de una plantilla o clase, el desarrollo del objeto a partir de una clase (plantilla) se llama instancia.

Ejemplos y Conceptos Adicionales

- Algoritmos, pseudocódigos, flujogramas, estructuras de control, tipos de variables, estructuras de datos, funciones y librerías básicas, documentación de programas, código fuente.

1.2 Características:

- **Encapsulación:** La encapsulación es la capacidad de un objeto para ocultar sus propiedades y comportamientos internos y solo exponer una interfaz pública. Esto ayuda a proteger la integridad de los datos y a mejorar la seguridad**.**

- **Abstracción:** La abstracción es la capacidad de un objeto para representar conceptos y entidades del mundo real de manera simplificada. Esto ayuda a reducir la complejidad y a mejorar la comprensión del sistema.
- **Herencia**: La herencia es la capacidad de una clase para heredar las propiedades y comportamientos de otra clase. La clase que hereda se llama subclase, y la clase de la que se hereda se llama superclase.
- **Polimorfismo**: El polimorfismo es la capacidad de un objeto para tomar diferentes formas. Esto se logra mediante la sobrecarga de métodos o la sobreescritura de métodos.
- **Composición**: La composición es la capacidad de un objeto para estar compuesto por otros objetos. Esto permite crear objetos complejos a partir de objetos más simples.
- **Interfaz:** La interfaz es la capacidad de un objeto para exponer una serie de métodos y propiedades que pueden ser utilizados por otros objetos.
- **Modularidad**: La modularidad es la capacidad de un sistema para estar dividido en módulos independientes que pueden ser desarrollados y mantenidos de manera separada.
- **Reutilización de código:** La reutilización de código es la capacidad de un sistema para reutilizar código existente en lugar de tener que escribirlo nuevamente. (HUMBERTO CISNEROS, 2017)

1.3 Propiedades y ventajas de la programación orientada a objetos:

⇨ **Propiedades**

- **Modularidad:** La POO permite dividir un sistema en módulos independientes, lo que facilita el desarrollo, mantenimiento y reutilización de código.
- **Reutilización de código:** La POO permite reutilizar código existente en lugar de tener que escribirlo nuevamente, lo que ahorra tiempo y recursos.
- **Flexibilidad:** La POO permite crear objetos que pueden ser utilizados en diferentes contextos y situaciones.

- **Escalabilidad**: La POO permite crear sistemas que pueden ser fácilmente escalados y modificados para adaptarse a nuevas necesidades.

- **Mantenimiento:** La POO permite crear sistemas que son fáciles de mantener y modificar, ya que los cambios se pueden realizar en un módulo específico sin afectar al resto del sistema**.**

  ⇨ **Ventajas más importantes de la programación orientada a objetos:**

  - **Reusabilidad.** Cuando hemos diseñado adecuadamente las clases, se pueden usar en distintas partes del programa y en numerosos proyectos.

  - **Mantenibilidad**. Debido a la sencillez para abstraer el problema, los programas orientados a objetos son más sencillos de leer y comprender, pues nos permiten ocultar detalles de implementación dejando visibles sólo aquellos detalles más relevantes.

  - **Modificabilidad**. La facilidad de añadir, suprimir o modificar nuevos objetos nos permite hacer modificaciones de una forma muy sencilla.

  - **Fiabilidad.** Al dividir el problema en partes más pequeñas podemos probarlas de manera independiente y aislar mucho más fácilmente los posibles errores que puedan surgir. (Roldán, 2025)

2. **Estructuras de control:** Clases y funciones. Librerías. Código fuente. Desarrollo de programas. Documentación de programas.

Estructuras de control: Son instrucciones en un programa que permiten alterar el flujo de ejecución en función de ciertas condiciones o repeticiones. Ayudan a que un programa tome decisiones, repita ejecutar acciones o bloques de código de manera condicional. Por ejemplo, en Dev C++ se usan:

- **if, else, while, for, etc.,** se utilizan para controlar el flujo del programa.

2.1 Clases y funciones: Las clases son plantillas para crear objetos (instancias) en programación orientada a objetos (OOP), y las funciones son bloques de código reutilizables que realizan tareas específicas.

- C++ soporta la Programación Orientada a Objetos (OOP), permitiendo el uso de clases, constructores y métodos.

2.2 Librerías: Son colecciones de funciones, clases y otros recursos que permiten realizar tareas comunes sin necesidad de escribir el código desde cero.

- Se pueden usar librerías estándar, como **math, stdio.h, conio.h**, para realizar tareas matemáticas comunes

2.3 Código fuente: Es el conjunto de instrucciones escritas en un lenguaje de programación que define el comportamiento de un programa.

2.4 Desarrollo de programas: Es el proceso completo de crear un software, desde la planificación, escritura de código, pruebas y documentación.

2.5 Documentación de programas: Es la descripción detallada del funcionamiento de un programa, incluyendo la explicación del código, las funciones, la estructura de datos y cómo se usa el software.

- Los comentarios en C++ pueden explicar el **propósito del código**, lo que es **útil** para la **comprensión** y **mantenimiento** del programa. (Prunello, 2024)

3. Análisis Estructurado de Sistemas

El análisis estructurado de sistemas es una metodología esencial en la ingeniería de software y la informática, que permite comprender, modelar y desarrollar sistemas de manera organizada y eficiente. Su enfoque se basa en descomponer los sistemas complejos en partes más simples y manejables, facilitando su diseño, implementación y mantenimiento. A continuación, se detallan sus componentes clave, herramientas y enfoques complementarios.

(Estudyando, 2025)

Objetivos, entradas, salidas y fases:

El análisis estructurado busca lograr una comprensión profunda del sistema y sus requisitos para garantizar que el diseño final sea funcional, eficiente y adaptable. Los elementos clave son:

3.1 Objetivos:

- Descomponer el sistema en módulos comprensibles.
- Mejorar la claridad y reducir la complejidad.
- Facilitar la implementación, pruebas y mantenimiento.
- Garantizar que el sistema cumpla con los requisitos del usuario.

3.2 Entradas:

- Datos brutos proporcionados por usuarios, sensores o sistemas externos.
- Documentación previa o análisis de sistemas heredados.

3.3 Salidas:

- Resultados procesados que satisfacen necesidades específicas del usuario.
- Reportes, interfaces gráficas o datos para otros sistemas.

3.4 Fases:

- Identificación de requisitos: Recolección de necesidades funcionales y no funcionales.
- Modelado del sistema: Creación de representaciones visuales y textuales de procesos y datos.
- Validación y verificación: Comprobación del diseño frente a los requisitos originales.
- Optimización: Revisión para mejorar eficiencia y escalabilidad.

(Cidecame, 2025)

3.5 Modelización de funciones y procesos:

La representación visual es clave para entender y comunicar el funcionamiento interno de un sistema. Las herramientas más utilizadas incluyen:

- Diagramas de flujo: Representan la secuencia de actividades y decisiones del sistema.
- Diagramas de contexto: Muestran la interacción del sistema con actores externos.
- Diagramas de procesos (DFD): Descomponen los procesos en niveles, mostrando cómo los datos fluyen entre ellos.

Diagramas de estado: Describen los diferentes estados por los que pasa el sistema y las transiciones entre ellos.

Además, se pueden integrar técnicas modernas como BPMN (Business Process Model and Notation) para mejorar la comprensión de los procesos de negocio asociados al sistema.

(Universidad de Valladolid, 2025)

3.6 Modelización de datos y análisis entidad-relación:

La estructura de los datos es tan importante como los procesos que los manejan. El análisis estructurado incluye:

- Modelo conceptual: Define la estructura lógica de los datos sin considerar detalles técnicos. Ejemplo: Entidades como Cliente, Producto y sus atributos.
- Análisis entidad-relación: Muestra las entidades, sus atributos y las relaciones entre ellas. Permite identificar cardinalidades y dependencias.
- Diccionario de datos: Documentación detallada de cada dato (tipo, formato, origen, destino, etc.) que asegura consistencia y facilita el mantenimiento.

Adicionalmente, se puede complementar con diagramas UML (Unified Modeling Language) como diagramas de clases para modelar más detalladamente las estructuras de datos.

(ComunidadBI, 2025)

3.5 Programación con sistemas gestores de bases de datos relacionales:

Los sistemas modernos requieren bases de datos robustas para almacenar y gestionar grandes volúmenes de información. En esta etapa, se emplean gestores como:

- MySQL, PostgreSQL, SQL Server: Permiten la creación de bases de datos relacionales.
- Lenguaje SQL: Se utiliza para consultas, inserciones, actualizaciones y eliminación de datos.Stored Procedures y Triggers: Automatizan procesos dentro de la base de datos para mejorar el rendimiento y garantizar la integridad.

(AppMaster, 2023)