INSTITUTO SUPERIOR
TECNOLÓGICO
VICENTE LEÓN

Guía general de estudio de la asignatura

PROGRAMACIÓN ORIENTADA A OBJETOS

Augusto Germánico Rodríguez Balarezo

Carrera de Desarrollo de Software
Asignatura: Programación Orientada a Objetos
DS09-2P196
Segundo Nivel

www.istvicenteleon.edu.ec

INSTITUTO SUPERIOR
TECNOLÓGICO
VICENTE LEÓN
Dirección Belisario Quevedo y Gral. Maldonado, Latacunga, Cotopaxi
Campus Matriz

PROGRAMACIÓN ORIENTADA A OBJECTOS

Autor: Augusto Germánico Rodríguez Balarezo

MSc. Ángel Velásquez Cajas Editor

Directorio editorial institucional
Mg. Omar Sánchez Andrade Rector
Mg. Fabricio Quimba Herrera Vicerrector
Mg. Milton Hidalgo Achig Coordinador de la Unidad de Investigación

Diseño y diagramación
Mg. Alex Zapata Álvarez
Mtr. Leonardo López Lidioma

Revisión técnica de pares académicos
– Camana Castro Carlos Aníbal
Instituto Superior Tecnológico Bolívar
c.camana@institutos.gob.ec
– Luis Gonzalo Borja Almeid
Universidad de las Fuerzas Armadas - ESPEL
lgborja2@espe.edu.ec

ISBN: 978-9942-676-39-9
Primera edición
Agosto 2024

RIMANA EDITORIAL

índice

DESARROLLO GUÍA DE ESTUDIO

1. Datos informativos

Augusto Germánico Rodríguez Balarezo, Ingeniero en Informática y sistemas computacionales, Licenciado en ciencias de la educación mención Sistemas Informáticos, Tecnólogo en Programación de Sistemas, Técnico Superior en Programación de Sistemas, Magister en Seguridad y prevención de riesgos del trabajo, Docente y técnico Tics del Instituto Superior Ramón Barba Naranjo, Docente del Instituto Superior Tecnológico Vicente León, asistente técnico en seguridad y riesgos laborales, técnico tics de empresas públicas y privadas.

2. Presentación de la Asignatura

La POO es un enfoque de programación que crea aplicaciones y programas informáticos al utilizar objetos y sus interacciones.

Se basa en encapsulamiento, polimorfismo, abstracción y herencia. A principios de la década de los años 90, se volvió muy popular.

Para internalizar los principios y técnicas esenciales de modelación y programación de sistemas de baja y media complejidad, los estudiantes aprenderán el paradigma de orientación a objetos, como técnica de programación, en esta clase.

3. Introducción de los Temas

Esta guía presenta los siguientes temas de estudio.

La unidad I trata aspectos relevantes referentes a la Introducción al paradigma de Programación Orientado a Objetos, Tipos de datos, variables, constantes y operadores, clase, constructores de clase, que permite al alumno resolver problemas algorítmicos aplicando técnicas de programación a través de un computador.

4. Objetivos de Aprendizaje

– Diseñar programas utilizando el paradigma orientado a objetos en casos de la vida real

– Entender la lógica de programación orientado a objetos a través de algoritmos y pseudocódigos.

– Conocer y entender las diferentes estructuras de programación orientado a objetos.

– Aplicar adecuadamente las diferentes estructuras de control de programación orientado a objetos en los problemas planteados.

5. Unidad y Subunidades

1. Introducción al Paradigma Orientado a Objetos
1.1. El Paradigma Orientado a Objetos
1.2. Lenguajes Orientados a Objetos
1.3. Tipos de Datos
1.4. Variables, Constantes y Operadores
1.5. Características Generales de la POO
1.6. Introducción a las Clase
1.7. Constructores de Clase

6. Resultados de Aprendizaje de la Unidad

– Analiza los problemas planteados y los soluciona a través de pseudocódigos y diagramas de flujos.

– Diseña programas utilizando el paradigma orientado a objetos en casos de la vida real.

– Realiza sentencias e instrucciones de programación usando un lenguaje

– De programación orientado a objetos.

– Realiza pruebas de la funcionalidad de los programas desarrollados.

– Describe como se aplican las sentencias e instrucciones de programación en la resolución de problemas planteados

7. Estrategias Metodológicas

Se espera que el aprendizaje se base en el análisis y resolución de problemas, el uso efectivo de la información y el fomento de la retención, comprensión, uso o aplicación de conceptos, ideas, principios y habilidades en la resolución de problemas.

La resolución de casos ayudará a desarrollar procesos complejos de pensamiento, como el análisis, razonamiento, argumentación, revisiones y profundización en una variedad de temas. Se desarrollan las habilidades proyectadas en función de las competencias y se utiliza el software que permitirá aplicar la teoría a la práctica.

Se realizan ejercicios específicos de la asignatura, así como otros relacionados con el campo de estudio.

El proceso de evaluación se llevará a cabo en tres etapas: diagnóstica, formativa y sumativa, evaluando el desarrollo del estudiante en cada tarea y en especial en las evidencias del aprendizaje de cada unidad.

Evaluación formativa: Proporciona retroalimentación continua y constructiva a los estudiantes para que puedan identificar sus fortalezas y áreas de mejora. Esto les ayuda a desarrollar habilidades de autorreflexión y adaptación.

8. Criterios de Evaluación

Fases	Instrumentos	Primer Parcial %(Puntos)	Segundo Parcial %(Puntos)	Promedio %(Puntos)

Fase 1: Trabajos Prácticos	Trabajos Individual	2	2	2
	Trabajo de clase o colaborativo	2	2	2
	Exposiciones	2	2	2
Fase 2: Lecciones	Escritas	2	2	2
Fase 3: Evaluación	Cuestionario	2	2	2
	Total	10	10	10

Horas Docencia= 24
Horas de Trabajo Autónomo= 0
Horas de prácticas experimentales= 24

9. Desarrollo de las Subunidades

1. Introducción al Paradigma Orientado a Objetos

El paradigma orientado a objetos es un enfoque de programación que organiza el código en entidades llamadas "objetos", que son instancias de clases. Una clase es un plano o plantilla que define la estructura y el comportamiento de los objetos (Blasco, 2019).

1.1. El Paradigma Orientado a Objetos

La programación orientada a objetos representa un modelo donde la resolución de problemas se logra mediante objetos interconectados a través de mensajes. Su esencia radica en la organización de programas de manera análoga a la disposición de objetos en el mundo real (Ceballos, 2019). Esta metodología simplifica la abstracción y conceptualización de sistemas complejos, proporcionando un enfoque intuitivo y eficaz para el desarrollo de software. En esencia, la POO busca imitar la estructura organizativa natural de objetos para ofrecer soluciones más claras y eficientes.

Además, para JavaTPoint (2021) la orientación a objetos es un paradigma de programación que facilita la creación de software de alta

calidad porque sus componentes fomentan el mantenimiento, la expansión y la reutilización del software creado bajo este paradigma. Este enfoque no solo simplifica la creación de software, sino que también promueve su mantenimiento eficiente, expansión continua y la valiosa reutilización de componentes. Java, con su sólido soporte para la OOP, se erige como un lenguaje poderoso que impulsa la construcción de aplicaciones robustas y escalables, fundamentadas en los principios de este paradigma.

La programación orientada a objetos busca adaptarse al pensamiento humano en lugar del de la máquina. Esto es posible gracias a la forma racional en que se manejan las abstracciones que representan las entidades del dominio del problema, así como a propiedades como el encapsulamiento o la jerarquía.

Este paradigma se basa en un objeto en lugar de la función, que es un componente fundamental de la programación. Un objeto es una representación de algo, uno de los pilares fundamentales de Java es el uso de la orientación a objetos.

Por ejemplo, en un sistema de gestión de biblioteca, podríamos tener objetos como "Libro" y "Usuario". Un objeto "Libro" podría tener atributos como título y autor, mientras que un objeto "Usuario" podría tener atributos como nombre y lista de libros prestados. Estos objetos se conectan a través de mensajes; por ejemplo, el objeto "Usuario" podría enviar un mensaje al objeto "Libro" para solicitar un préstamo. Así, el objeto "Usuario" podría enviar un mensaje al objeto "Libro" para solicitar un préstamo. Así, la programación orientada a objetos modela la interacción entre objetos, replicando la organización intuitiva del mundo real en el diseño de software.

1.1.1. Principios de la Programación Orientada a Objetos

La Programación Orientada a Objetos (POO) se basa en cuatro principios fundamentales: abstracción, encapsulamiento, herencia y polimorfismo (Edteam, 2023).

1.1.1.1. Abstracción: Consiste en simplificar sistemas complejos al descomponerlos en componentes manejables. En la POO, este proceso se

lleva a cabo mediante la creación de clases y objetos que encapsulan los detalles internos de manera efectiva.

1.1.1.2. Encapsulamiento: La encapsulación es el principio de ocultar los detalles internos de una clase y exponer solo lo que es necesario. Esto se logra mediante el uso de modificadores de acceso (como públicos, privados y protegidos) para controlar el acceso a los miembros de una clase.

1.1.1.3. Herencia: Permite crear una clase nueva basada en una clase existente, heredando sus propiedades y comportamientos. Esto promueve la reutilización del código y la creación de jerarquías de clases.

1.1.1.4. Polimorfismo: El polimorfismo permite que un objeto pueda tomar muchas formas. Puede referirse tanto a la capacidad de una clase para tomar múltiples formas (polimorfismo de tiempo de compilación) como a la capacidad de un objeto para cambiar su comportamiento en tiempo de ejecución (polimorfismo de tiempo de ejecución).

Estos principios son fundamentales para entender y aplicar la programación orientada a objetos de manera efectiva. Al seguir estos conceptos, los desarrolladores pueden crear código más modular, flexible y fácil de mantener.

El siguiente ejemplo ilustra la aplicación de los principios de fundamentales de la programación orientada a objetos. Supongamos que estamos creando un sistema de gestión de una biblioteca.

Abstracción:

– Se crea una clase llamada Libro que encapsula los detalles internos de un libro, como el título, autor y número de páginas. Esta clase proporciona una abstracción del concepto de un libro.

```
class Libro:

def __init__(self, titulo, autor, num_paginas):
        self.titulo = titulo
```

```
        self.autor = autor
        self.num_paginas = num_paginas
```

Encapsulación:

– Se utiliza modificadores de acceso para encapsular los detalles del libro. Por ejemplo, se podría hacer que el número de páginas sea privado para limitar el acceso directo desde fuera de la clase.

```
class Libro:

def __init__(self, titulo, autor, num_paginas):
        self.titulo = titulo
        self.autor = autor
        self.__num_paginas = num_paginas

def obtener_num_paginas(self):
        return self.__num_paginas
```

Herencia:

– Se crea una clase LibroDeFiccion que hereda de la clase base Libro. Esto permite reutilizar la funcionalidad de la clase base y agregar características específicas para los libros de ficción.

```
class LibroDeFiccion(Libro):

def __init__(self, titulo, autor, num_paginas, genero):
        super().__init__(titulo, autor, num_paginas)
        self.genero = genero
```

Polimorfismo:

– Se puede utilizar el polimorfismo para tratar tanto a Libro como a LibroDeFiccion de manera uniforme al acceder a métodos comunes, como obtener el número de páginas.

```
def imprimir_info_libro(libro):

print(f'Título: {libro.titulo}, Autor: {libro.autor}, Páginas: {libro.
obtener_num_paginas()}')
```

Este ejemplo ilustra cómo los principios de la POO se aplican en la creación de un sistema de gestión de biblioteca, proporcionando un diseño claro, modular y fácil de entender. La abstracción, encapsulación, herencia y polimorfismo trabajan juntos para mejorar la estructura y la flexibilidad del código.

1.1.2. Beneficios de la Programación Orientado a Objetivos.

La programación orientada a objetos (POO) tiene varios beneficios. En la Tabla 1 se detallan los beneficios del POO.

Tabla 1

Beneficios de la Programación Orientado a Objetivos

Beneficio	Descripción
Reutilización de código	La programación orientada a objetos facilita la reutilización de código mediante el uso de clases y objetos. Esto puede ayudar a reducir el tiempo de desarrollo y los costos de mantenimiento.
Modularidad	La programación orientada a objetos permite dividir el código en módulos independientes, lo que facilita su mantenimiento y comprensión.
Extensible	La POO permite agregar nuevas características o modificar las existentes sin necesidad de modificar todo el código.
Instalación de mantenimiento	La POO facilita la detección y corrección de errores, ya que los objetos están bien definidos y aislados entre sí.
Similitud con el mundo real	La POO modela el mundo real de forma más natural, lo que puede ayudar a los programadores a comprender mejor el problema que están tratando de resolver.

Nota. Adaptado de (Blasco, 2019)

1.1.3. Conceptos de Programación Orientada a Objetos

A continuación, se conocerá los conceptos para comprender mejor cómo la POO mejora la eficiencia y la calidad del software.

1.1.3.1. Objeto: Un objeto en la POO es una entidad virtual (o entidad de software) con datos y funciones que imitan sus propiedades. Así lo menciona Blasco (2019) un objeto es un "elemento de forma elíptica, con el nombre de la clase a que responde en la parte superior" (p. 91).

Un objeto se caracteriza por dos elementos fundamentales. Entre estos elementos se tiene su estado y su comportamiento. Por su estado hace referencia a los datos vinculados al objeto que describen su condición interna en un momento especifico. Por otra parte, por su comportamiento, se refieren a la forma en que el objeto reacciona ante estímulos provenientes del entorno externo.

Los atributos, como "saldo" (double), "calificacion" (float) y "on" (boolean), representan la información inherente al objeto y reflejan su estado mediante valores específicos. Por otro lado, los métodos son funciones llamadas por otros objetos que definen acciones y comportamientos. Estos métodos pueden influir en el objeto cuando hay modificaciones en los valores de los atributos. Ejemplos ilustrativos de funcionalidades asociadas al objeto incluyen "obtenerSaldo()" para devolver el saldo, "calcularPromedio()" para calcular el promedio y "seOprimioOnOff()" para verificar el estado del interruptor.

1.1.3.2. Clase: "Una clase se interpreta como la abstracción de un tipo específico de objetos, si bien cada instancia construida a partir de esa clase contendrá datos únicos" (Ceballos, 2019, p. 41).

En otras palabras, una clase se concibe como una entidad que encapsula tanto atributos como métodos, delineando así un tipo particular de objeto. Puede visualizarse una clase como un 'modelo' o 'plantilla' destinada a generar objetos. Estos objetos, por su parte, representan instancias concretas de la clase y cada uno posee valores distintos para los atributos definidos en la mencionada clase.

Un programador designa como clase a un tipo de dato que establece para la creación de objetos. Cada objeto se considera una instancia única perteneciente a una clase específica de objetos.

Las propiedades comunes de un conjunto de objetos se definen por clase. Como lo hace con un tipo de datos compuesto, el programador define una clase y le da un nombre.

Los objetos en programación representan instancias de una clase. A continuación, se presenta la definición de la clase "Reloj" como ejemplo. Siguiendo la convención, los nombres de las clases siempre inician con mayúscula. En esta situación, un reloj está compuesto por tres indicadores: horas, minutos y segundos, que se traducen en las propiedades de la clase "Reloj", representadas por tres valores numéricos enteros. Las acciones que definen el funcionamiento de este reloj abarcan la inicialización de la hora con un valor predeterminado, el aumento de la lectura actual en un segundo y la obtención de la hora actual. Estas acciones se definen a través de los métodos set, incrementar y getHora.

```
public class Reloj {
private int hora;
private int minuto;
private int segundo;
hora = h % 24;
minuto = m % 60;
segundo = s % 60;
}
public void incrementar() {
segundo = (Segundo + 1) % 60;
If (segundo == 0 ) {
minuto = (minuto + 1) % 60;
If (minuto == 0 ) {
hora = (hora + 1) % 24;
}
}
}
```

```
public String getHora() {
return hora + ":" + ":" + segundo;
}
}
```

1.2. Lenguajes Orientados a Objetos

La programación orientada a objetos constituye un paradigma de programación en el cual se construye su estructura en torno a datos u objetos, en lugar de depender de funciones y lógica aislada (Ceballos, 2019). Estos tipos de lenguajes se centran en los objetos que los programadores requieren manipular, desviándose de la atención exclusiva en la lógica necesaria para dicha manipulación. De manera más específica, un objeto se caracteriza como un conjunto de datos con atributos y comportamientos distintivos

De este modo, una característica fundamental de esta modalidad de programación radica en su capacidad para manejar objetos vinculados a tipos o clases particulares. Estos tipos pueden heredar diferentes atributos de clase superior o general. Es decir, esta metodología de programación se emplea especialmente en programas extensos y complejos que requieren actualizaciones periódicas.

Comprender los elementos del programa es muy importante en el desarrollo del mismo. La idea central del paradigma de la programación orientada a objetos es el concepto de objetos como entidades que contienen datos y funcionalidad (García & Pardo, 2018).

La identificación de objetos es la clave o cuello de botella en la aplicación del diseño y análisis orientado a objetos. Existen varios métodos para determinar qué abstracciones representan o capturan mejor la semántica del problema que se resuelve (García & Pardo, 2018).

En programación funcional, las funciones son elementos clave que definen el comportamiento de un programa. Las conversiones de funciones únicamente pueden manifestarse como información relativa a los valores que han sido ingresados o producidos por la función en cuestión. Sin embargo, este

es el fin de su existencia, no tienen otra vida que la función. Sin embargo, en la programación imperativa, los datos pueden considerarse los componentes básicos de un programa.

Se trata de entidades independientes y las funciones las utilizan y ajustan. La POO en diferentes lenguajes informáticos constituye una de las herramientas más importantes para poder moldear y especificar diferentes procesos del mundo real que son difíciles de expresar en formatos de programación funcional, ya que puede mantener y manipular estados específicos (datos, valores variables). El estado de un programa se puede alterar implementando funciones.

Hoy en día existe, existen varios tipos de lenguaje de programación orientado a objetos que se estudian a continuación:

1.2.1. Java

Es un lenguaje de programación que tuvo su origen de inicio como una herramienta enfocada a la programación, determinada para el análisis de datos básicos que posteriormente se le llamaron lenguajes (Blasco, 2019).

Fue diseñado por los programadores James Gosling y Bill Joy. Java fue desarrollado a partir de un lenguaje de programación denominado Oak cuyo fundamento era objetivo era la creación de un software diseñado de manera directa al servicio de una interface para la televisión que pueda relacionarse de manera inteligente con el usuario (Fernández, 2015).

Algunas de las características más destacadas de este programa es que algunos pueden considerarlo un lenguaje compilado e interpretado, lo que significa que le da un grado de independencia muy suficiente al ordenador, generalmente es programación orientada a objetos.

Los métodos utilizados en este lenguaje de programación requieren de una verificación constante que se realiza durante su ejecución, mientras que este procedimiento Java siempre garantiza la seguridad al verificar el código antes de la ejecución y volver a hacerlo en tiempo de ejecución.

Frecuentemente se sostiene que Java exhibe un rendimiento más lento debido a su necesidad de interpretar códigos de bytes y convertirlos en código nativo antes de la ejecución de un método. Sin embargo, la tecnología JIT (compilación en tiempo de ejecución) ha optimizado este proceso al ejecutarlo solo una vez. Después de esta primera ejecución, el código nativo se almacena y permanece disponible para su utilización en futuras ocasiones, contribuyendo así a mejorar la eficiencia del rendimiento (Deitel, 2016).

De la misma manera se puede finalizar mencionando que Java cuenta con gestión de seguridad para bloquear de manera dinámica todo acceso a los recursos que presenta el sistema.

1.2.2. Ada

Ada es conocido como un lenguaje de programación que se enfoca de manera general en los objetos especial mente aquellos que son de forma estática (Ceballos, 2019).

Un aspecto clave de la tecnología Ada-CCM es facilitar la generación automática de código de todos los elementos posibles, centrando así los esfuerzos del implementador en el diseño del código de negocio, liberándolo del conocimiento de los detalles técnicos, el entorno o las interacciones entre componentes. Este objetivo guió el diseño de la estructura de componentes propuesta (Munguía, 2017).

El lenguaje fue desarrollado en nombre del Departamento de Defensa de Estados Unidos de Norteamérica. En los años 70, este departamento tenía proyectos en varios idiomas y gastaba mucho dinero en software (Minguet, 2013).

En sus orígenes, Ada generó mucha expectativa, ya que se consideraba como un lenguaje diseñado para dominar de manera efectiva trasladándose de temas relacionados con la defensa hacia el público en general.

En sus inicios, Ada recibió mucha atención de toda la comunidad de programación. Sus partidarios y otros que podría convertirse en el lenguaje

dominante para la programación de propósito general, no sólo para trabajos relacionados con la defensa. Ichbiah declaró públicamente que dentro de diez años solo quedarán dos lenguajes de programación (Minguet, 2013).

1.2.3. C++

Es un lenguaje de programación el cual es atribuido de manera directa a que fue desarrollada para extender la programación que no reciba ni parámetros ni argumentos (Díaz, 2016).

Bjarne Stroustrup desarrolló C++ en 1979. C++ fue desarrollado para ampliar las capacidades del lenguaje de programación C. Además, se introdujeron atributos que simplificarían la manipulación de objetos en C++. Desde el punto de vista de los lenguajes orientados a objetos, se clasifica a C++ como un lenguaje híbrido. (Díaz, 2016).

Entre la mayoría de los requerimientos especializados de este lenguaje de programación, tenemos alguna de los siguientes:

– Claridad conceptual
– Modelo puro de objetos
–Seguridad
– Alto nivel
– Modelo sencillo
– Sintaxis legible
– No ser redundante
– Pocas construcciones
– Facilidad de transición
– Facilidad de corrección
– Ambiente de trabajo amigable

Cada una de estas características es importante para la formación misma del lenguaje de programación.

Considerando los conocimientos y destrezas que se adquiere en el ámbito de la herencia, se sugiere la elaboración de ejercicios que permitan la

creación de nuevas clases, ampliando aquellas que se hayan desarrollado en asignaturas anteriores. Esto podría implicar la creación de una nueva categoría o la expansión de las funciones de la clase existente, aunque se aconseja evitar este último enfoque (Miños, 2017).

1.2.4. C#

C# cuyo significado en inglés es Sharp y se puede traducir al español como almohadilla es un nuevo lenguaje de programación desarrollado de manera general para la plataforma de Microsoft Sus principales creadores son Scott Wiltamuth y Anders Hejlsberg, a quein también se la atribuye el diseño del lenguaje de programación Turbo Pascal y la herramienta RAD Delphi (De Algoritmos, 2016).

En síntesis, este lenguaje incorpora las características más destacadas de otros lenguajes como Visual Basic, Java y C++, fusionándolas en uno único. Aunque sea un lenguaje relativamente reciente, su novedad no implica inmadurez, ya que Microsoft lo utilizó extensivamente en la creación de la Biblioteca de Clases Base (BCL). Esto ha resultado en que su compilador sea uno de los más precisos y optimizados dentro del Software Development Kit (SDK) de .NET Framework.

Alguna de las características más impresionantes de este lenguaje de programación es:

– Modernidad
– Está enfocada de manera directa al objeto
– Gestión automática de memoria.
– Instrucciones seguras
– Extensibilidad de operadores
– Eficiente

En el proceso de construcción de la plataforma .NET, las bibliotecas de clases se escribieron originalmente en un sistema de código administrado denominado Simple Managed C (SMC). En el mes de abril de 1999, Anders Hejlsberg encabezó un equipo con la tarea de desarrollar un lenguaje orientado

a objetos completamente nuevo. Sin embargo, fue necesario cambiar su nombre debido a problemas de marca registrada, y finalmente, se adoptó la denominación C# (Visual C#, 2013).

1.2.5. Ruby

Dentro de los muchos lenguajes de programación todos ellos están enfocados a la generación de códigos abiertos que puedan ser distribuidos, para el uso eficiente de recursos, objetos y sistemas del equipo.

Ruby representa un lenguaje de programación interpretado, reflexivo y centrado en objetos desarrollado por el programador japonés Yukihiro "Matz" Matsumoto. Matsumoto inició su labor en 1993 con Ruby, lo presento al público en 1995. Este lenguaje fusiona una sintaxis inspirada en Python y Perl, junto con características de programación orientada a objetos semejantes a las de Smalltalk. (Pérez & Esquivel, 2007).

Ruby fue desarrollado por Matsumoto, quien inició su trabajo en este lenguaje el 24 de febrero de 1993 y lo lanzó al público en 1995. Entre los amigos de Matsumoto, este lenguaje recibía en broma el sobrenombre de Ruby, en referencia a la "joya" del lenguaje de programación Perl en inglés (Pérez, 2005).

1.2.6. Python

Python es un lenguaje de programación de gama alta de nivel de programación porque corre de manera que es completamente legible pues su código es de fácil entendimiento, se utiliza de manera muy funcional para el desarrollo aplicaciones digitales entre ellas tenemos las más utilizadas en la actualidad como, por ejemplo: Instagram, Netflix, Spotify, Panda3D, entre otros (Challenger et al., 2014).

Python se clasifica como un lenguaje de programación multiparadigma, por que respalda de manera parcial la programación orientada a objetos, la programación imperativa y, en menor medida, la programación funcional. Además, es interpretado, dinámico y cuenta con compatibilidad en varias plataformas.

Python fue creado por Guido van Rossum, un desarrollador nacido en Holanda a finales de la década de los 80 y principios de los 90, mientras trabajaba en el sistema operativo Amoeba. Se diseñó principalmente para gestionar excepciones y conectarse a Amoeba, sirviendo como sucesor del lenguaje ABC (Challenger et al., 2014).

Para respaldar otros paradigmas, se han introducido extensiones. La escritura dinámica y las referencias son características clave utilizadas por Python para gestionar la memoria.

Un aspecto relevante de Python es su capacidad para la resolución dinámica de nombres, es decir, la vinculación de métodos y nombres de variables durante la ejecución del programa, también conocido como enlace dinámico de métodos.

1.2.7. PHP

El lenguaje PHP, cuyo nombre es un acrónimo de PHP: Hipertext Preprocessor, se clasifica como un lenguaje de programación interpretado y presenta una estructura de escritura similar a la de C++ o JAVA. Aunque tiene la capacidad de ser utilizado para la creación de diversos tipos de programas, su mayor popularidad se ha alcanzado en la generación dinámica de páginas web. Con frecuencia, se incorpora directamente en páginas HTML (o XHTML), ejecutándose a través del servidor web correspondiente (Duarte & Pérez, 2007).

Hoy en día la programación es muy importante. Al estar en un mundo que está afrontando un cambio digital se vuelve una herramienta de trabajo diario.

El desconocimiento de las personas sobre lenguajes de programación tiende a iniciar el proceso de aprendizaje con este programa siendo muy accesible y fácil de usar.

Por esta razón las personas que utilizan este lenguaje de programación pueden usarlo para la resolución de conflictos y en el proceso poder desarrollar sus habilidades y aprender más (Troya, 2019).

También hay muchas ventajas de utilizar el lenguaje PHP que también refuerzan este escenario tan positivo.

PHP es un lenguaje de programación creado con el propósito de desarrollar aplicaciones web y crear páginas web que faciliten la interacción entre servidores y la interfaz de usuario. Un aspecto que contribuyó significativamente a la popularidad de PHP es su naturaleza de código abierto (Troya, 2019).

Por lo tanto, cualquier programador puede modificar la estructura de programación ya que se trata de un software de código abierto y no existen restricciones de uso relacionadas con los derechos.

El usuario puede utilizar PHP para programar en cualquier tipo de proyecto y lograr comercializarlo sin ningún problema. Gracias a una comunidad de desarrolladores involucrados de manera directa y proactiva, se mejora constantemente.

1.3. Tipos de Datos

Los primeros lenguajes de programación no utilizaban objetos, sólo variables. Una variable puede describirse como un espacio en la memoria de su computadora donde asigna contenido.

Las variables pueden ser numéricas (solo numéricas, incluidos los valores calculados) o de caracteres o cadenas (valores alfanuméricos que constan únicamente de texto) números y texto mixtos.

Los tipos de datos pueden contener una variedad de datos, como números enteros, números decimales, caracteres individuales, cadenas (texto), valores booleanos (verdaderos o falsos), fechas y horas, y estructuras de datos complejas, como tablas o matrices (Dadtdata, 2023).

La mayoría de los lenguajes de progresión que son utilizados hoy en dos es fundamental conocer que existirán dos tipos de datos muy diferenciados y estos serán los que modifiquen las variables y parámetros que direccionan la función del programa. Importancia de los tipos de datos

Los tipos de datos son muy importantes por que ayudan que los diferentes tipos de leguajes de programación puedan procesar datos de manera muy precisa y eficiente, los tipos de datos están estrechamente ligados de manera clara con el enfoque de que los programas entiendan y se acoplen a varios modos de información como pueden ser texto, fechas, números y unidades de tiempo y conjuntos de datos de gran dificultad (Dadtdata, 2023).

Entre ellos tenemos:

Enteros (integers):

Los números enteros son representaciones de valores que están dentro del conjunto de números que no poseen decimales es decir 1,2,3,4 hasta el infinito ya sean positivos y negativos, son usados en operaciones aritméticas y de lógica matemática (Dadtdata, 2023).

Flotantes (floats):

Los números flotantes son representaciones de valores numéricos decimales en los lenguajes de programación se representan con notación decimal usando un punto que significa el inicio de la parte decimal como por ejemplo 1.5, 5.3, 5.7.

hasta el infinito ya se utilizan de manera muy común en operaciones aritméticas complejas que requieren precisión decimal (Dadtdata, 2023).

Caracteres (characters):

Son un conjunto de pictogramas y símbolos individuales, como letras, números y signos de puntuación entre otros. En estos lenguajes de programación, los caracteres se escriben utilizando comillas simples como por ejemplo, ‘m’, ‘b’, ‘c’, ‘5’, ‘2’, ‘2’, ‘@’, ‘#’ entre otros.

Los caracteres suelen utilizarse se utilizan comúnmente en el almacenamiento de texto y de cadenas de caracteres específicos con significados propios (Dadtdata, 2023).

Cadenas de caracteres (strings):

Estos datos se conforman por una secuencia estructural de caracteres que están conformados por palabras que pueden conformar frases y oraciones completas, en los lenguajes de programación conocidos estas cadenas se representan utilizando comillas dobles (por ejemplo, "hola", "123abc", "esto es una cadena", etc.). Las cadenas de caracteres son utilizadas para que las entradas del usuario se puedan almacenar en forma de texto que guarda información de un conjunto de datos (Dadtdata, 2023).

Booleanos (booleans):

Estos datos son un conjunto de valores que se pueden manifestar como proposiciones que tienen valores de verdadero o falso. En la mayoría de los lenguajes de programación, los valores booleanos se introducen mediante el uso de palabras claves como lo son "True" o "False".Este tipos de valores booleanos se utilizan en expresiones condicionales que son directores para la toma de decisiones lógicas que se desarrollan dentro del equipo (Dadtdata, 2023).

Arreglos (arrays):

Las matrices son colecciones de datos similares agrupados en estructuras de una, dos, tres o más dimensiones. Las matrices se utilizan normalmente para almacenar conjuntos de datos que están relacionados entre ellos mediante índices numéricos. (Dadtdata, 2023).

Fechas y horas (dates and times):

Son datos que representan unidades de tiempo. En la gran parte de este tipo de lenguajes de programación, la información de fecha y hora se representa en un formato específico que esta entre comillas "2022-01-01" entendiéndose que el 1 de enero de 2022.

Esta información de fecha y hora se utiliza a menudo en aplicaciones e informes que requieren un manejo de unidades de tiempo (Dadtdata, 2023).

1.3.1. Datos Primitivos

Como se ha mencionado, Java es un lenguaje de tipado estático. En otras palabras, el tipo de datos de una variable se define cuando se define.

Por lo tanto, a todas las variables se les asigna un tipo de datos.

Es importante saber que estos son tipos de datos del lenguaje y no representan objetos. Algo que ocurre con otros elementos del lenguaje Java.

Byte

Es la representación de una forma de datos conformada por 8 bits con signo. La cual puede almacenar valores numéricos que están en un rango de relación de 128 a 127 (ambos inclusive) (Canal Línea de código, 2016).

Short

Representa a una relación de datos de 16 bits con signo. Puede almacenar valores establecidos en rangos numéricos de:

-32.768 a 32.767 (Canal Línea de código, 2016).

Int

Este tipo de dato consta de 32 bits con signo y se utiliza para poder almacenar valores de carácter numérico. Cuyos límites están entre:

-231 y 231-1 (Canal Línea de código, 2016).

Long

Este conjunto de datos se expresa en de 64 bits con signo, este tipo de datos almacena valores numéricos comprendidos entre :

-263 a 263-1 (Canal Línea de código, 2016).

Float

Es un tipo dato se utiliza para almacenar números cuya característica principal es que tienen una coma flotante con precisión simple que se enmarca en 32 bits (Canal Línea de código, 2016).

Doublé

Estos datos son utilizados para el almacenamiento de números con una coma flotante la cual tiene doble precisión de 64 bits (Canal Línea de código, 2016).

Boolean

Este tipo de datos sirve para establecer datos booleanos. Es decir, aquellos que tienen un valor lógico de dos opciones de respuesta es decir de true o false. El espacio que ocupa es de 1 bit de información (Canal Línea de código, 2016).

Char

Este es un tipo de datos que está enfocado en representar un carácter Unicode de manera sencilla de 16 bits. (Canal Línea de código, 2016).

1.3.2.Datos No Primitivos

Considerando que Java es un lenguaje de POO, no resulta sorprendente que una porción significativa de su sistema de tipos de datos esté conformada por tipos complejos, es decir, objetos que derivan de clases.

La biblioteca estándar de Java consta de cientos (quizás miles) de clases que, en última instancia, representan un tipo de datos particular. Cabe señalar que los programadores pueden crear sus propios tipos de datos a partir de aquellos que componen el propio sistema de Java, pero como estos son los tipos existentes, no son parte del estándar Java y, por lo tanto, no son una parte oficial del sistema ni del lenguaje de programación. Una de las clases principales que

componen estos tipos complejos es la clase de cadena. Las cadenas no son tipos primitivos (son objetos), sino que constan de una secuencia (matriz) de caracteres (que son primitivos) que son. La clase en sí tiene un conjunto de propiedades y métodos que podemos usar para manipular nuestra cadena (Fernández, 2015).

1.4.Variables, Constantes y Operadores

1.4.1.Variables

Las variables son elementos fundamentales que se utilizan para manipular y almacenar datos. El autor IBM (2021) menciona que "Una constante es un elemento de datos con nombre con un valor predefinido" (p. 1). Es decir, las variables son espacios de almacenamiento nombrados que se utilizan para contener valores o datos en un programa. Estos valores pueden variar mientras se ejecute el programa. En muchos lenguajes de programación, las variables tienen un tipo de datos asociado que determina qué tipo de valores pueden almacenar.

Ejemplo en Python:
edad = 25
nombre = "Juan"

1.4.2. Constantes

Las constantes son valores que no cambian durante la ejecución de un programa (Blasco, 2019). Sin embargo, su valor no puede ser modificado cuando ya fue asignado. Algunos lenguajes de programación tienen palabras clave o convenciones para indicar que una variable es una constante.

Ejemplo en C++:
const int PI = 3.14;

1.4.3. Operadores

Los operadores son símbolos que representan operaciones entre variables o valores (Ceballos, 2019). Los operadores pueden realizar acciones como sumar, restar, multiplicar, dividir, comparar, asignar, etc. Dependiendo del tipo

de datos involucrados, se utilizan diferentes tipos de operadores (aritméticos, lógicos, de comparación, etc.).

Ejemplo en JavaScript:

```
let suma = 5 + 3;     // Operador aritmético (suma)
let esMayor = (10 > 5);  // Operador de comparación (mayor que)
```

Estos elementos son esenciales para la manipulación de datos y la ejecución de operaciones en cualquier lenguaje de programación.

Las variables y constantes permiten el almacenamiento y la referencia a valores, mientras que los operadores permiten realizar diversas operaciones con estos valores. La combinación de estas herramientas forma la base para la creación de algoritmos y programas.

1.5. Características Generales de la POO

En la Tabla 2 se detallan las características generalas de la programación orientada en objetos.

Tabla 2

Características Generales de la POO

Característica	Definición	Ventaja
Abstracción	Representar los conceptos del mundo real como objetos con atributos y métodos.	Facilitar la comprensión y el mantenimiento del código.
Herencia	Permitir que una clase herede los atributos y métodos de otra clase	Reduce la duplicación de código, facilita la reutilización y mejora la modularidad.
Encapsulamiento	Ocultar los detalles de implementación de los objetos.	Mejora la seguridad, la mantenibilidad y la flexibilidad.

Polimorfismo	Permitir que objetos de diferentes clases respondan de forma diferente a una misma llamada	Aumenta la flexibilidad y la reutilización del código.

Nota. Adaptado de (Ceballos, 2019)

1.6. Introducción a las Clase

"Una aplicación Java se compone de unidades llamadas clases. Y una clase en Programación Orientada a Objetos actúa como una plantilla, modelo, o prototipo, a partir de la cual se obtienen instancias, habitualmente llamadas objetos" (Blasco, 2019, p. 89). Por lo tanto, una clase es una plantilla para crear objetos. Además, la clase define un conjunto de atributos y métodos que los objetos creados a partir de esa clase tendrán. Una clase actúa como un modelo o un plano para los objetos, proporcionando una estructura que organiza la información y el comportamiento asociados.

Ejemplo en Python:

```
class Coche:

  def __init__(self, marca, modelo, color):
    self.marca = marca
    self.modelo = modelo
    self.color = color

  def mostrar_info(self):
    print(f"Coche: {self.marca} {self.modelo}, Color: {self.color}")

# Crear una instancia de la clase Coche
mi_coche = Coche(marca="Toyota", modelo="Corolla", color="Azul")
# Acceder a los atributos y llamar a un método de la instancia
print(mi_coche.marca)    # Imprime: Toyota
mi_coche.mostrar_info()  # Imprime: Coche: Toyota Corolla, Color:
Azul
```

En este ejemplo, Coche es una clase que tiene atributos como marca, modelo y color, y un método llamado mostrar_info que imprime información sobre el coche. La instancia mi_coche se crea a partir de esta clase y tiene sus propios valores para los atributos. La clase sirve como un modelo para crear objetos similares con la misma estructura y comportamiento.

1.7. Constructores de Clase

Un constructor es un método especial en una clase que se ejecuta automáticamente cuando creamos un objeto de esa clase (Ceballos, 2019). Su propósito principal es inicializar el objeto, estableciendo sus atributos o realizando otras acciones necesarias para preparar el objeto para su uso. Es como el "constructor" de un edificio, encargado de poner los cimientos y preparar la estructura. En resumen, el constructor es llamado automáticamente cada vez que creamos una instancia de la clase, asegurando que el objeto tenga un estado inicial coherente.

Se va a tomar un ejemplo en Python para ilustrar un constructor. Suponiendo que se crea una clase Persona con atributos como nombre y edad. El constructor de esta clase se llama __init__ en Python y se utiliza para inicializar los atributos se crea un objeto de la clase.

```
class Persona:

  def __init__(self, nombre, edad):
    self.nombre = nombre
    self.edad = edad

# Crear una instancia de la clase Persona usando el constructor
persona1 = Persona(nombre="Juan", edad=30)
# Acceder a los atributos del objeto creado
print(persona1.nombre) # Imprime: Juan
print(persona1.edad)  # Imprime: 30
```

En este ejemplo, el método __init__ es el constructor de la clase Persona. Cuando se crea una nueva instancia de Persona (persona1 en este

caso), el constructor se ejecuta automáticamente, inicializando los atributos nombre y edad con los valores proporcionados.

Esto asegura que cada objeto Persona tenga un estado inicial coherente.

10. Actividad de aprendizaje

Actividad de Aprendizaje 1

Introducción a la Programación Orientada a Objetos

Objetivo: Introducir a los estudiantes en los conceptos fundamentales de la Programación Orientada a Objetos (POO) y proporcionarles la oportunidad de aplicar estos conceptos en un ejercicio práctico.

Instrucciones:

1. Teoría

Investigar acerca de los principios esenciales de la Programación Orientada a Objetos, abordando conceptos como clases, objetos, abstracción, encapsulamiento, herencia, polimorfismo y modularidad.

2. Ejercicio Práctico

Implementa una clase en el lenguaje de programación de su elección (Python, Java, C++, etc.). La clase debe representar un concepto real o abstracto y contener al menos tres atributos y tres métodos. Ejemplos podrían incluir una clase "Estudiante" con atributos como nombre, edad y calificaciones, o una clase "Producto" con atributos como nombre, precio y cantidad en inventario.

3. Aplicación

Cree instancias de la clase que haya diseñado en el paso anterior y muestre cómo utilizar los métodos y acceder a los atributos.

Rúbrica de Calificación (10 puntos):

Criterio	Puntos
Teoría (Explicación de POO)	3
Ejercicio Práctico (Implementación)	5
Aplicación (Uso de instancias)	2

Actividad de Aprendizaje 2

Diseño y Desarrollo de una Aplicación Orientada a Objetos

Objetivo: Fomentar la comprensión profunda de los conceptos de clases y objetos a través del diseño y desarrollo de una aplicación práctica en un lenguaje de programación orientado a objetos.

Instrucciones:

1. Definición del Proyecto

Proponer un proyecto que requiera la implementación de al menos dos clases. Pueden elegir entre aplicaciones simples como un sistema de gestión de biblioteca, un juego sencillo, o un sistema de gestión de estudiantes. Deben describir brevemente el propósito de la aplicación y las clases que planean implementar.

2. Diseño de Clases

Elaborar un diseño detallado de las clases que planean implementar. Deben especificar los atributos, métodos y relaciones entre las clases. Utiliza diagramas UML u otros recursos visuales para representar el diseño de clases.

3. Implementación

mplementar las clases en el lenguaje de programación de tu elección. Asegúrate de seguir buenas prácticas de programación y comentarios adecuados en el código.

4. Demostración y Documentación (1 punto):

Presentar una demostración de la aplicación implementada, mostrando cómo las clases interactúan entre sí. Además, proporciona documentación breve que explique el propósito de cada clase, sus métodos y cómo utilizar la aplicación.

Rúbrica de Calificación (10 puntos):

Criterio	Puntos
Definición del Proyecto	2
Diseño de Clases (Incluyendo UML)	3
Implementación (Código)	4
Demostración y Documentación	1

Actividad de Aprendizaje 3

Comparación de Lenguajes Orientados a Objetos

Objetivo: Explorar y comparar los conceptos de programación orientada a objetos en diferentes lenguajes de programación.

Comprender cómo estos lenguajes implementan y manejan los principios fundamentales de la POO.

Instrucciones:

1. Elección de Lenguajes

Seleccionar dos lenguajes de programación orientados a objetos para la comparación. Pueden elegir entre Java, Python, C++, C#, Ruby, entre otros. Proporciona una breve justificación de la elección de estos lenguajes.

2. Implementación de Clases

Implementar una clase simple en ambos lenguajes seleccionados. La clase puede representar cualquier entidad (por ejemplo, un vehículo, una persona, etc.). Asegúrate de mostrar cómo se definen atributos y métodos en cada lenguaje.

3. Herencia y Polimorfismo

Explorar cómo se implementa la herencia y el polimorfismo en ambos lenguajes. Crea un ejemplo que demuestre estas características y compáralo en términos de sintaxis y funcionalidad.

4. Manejo de Encapsulamiento

Investigar cómo se maneja el encapsulamiento en los lenguajes seleccionados. Proporciona ejemplos que demuestren la visibilidad de miembros de la clase (públicos, privados, protegidos, etc.) en cada lenguaje.

5. Documentación y Presentación

Preparar una presentación que resuma las similitudes y diferencias encontradas en la implementación de clases, herencia, polimorfismo y encapsulamiento en los dos lenguajes. Acompaña la presentación con documentación escrita.

Rúbrica de Calificación (10 puntos):

Criterio	Puntos
Elección Justificada de Lenguajes	2
Implementación de Clases en ambos Lenguajes	4
Herencia y Polimorfismo (Comparación)	2
Manejo de Encapsulamiento (Comparación)	2
Documentación y Presentación	1

11. Autoevaluación

1. ¿Cuál es la definición correcta de Programación Orientada a Objetos (POO)?
a) Un paradigma de programación basado en instrucciones lineales.
b) Un enfoque que utiliza objetos y clases para organizar el código.
c) Un método de programación que utiliza solo funciones.
d) Una técnica para optimizar algoritmos.

2. ¿Qué característica de POO implica simplificar sistemas complejos mediante la creación de clases y objetos?
a) Modularidad.
b) Encapsulamiento.
c) Polimorfismo.
d) Abstracción.

3. ¿Cuál de los siguientes NO es un principio de Programación Orientada a Objetos?
a) Encapsulamiento.
b) Modularidad.
c) Iteración.
d) Herencia.

4. ¿Qué beneficio ofrece la herencia en Programación Orientada a Objetos?
a) Reutilización de código.
b) Mayor velocidad de ejecución.
c) Menor consumo de memoria.
d) Mayor seguridad.

5. ¿Cuál es una característica clave de las clases en Programación Orientada a Objetos?
a) Atributos y métodos.
b) Solo métodos.
c) Solo atributos.
d) Variables globales.

6. ¿Qué lenguaje de programación es ampliamente conocido por su uso en Programación Orientada a Objetos?
a) HTML.
b) JavaScript.
c) Python.
d) SQL.

7. ¿Cuál de las siguientes NO es una característica de los lenguajes orientados a objetos?
a) Herencia.
b) Encapsulamiento.
c) Procedimientos.
d) Polimorfismo.
8. ¿Cuál es el propósito principal de un constructor en Programación Orientada a Objetos?
a) Iniciar un objeto.
b) Realizar cálculos avanzados.
c) Imprimir mensajes en consola.
d) Finalizar un programa.
9. ¿Qué se entiende por polimorfismo en Programación Orientada a Objetos?
a) Capacidad de un objeto para cambiar su comportamiento en tiempo de ejecución.
b) Heredar propiedades de una clase base.
c) Agrupar datos y métodos en una unidad única.
d) Ocultar los detalles internos de una clase.
10. ¿Cuál es el término que se refiere a la ocultación de los detalles internos de una clase en Programación Orientada a Objetos?
a) Abstracción.
b) Encapsulamiento.
c) Herencia.
d) Polimorfismo.
11. ¿Cuál de los siguientes NO es un tipo de dato en Programación Orientada a Objetos?
a) Entero.
b) Flotante.
c) Objeto.
d) String.
12. ¿Cómo se llama el proceso de crear una nueva clase basada en una clase existente en Programación Orientada a Objetos?
a) Herencia.
b) Polimorfismo.
c) Abstracción.
d) Encapsulamiento.

13. ¿Cuál es uno de los beneficios clave de la Programación Orientada a Objetos?
a) Mayor complejidad del código.
b) Menor reutilización de código.
c) Mayor modularidad.
d) Menor flexibilidad.
14. ¿Qué significa el término "polimorfismo de tiempo de compilación"?
a) Cambiar el comportamiento de un objeto en tiempo de ejecución.
b) Realizar conversiones de tipo en tiempo de compilación.
c) Permite que una clase tome múltiples formas.
d) Crear instancias de una clase en tiempo de compilación.
15. ¿Qué lenguaje de programación es conocido por su uso extensivo de la Programación Orientada a Objetos y está diseñado para la plataforma Java?
a) C++.
b) Python.
c) C#.
d) Java.

12. Evaluación final

Formato de evaluación:

La evaluación constará de una combinación de preguntas de opción múltiple y problemas prácticos. La evaluación se lo realizará a través de un cuestionario que consta de 10 preguntas relacionadas con conocimientos generales y específicos de Estadística, cada pregunta tiene un valor de 1,00 punto, siendo el valor del sumatorio máximo 10 y mínimo 0.

Duración:

El tiempo asignado para completar la evaluación es de 120 minutos.
La gestión eficiente del tiempo es crucial para abordar todas las secciones.

Recursos Permitidos:

En la evaluación presencial, se permite el uso de calculadoras científicas estándar.

Instrucciones Específicas:

Lee cuidadosamente cada pregunta antes de responder.

En problemas prácticos, se valorará no solo la respuesta correcta, sino también el proceso y la justificación. Las respuestas deben ser claras y concisas.

Feedback:

Se proporcionará un resumen de resultados individual después de completar la evaluación.

13. Solucionario de las autoevaluaciones

1. ¿Cuál es la definición correcta de Programación Orientada a Objetos (POO)?
Respuesta: b) Un enfoque que utiliza objetos y clases para organizar el código.
2. ¿Qué característica de POO implica simplificar sistemas complejos mediante la creación de clases y objetos?
Respuesta: d) Abstracción.
3. ¿Cuál de los siguientes NO es un principio de Programación Orientada a Objetos?
Respuesta: c) Iteración.
4. ¿Qué beneficio ofrece la herencia en Programación Orientada a Objetos?
Respuesta: a) Reutilización de código.
5. ¿Cuál es una característica clave de las clases en Programación Orientada a Objetos?
Respuesta: a) Atributos y métodos.
6. ¿Qué lenguaje de programación es ampliamente conocido por su uso en Programación Orientada a Objetos?
Respuesta: c) Python.
7. ¿Cuál de las siguientes NO es una característica de los lenguajes orientados a objetos?
Respuesta: c) Procedimientos.
8. ¿Cuál es el propósito principal de un constructor en Programación Orientada a Objetos?
Respuesta: a) Iniciar un objeto.

9. ¿Qué se entiende por polimorfismo en Programación Orientada a Objetos?
Respuesta: a) Capacidad de un objeto para cambiar su comportamiento en tiempo de ejecución.
10. ¿Cuál es el término que se refiere a la ocultación de los detalles internos de una clase en Programación Orientada a Objetos?
Respuesta: b) Encapsulamiento.
11. ¿Cuál de los siguientes NO es un tipo de dato en Programación Orientada a Objetos?
Respuesta: c) Objeto.
12. ¿Cómo se llama el proceso de crear una nueva clase basada en una clase existente en Programación Orientada a Objetos?
Respuesta: a) Herencia.
13. ¿Cuál es uno de los beneficios clave de la Programación Orientada a Objetos?
Respuesta: c) Mayor modularidad.
14. ¿Qué significa el término "polimorfismo de tiempo de compilación"?
Respuesta: b) Realizar conversiones de tipo en tiempo de compilación.
15. ¿Qué lenguaje de programación es conocido por su uso extensivo de la Programación Orientada a Objetos y está diseñado para la plataforma Java?
Respuesta: d) Java.

14. Glosario

Abstracción: La capacidad de simplificar sistemas complejos enfocándose en los aspectos esenciales y omitiendo detalles innecesarios.
Atributo: Una variable que representa una propiedad o característica de un objeto.
Clase: Una plantilla que define la estructura y comportamiento de los objetos en POO.
Coherencia: La consistencia en la aplicación de los principios de POO a lo largo de un sistema, garantizando un diseño uniforme y claro.
Constructor: Un método especial de una clase que se llama automáticamente al crear un objeto para inicializarlo.
Encapsulamiento: El ocultamiento de los detalles internos de una clase, permitiendo el acceso controlado a sus atributos y métodos.
Herencia: El mecanismo que permite que una clase herede propiedades y comportamientos de otra, fomentando la reutilización del código.
Lenguajes Orientados a Objetos: Lenguajes de programación que admiten y facilitan los conceptos de POO, como Java, C++, Python, y C#.

Método: Una función asociada a una clase que realiza operaciones específicas sobre los datos de la clase.
Modularidad: La organización de un programa en módulos independientes, como clases y objetos, para facilitar el desarrollo y el mantenimiento.
Objeto: Una instancia específica de una clase que tiene atributos y métodos.
Polimorfismo: La capacidad de objetos de diferentes clases de responder de manera uniforme a ciertas operaciones.
Programación Orientada a Objetos (POO): Un paradigma de programación que organiza el código alrededor de objetos y clases, facilitando la reutilización y el mantenimiento del software.
Reutilización de Código: La práctica de utilizar código existente en nuevos contextos, facilitada por la herencia y la composición en POO.
Tipo de Dato: La naturaleza de los datos que una variable puede contener, como enteros, flotantes, cadenas, y también objetos en POO.

15. Referencias bibliográficas

– Blasco, F. (2019). Programación orientada a objetos en Java: (ed.). Bogotá, Ediciones de la U.
– Canal Línea de Código. (24 de marzo de 2016). Antanas Mockus en Confesiones [Archivo de Vídeo]. Youtube. https://www.youtube.com/watch?v=--UcZ6HW-3fw&list=PLLVIhySQmrVbjCFPla5c0OIp6iNWfM-hq&t=54s.
– Ceballos Sierra, F. J. (2019). Programación orientada a objetos con C++: (5 ed.). Bogotá, Ediciones de la U.
– Challenger-Pérez, I., Díaz-Ricardo, Y., & Becerra-García, R. A. (2014). El lenguaje de programación Python. Ciencias Holguín, 20(2), 1-13.
– Dadtdata. (2023), Tipos de datos. Obtenido de: https://www.datdata.com/blog/tipos-de-datos#:~:text=Los%20tipos%20de%20datos%20pueden,complejas%20como%20matrices%20o%20tablas. [2023, 5 de noviembre]
– Deitel, H. M. (2016). Java: como programar. Recuperado de: https://www.mfbarcell.es/docencia_uned/fund_inf_ing/libros/-%20Como%20Programar%20en%20Java%20Deitel.pdf [2023, 25 de noviembre]
– Díaz, J. (2016). Enseñando programación con C++: una propuesta didáctica. Revista de Informática Educativa y Medios Audiovisuales, 3(7), 12-21.
– Duarte, M. P., & Pérez, I. M. (2017). Programación en PHP a través de ejemplos. Recuperado de: https://www.cartagena99.com/recursos/tuneapdf/index.

php?archivo=programacion/apuntes/apuntes_php.pdf [2023, 20 de noviembre]
– Edteam. (2023). ¿Qué es la Programación Orientada a Objetos (POO)?. Recuperado de: https://ed.team/blog/que-es-la-programacion-orientada-a-objetos-poo. [2023, 5 de diciembre]
– Fernández, O. B. (2015). Introducción al lenguaje de programación Java. Una guía básica, 9. Recuperado de: https://www.manualweb.net/java/introduccion-java/ [2023, 12 de noviembre]
– García-Peñalvo, F. J., & Pardo Aguilar, C. (2018). Introducción al Análisis y Diseño Orientado a Objetos. Recuperado de: https://repositorio.grial.eu/bitstream/grial/265/1/ADOO.pdf [2023, 28 de noviembre]
– IBM. (2021). Constantes y Variables. Recuperado de: https://www.ibm.com/docs/es/tcamfma/6.3.0?topic=tesl-constants-variables-4 [2023, 10 de diciembre]
– Javatpoint. (2021). Conceptos de POO en Java. Recuperado de: https://www.javatpoint.com/pt/conceitos-de-poo-em-java [2023, 12 de diciembre]
– Minguet, A. R. E. (2013). Extensiones al Lenguaje Ada y a los Servicios Posix para Planificación en Sistemas de Tiempo Real Estricto. Recuperado de: https://riunet.upv.es/handle/10251/17743 [2023, 15 de diciembre]
– Miños Fayad, A. (2017). Elementos estructurantes de la Didáctica de la Informática. Virtualidad, Educación y Ciencia. Recuperado de: https://revistas.unc.edu.ar/index.php/vesc/article/view/17337 [2023, 5 de noviembre]
– Munguía, P. P. (2017). Ada-CCM: Tecnología de componentes basada en el lenguaje de programación Ada 2005. Recuperado de: https://www.istr.unican.es/publications/academic/ppm-jmd-2007a.pdf [2023, 17 de noviembre]
– Pérez, H. D. J. C., & Esquivel, J. A. A. (2007). Ruby: lenguaje de programación para sistemas distribuidos. Conciencia Tecnológica, (33), 81-83. Recuperado de: https://www.redalyc.org/pdf/944/94403319.pdf [2023, 18 de noviembre]
Pérez, R. (2005). Rubí: sencillo y versátil. De la idea al código. Mundo Linux: Sólo programadores Linux, (81), 52-57. Recuperado de: https://dialnet.unirioja.es/servlet/revista?codigo=8138 [2023, 18 de diciembre]
– Sánchez, D. (2016). Lenguaje de programación C#. Recuperado de: https://www.academia.edu/28564115/Lenguaje_de_programaci%C3%B3n_C_ [2023, 7 de noviembre]
– Troya Torres, C. F. (2019). Aula Virtual en moodle para el aprendizaje del lenguaje de programación en bachillerato técnico.

16. Anexos o recursos

Los recursos que se utilizarán para mejorar el proceso enseñanza aprendizaje son los siguientes:

– https://www.javatpoint.com/pt/conceitos-de-poo-em-java
– https://www.youtube.com/watch?v=DlphYPc_HKk
– https://www.youtube.com/watch?v=SI7O81GMG2A
– https://www.youtube.com/watch?v=uNlB7141umY
– https://kataix.umag.cl/~ruribe/Utilidades/Introduccion%20a%20la%20Programacion%20Orientada%20a%20Objetos.pdf

Banco de imágenes libres

https://unsplash.com/

INSTITUTO SUPERIOR
TECNOLÓGICO
VICENTE LEÓN

Guía general de estudio de la asignatura

Agosto 2024

ISBN: 978-9942-676-39-9