

Programación Orientada a
Objetos. Clases. Herencia.
Polimorfismo. Lenguajes.

TEMA 27 (29 SAI)

ABACUS NT

Oposiciones 2021

Índice

- 1. Introducción.**
- 2. Programación orientada a objetos**
 - 2.1. Características**
 - 2.2. Ventajas**
- 3. Clases**
 - 3.1. Definición de clase**
 - 3.2. Miembros de una clase (encapsulación)**
 - 3.3. Constructores y Destructores**
 - 3.4. Niveles de acceso a los componentes de una clase.**
 - 3.5. Objetos**
 - 3.5.1. Estructura**
 - 3.5.2. Características**
- 4. Herencia**
 - 4.1. Definición**
 - 4.2. Tipos de Herencia:**
 - 4.3. Funcionalidades de la herencia:**
- 5. Polimorfismo**
 - 5.1.1. Sobrecarga de método**
 - 5.1.2. Sobrecarga de operadores**
 - 5.1.3. Polimorfismo en la herencia**
- 6. Lenguajes Orientados a Objetos**
 - 6.1. Java**
 - 6.2. Lenguajes más utilizados en el sector productivo**
- 7. Conclusión**
 - 7.1. Relación del tema con el sistema educativo actual**
- 8. Bibliografía**

1. Introducción.

La Programación Orientada a Objetos (en adelante POO) se ha convertido en **el nuevo paradigma** de la programación. De hecho, no hay libro de POO que no utilice la palabra “paradigma” para referirse a la orientación a objetos. Algunos autores, no sin razón afirman que es más fácil aprender a programar orientación a objetos desde cero que reconvertir a un programador que no haya nunca programado nunca POO.

Paradigma del griego **παράδειγμα** (traducido como ejemplo), se refiere a una teoría o conjunto de teorías que sirve de modelo a seguir para resolver problemas o situaciones determinadas que se planteen.

En la programación estructurada -un paradigma clásico, **los datos y las operaciones permanecen claramente separados**. Los **módulos** de nuestros programas se iban pasando de unos a otros los datos que necesitaban a través de **parámetros**.

En la POO tanto **los datos como el código se agrupan formando un módulo que denominamos clase**. Esta abstracción es más **parecida al mundo real**, en el que las “cosas” son objetos con los que interactuamos y **no módulos parametrizados**.

2. Programación orientada a objetos

Según el autor **J.Glenn Brookshear**, en su libro “Introducción a la computación (2012)”, en este paradigma el sistema software se ve conceptualmente como un *conjunto de unidades, denominadas objetos, los cuales son capaces de llevar a cabo las acciones que le afectan directamente, así como de solicitar acciones a otros objetos*.

La programación orientada a objetos es un paradigma de programación que consiste en **modelar la realidad como un conjunto de objetos** que **interactúan** entre sí para resolver un problema.

2.1. Características

Las **características básicas comunes** que presentan los lenguajes orientados a objetos son:

- **Encapsulamiento.** Las propiedades y el comportamiento de los objetos están definidos en las clases, formadas por estructuras de datos y algoritmos denominadas atributos y métodos. Un objeto es una instancia de una determinada clase y tiene su propio conjunto de datos. Los métodos se aplican a un objeto concreto y son propias de la clase a la que pertenece.
- **Ocultación.** Para aumentar la modularidad y como medida de protección, un objeto se considera como una caja negra en la que se puede introducir o de la que se puede extraer información sin necesidad de conocer su funcionamiento interno.

- **Envío de mensajes.** Consiste en nombrar y activar con los parámetros adecuados uno de los métodos del objeto al que se envía el mensaje. Es la manera que tienen los objetos de relacionarse, y es equivalente a las llamadas a procedimientos de los lenguajes imperativos.
- **Herencia.** Es un mecanismo mediante el cual pueden crearse clases a partir de otras, transmitiendo todos los atributos y todos los métodos de clases padres a clases hijas, con la ventaja de que en estas últimas pueden añadirse más atributos y métodos.
- **Polimorfismo.** Es la capacidad que tiene un objeto de una determinada clase de hacerse pasar por un objeto de una clase ancestro. Esta característica, junto a la posibilidad de las clases hijas de redefinir los métodos heredados, permiten la reutilización de código.

2.2. Ventajas

La razón de que la P.O.O goce de tanta **popularidad** actualmente es, precisamente por esto, porque los programas se encuentran conformados por unidades aisladas y bien definidas, lo que deriva en los siguientes beneficios:

- Favorece la reutilización,
- Aumento del nivel de abstracción
- Acelerar la fase de desarrollo
- Facilitar el mantenimiento
- Permitir la creación de sistemas complejos
- Proteger la información

La POO también tiene ciertas desventajas respecto a la programación clásica:

- Tiempo: El tiempo de ejecución del mecanismo de vinculación dinámica (polimorfismo) es lento.
- Complejidad: La implementación requiere realizar un análisis más exhaustivo de su aplicación
- Curva de aprendizaje: Es necesario conocer las librerías de clase que los lenguajes incorporan

3. Clases

3.1. Definición de clase

Una clase es un modelo a partir del cual se pueden construir tantos elementos como se quiera; a esos elementos los llamamos objetos de la clase. Las clases por tanto, no son utilizables directamente, sólo sirven para crear instancias u objetos de la clase.

Dos instancias (objetos) diferentes de una misma clase comparten el mismo conjunto de procedimientos y la misma lista de datos, pero con valores diferentes.

Una **clase** es un concepto abstracto que sirve para definir una **generalización de un tipo** determinado de objetos mediante una plantilla. Se caracterizan por:

- Un **identificador**
- Unos **componentes**
- Definición del **nivel de acceso** de cada componente.
- **Relación de la clase con otras clases** existentes

Un **ejemplo** de declaración de una clase en Java podría ser:

```
public class Rectangulo{  
    //atributos  
    private int base;  
    private int altura;  
    ...  
    //Métodos  
    public int getArea () {  
        return base*altura;  
    }  
    ...  
}
```

3.2. Miembros de una clase (encapsulación)

Los **miembros de una clase** o componentes que conforman la clase son: los atributos, los métodos y los objetos:

1. **Objetos** Un **objeto** es una **instancia de una clase**. Se dice que un objeto es **miembro de una clase** cuando es una instancia de la misma.

2. **Atributos:** Determina los tipos de datos que se van a guardar de cada objeto, así los objetos se diferenciarán unos de otros en función del valor que tenga para dichos datos. En el ejemplo anterior, base y altura serían los atributos de la clase.
3. **Métodos:** Son los **procedimientos de la clase** que pueden ser llamados por los objetos para realizar operaciones con sus datos. Los tipos de procedimientos que podemos encontrar son:
 - **Observadores:** Sirven para obtener información del objeto.
 - **Modificadores:** Son utilizados para modificar datos de los objetos.
 - **Constructores:** Son utilizados para asignar valores a los datos de un objeto en el momento de su creación.
 - **Destructores:** Son ejecutados cuando un objeto va a ser eliminado.

3.3. Constructores y Destructores

Todas las clases disponen de métodos para crear y destruir objetos llamados constructores y destructores.

En general los constructores son unas funciones miembro que se llaman de modo automático al crear los objetos y dan un valor inicial a cada una de sus variables miembro. Se pueden definir varios tipos de constructores para una clase:

- **Constructor por defecto:** no necesita parámetros para inicializar las variables miembro o si los tiene, todos sus argumentos tienen asignado un valor por defecto en la declaración del constructor.
- **Constructor de oficio:** si una clase no tienen definido un constructor, el compilador crea un constructor de oficio sin argumentos.
- **Constructores de copia:** Se utiliza cuando se crea un objeto inicializado a partir de otro objeto de la misma clase.

Es necesario también un método que destruya el objeto cuando sea necesario llamado **destructor**. El destructor que se encarga de eliminar el objeto del sistema es siempre único (no puede estar sobrecargado), no tienen argumentos ni tampoco valor de retorno.

3.4. Niveles de acceso a los componentes de una clase.

Uno de los problemas que aparecen con la herencia es el del control del acceso a los datos. ¿Puede una función de una clase derivada acceder a los datos privados de su clase base? En principio, una clase no puede acceder a los datos privados de otra, pero podría ser muy conveniente que una clase derivada accediera a todos los datos de su clase base.

Los componentes de una clase suelen tener asociados un nivel de acceso o visibilidad, el cual determina desde donde se puede acceder al componente, encapsulándolo.. Normalmente, se diferencian los siguientes niveles:

Privado: El componente puede ser accedido únicamente desde los métodos de su propia clase.

Público: El componente es accesible desde cualquier otra clase o subclase.

Protegido: El componente puede ser accedido únicamente desde los métodos de su propia clase y de las clases derivadas (o clases hijas).

3.5. Objetos

Existe una diferencia clara entre objeto y clase: la clase es una abstracción, donde se intenta definir sus características generales. Los objetos son entidades concretas que existen en un tiempo y espacio, que tienen unos atributos determinados, pero con un comportamiento común entre ellos.

Cada objeto conoce a la clase a la que pertenece y la mayoría de los lenguajes orientados a objetos permiten determinar en tiempo de ejecución la clase a la que pertenece un determinado objeto.

3.5.1. Estructura

Cada objeto tiene dos partes diferenciadas:

La interfaz: es pública, conocida por los demás objetos del sistema, y es el resultado de aplicar la abstracción al objeto. Está formado por los mensajes que el objeto entiende.

La implementación: es la parte oculta, resultado de aplicar la encapsulación al objeto en cuestión. Contiene la especificación de los métodos y los atributos o propiedades del objeto.

3.5.2. Características

Un objeto se caracteriza por las siguientes propiedades:

Encapsulamiento: Consiste en integrar en un objeto métodos y propiedades; de este modo los mantiene protegidos frente a cualquier interferencia y mal uso. Es un mecanismo de seguridad.

Ocultación: Los objetos son como cajas negras, el usuario no necesita saber cómo funciona el objeto sino como usarlo. Esta propiedad de los objetos se denomina ocultación de la información. Consiste en separar la implementación interna de un objeto, de su uso.

Mensajes: Durante la ejecución de un programa, los objetos pueden recibir mensajes, los cuales consisten en realizar una llamada a un método de la clase a la que pertenece. Así, un ejemplo con la clase anterior, sería:

```
int area=base.Casa.area();
```

4. Herencia

4.1. Definición

Qué

La herencia consiste en la compartición de componentes entre clases, de tal forma que una clase hija (o clases derivadas o subclases) puede heredar **componentes** de una clase padre (o clase base o superclase), estableciéndose, de esta forma, una organización jerárquica entre clases. En algunos lenguajes, como por ejemplo C++, una clase puede heredar de varias clases (**herencia múltiple**).

Cómo

La herencia, permite definir una clase modificando permitir definir una clase **modificando** una o más clases ya existentes. Estas modificaciones consisten habitualmente en añadir nuevos miembros (variables o funciones), a la clase que se está definiendo, aunque también se puede definir variables o funciones miembros ya existentes. Esto se puede lograr principalmente mediante los mecanismos de **especialización y extensión** (que se verán más adelante).

Quién

Clases base y clase derivada

La clase de la que parte en este proceso recibe el nombre de **clase base**, y la nueva clase que se obtiene se denomina **clase derivada**. También pueden ser llamadas **superclase o clase padre** y **subclase o clase hija** respectivamente. Ésta a su vez puede ser clase base en un nuevo proceso de derivación, iniciando de esta manera una **jerarquía** de clases. Las clases base suelen ser más generales que las clases derivadas. Esto es así porque a las clases derivadas se les suelen ir añadiendo características, en definitiva, variables y funciones que se diferencian concretan y particularizan.

Clases ABC

En algunos casos, una clase no tiene otra utilidad que ser clase base para otras clases que se deriven de ella. A este tipo de clases base, de las que no se declara ningún objeto, se les denomina clases base abstractas (**ABC, Abstract Base Class**) y su función es la de agrupar miembros comunes de otras clases que se derivarán de ellas. Por ejemplo, se puede definir la clase vehículo para después衍生 de ella coche, bicicleta, patineta, etc., pero todos los objetos que se declaran pertenecerán a alguna de estas últimas clases; no habrá vehículos que sean sólo vehículos.

Porqué

Este mecanismo de herencia presenta múltiples ventajas, siendo la principal la **reusabilidad del código**. Otras ventajas son:

- Evitar **duplicidad** y favorecer la reutilización de código (las subclases utilizan el código de superclases).

- Facilitar el **mantenimiento** de aplicaciones. Podemos cambiar las clases que usamos fácilmente.
- Facilitar la **extensión** de las aplicaciones.

4.2. Tipos de Herencia:

1. **Herencia simple:** Cuando cada clase no puede tener más que un sólo padre.
2. **Herencia múltiple:** Si una clase tiene el derecho de poseer varios padres. Esta propiedad de herencia múltiple nos va a permitir modelar los objetos del mundo real cuando se tienen varias visiones de este objeto. Por ejemplo, podemos tener un clase automóvil con tres padres: clase vehículo, clase producto de tecnología y clase máquina.
3. **Herencia repetida:** Esta situación se produce cuando las características de dos clases que son herederas comunes de una tercera clase son utilizadas para definir una nueva clase.
4. **Herencia selectiva:** Es una clase que no hereda más que una parte de las propiedades de una clase existente. Es una protección contra el riesgo de una mal definición: si no se conoce con suficiente detalle la clase de la que se hereda, se puede crear una nueva clase restringiendo la herencia a las variables y métodos conocidos.

4.3. Funcionalidades de la herencia:

1. **Especialización:** Tenemos una clase A y otra B, que es una especialización de A: Un B es una A: por ejemplo: Un coche es un vehículo.
2. **Extensión:** Una clase puede extender la funcionalidad proporcionada por la clase base, sin representar necesariamente un concepto más específico: Por ejemplo, la clase lista Salvable es una lista enlazada que además contiene una operación para salvar a disco y recuperar la lista en un fichero.
3. **Especificación:** Una clase puede servir para especificar la funcionalidad mínima común que debe verificar un conjunto de clases. Por ejemplo, los objetos gráficos en un programa de diseño línea, texto y cuadrado tienen funcionalidades mínimas comunes como el cambiar el color.
4. **Construcción:** Una clase puede servir simplemente para construir otra, porque la clase hija necesita toda o parte de su funcionalidad internamente, aunque representen conceptos radicalmente distintos: por ejemplo, la clase pila hereda de la clase lista.

5. Polimorfismo

Según el autor Luis Joyanes Aguilar, en su libro **Fundamentos de programación. Algoritmos, estructuras de datos y objetos (2008)**, “el **polimorfismo** es la propiedad que permite a una

operación tener el mismo nombre (**sobrecarga**) en clases diferentes y que actué de modo diferente en cada una de ellas". Los tipos que podemos encontrar son:

5.1.1. Sobrecarga de método

Se produce **cuando dos métodos se denominan igual**. A su vez, se pueden distinguir los siguientes **subtipos**:

- **Sobrecarga de métodos pertenecientes a clases distintas**: En este caso el compilador sabe cuál es el método que tiene que ejecutar, al tener en cuenta el objeto al que se le envía el mensaje (es el ejemplo al que se refería el autor Luis Joyanes).
- **Sobrecarga de métodos pertenecientes a una misma clase**: En este caso el compilador discrimina el método que debe utilizar en función de los parámetros que emplean los métodos sobrecargados.

5.1.2. Sobrecarga de operadores

Algunos lenguajes, como C++, permiten redefinir el comportamiento de determinados operadores, como por ejemplo "+", **pudiéndose utilizar, por tanto, un mismo operador para realizar distintas tareas**.

5.1.3. Polimorfismo en la herencia

En este caso lo que se desea es sustituir o sobrescribir un método de la clase padre por un método de la clase hija. En Java esto se indica con la anotación "**@overwrite**", y la máquina virtual, en tiempo de ejecución, decide qué método se va a ejecutar, mediante la ligadura dinámica.

6. Lenguajes Orientados a Objetos

Las técnicas de POO han hecho que los lenguajes de programación tradicionales hayan evolucionado para poder desarrollar estas técnicas de programación (Como Pascal, C e incluso Cobol). Por otro lado, se han desarrollado Smalltalk, Eiffel, Delphi, Java, etc. que son lenguajes diseñados específicamente para desarrollar POO.

Los lenguajes orientados a objetos se pueden dividir en **dos grupos**:

- **Lenguajes puros**: Son lenguajes en los que **únicamente se puede trabajar con objetos**. Como ejemplo tenemos Smalltalk, o Ruby.
- **Lenguajes híbridos**: Son lenguajes que permiten utilizar otro tipo de datos diferente a los objetos. Como ejemplo tenemos Java, o C#.

6.1. Java

Características principales.

Java es un lenguaje basado en la sintaxis de C++ que sólo admite la creación de programas a partir de clases (Orientado a objetos puro). Fue propuesto por Sun en 1995 y es un lenguaje compilado e interpretado de propósito general que soporta la programación concurrente y elimina la aritmética de punteros.

Entre sus características destacamos:

- Sintaxis similar a C++
- Dispone de un recolector de basura
- Elimina el uso de punteros, utilizando referencias
- Soporta programación multihilo
- Dispone de mecanismos de detección de errores
- Tiene amplias librerías de clases

Clases, objetos, métodos y mensajes.

En Java no existen las estructuras (struct) ni los punteros. Las clases están formadas por atributos y métodos, cuyo acceso puede ser public, protected y private, como en C++.

Además de clases existen paquetes (packages) que agrupan una gran cantidad de clases. Si se quiere utilizar una clase situada dentro de un paquete se puede importar el paquete (import).

No existen funciones amigas, y los objetos son creados implícitamente mediante constructores definidos con el mismo nombre de la clase, los cuales realizan la inicialización de los atributos antes de que los objetos sean creados. También pueden crearse explícitamente utilizando *new*.

Los objetos siempre se refieren mediante variables del tipo objeto.

No existen destructores en Java, se realiza una recolección automática de memoria. Existe un método *finalize* para casos especiales en los que se asigna memoria por un procedimiento distinto al normal (*new*). Este método se invoca justo antes de la recolección de memoria.

Para llamar un método, se utiliza el identificador del objeto y '.' y los parámetros de los métodos siempre se pasan por valor excepto los objetos, que se pasan por referencia automáticamente.

Herencia y polimorfismo.

En Java existe la herencia de tipos y la herencia de implementación, y está permitida solamente la herencia simple. Existe una clase raíz denominada Object.

La herencia no cambia el nivel de protección de los miembros de la clase base. Como en C++, los constructores no pueden ser heredados. Se puede incluir una llamada explícita con la palabra super

al constructor de la clase padre como primer enunciado del constructor de la subclase. En otro caso, se llamará implícitamente al constructor por omisión.

Una interfaz es el nombre que se le da a un conjunto de métodos que debe definir una clase que lo implemente, con independencia de su relación jerárquica, y no incluye ningún detalle de implementación. Se dice que una clase implementa una interfaz si define todos los métodos abstractos de la interfaz. Las interfaces son las clases más abstractas que existen, puesto que consisten sólo en métodos públicos y abstractos, y atributos públicos y finales (constantes).

Los métodos abstractos son métodos que no tienen implementación y que obligan a ser implementados en las subclases. Si una clase posee un método abstracto es una clase abstracta, de la cual no se pueden instanciar objetos. Si una clase tiene una superclase abstracta y no implementa algunos de sus métodos abstractos, entonces los hereda y se convierte en abstracta. Para poder crear objetos de esa clase, es necesario implementar todos los métodos abstractos heredados. Una clase se hace abstracta declarándola con la palabra clave *abstract*.

En Java el método que tiene el mismo nombre que un método de la superclase anulará siempre automáticamente el método de la superclase, no hay que indicarlo explícitamente. Para referirse explícitamente a la función de la superclase se utiliza la palabra *super* (*super.toString()*).

Por defecto, se utiliza la ligadura dinámica para determinar qué método se aplica a un determinado objeto. Si se utiliza una referencia de superclase para referirnos a un objeto de subclase, el programa escogerá el método de la subclase correcta en tiempo de ejecución mediante la búsqueda ascendente entre clases, comenzando en la clase del receptor, para encontrar un método que responda a un mensaje. Si se alcanza la última superclase sin encontrar ningún método, se emite un mensaje de error.

6.2. Lenguajes más utilizados en el sector productivo

Según la publicación “**2019 The top Programming Languages**” del IEEE Spectrum (el espectro del Instituto de ingeniería eléctrica y electrónica), de los diez lenguajes con más importancia en el sector productivo, 9 son multiparadigma y/o orientados a objetos, lo que muestra la gran importancia de la P.O.O actualmente. Estos lenguajes son:

1. **Python:** Desarrollado por Guido Van Rossum que liberó su código en 1991 bajo licencia Python License originalmente y GNU GPL en la actualidad, es un lenguaje de programación interpretado, multiplataforma, que soporta P.O.O, programación imperativa y funcional. Hace hincapié en la legibilidad del código, utilizando un tipado dinámico y conteo para el administrador de memoria. Debe su nombre a la aficción de su desarrollador por los Monthly Python.
2. **Java:** Fue desarrollado en 1985 y está enfocado hacia la red, debido al auge de Internet en los años 90. Es independiente del ordenador donde se ejecute y para conseguir esta

independencia utiliza una máquina virtual java (JVM) que nos permite utilizar código compilado en otra computadora distinta.

3. **C:** Desarrollado en 1972 a partir de otro lenguaje llamado B, con la idea de conseguir un lenguaje de alto nivel, pero con posibilidad de acceso a bajo nivel. Esta propiedad hace que este lenguaje sea muy adecuado para la programación de sistemas.

Es adecuado para programadores expertos porque hace programas muy eficientes, por el contrario, su depuración puede ser bastante compleja. Tiene las siguientes características:

- Es débilmente tipado
 - Impone pocas restricciones al programador
 - Dispone de operadores a nivel de bit
 - Soporta el uso de punteros
 - Utiliza librerías
4. **C++:** Fue inventado en los laboratorios de AT&T en 1980. Inicialmente fue una gran mejora de C. Actualmente tiene un estándar, denominado ISO C++.

C++ soporta programación orientada a objetos incluyendo características como:

- Implementación de clases
- Herencia múltiple
- Acceso a los atributos y métodos

5. R:

Es un entorno y lenguaje de programación con un enfoque al análisis estadístico.

Se trata de uno de los lenguajes de programación más utilizados en investigación científica, junto con Matlab, siendo además muy popular en los campos de aprendizaje automático (machine learning), minería de datos, investigación biomédica, bioinformática y matemáticas financieras. R es orientado a objetos, se puede integrar con bases de datos, tiene numerosas funciones de cálculo y funciones gráficas compatibles con LaTeX.

6. Javascript:

Es un lenguaje de programación interpretado, dialecto del estándar ECMAScript. Se define como orientado a objetos, basado en prototipos, imperativo, débilmente tipado y dinámico.

Se utiliza principalmente en su forma del lado del cliente (client-side), implementado como parte de un navegador web permitiendo mejoras en la interfaz de usuario y páginas web

JavaScript se diseñó con una sintaxis similar a C, aunque adopta nombres y convenciones del lenguaje de programación Java. Sin embargo, Java y JavaScript tienen semánticas y propósitos diferentes.

Todos los navegadores modernos interpretan el código JavaScript integrado en las páginas web. Para interactuar con una página web se provee al lenguaje JavaScript de una implementación del Document Object Model (DOM).

Actualmente es ampliamente utilizado para enviar y recibir información del servidor junto con ayuda de otras tecnologías como AJAX.

7. C#:

Pronunciado C sharp, es un lenguaje de programación multiparadigma desarrollado y estandarizado por Microsoft como parte de su plataforma .NET, de los lenguajes de programación diseñados para la infraestructura de lenguaje común (CLI).

Su sintaxis básica deriva de C/C++, utiliza el modelo de objetos de la plataforma .NET, similar al de Java, aunque incluye mejoras derivadas de otros lenguajes.

8. Matlab:

Matrix Laboratory es un sistema de cómputo numérico que ofrece un IDE y lenguaje de programación llamado M que es interpretado y orientado a objetos. Este lenguaje permite operaciones de vectores y matrices, funciones, cálculo lambda y representación de gráficos 2D y 3D.

9. Swift

Swift es un lenguaje de programación **multiparadigma** creado por **Apple** enfocado en el desarrollo de **aplicaciones para iOS y macOS**. Fue presentado en la WWDC 2014 y puede usar cualquier biblioteca programada en Objective-C y llamar a funciones de C.

10. Go

Desarrollado por Google en 2009, la concurrencia, la sencillez, tipado estático, y el no utilizar excepciones son las principales características de Go. Es el lenguaje de programación de sistemas multiproceso más eficaz existente en la actualidad, inspirado en la sintaxis de C. Soporta orientación a objetos parcialmente, ya que no implementa mecanismos de herencia y las clases son declaradas como componentes separados de Interfaces y estructuras.

7. Conclusión

La orientación a objetos nace con la intención de modelar el mundo real y sus interacciones en una forma muy efectiva de crear modelos elegantes, correctos y funcionales.

Algunas ideas de la programación a objetos, tales como la encapsulación, el polimorfismo y la herencia son simplemente brillantes y por tanto son susceptibles de ser adaptados por nuevos paradigmas en el futuro.

Sin embargo, la POO ha sido comparada a menudo con “matar moscas a cañonazos” por la complejidad que implica su desarrollo.

La POO modela el mundo real, y es en efecto tan buena en eso que en seguida fue fácilmente aceptada a la hora de crear software para empresas (enterprise software) donde los complejos procesos de trabajo y estructuras de datos podían ser modeladas mucho más fácilmente.

Actualmente numerosos lenguajes de programación admiten orientación a objetos, lo que implica que es altamente utilizada, pero también es cierto que muchos de estos lenguajes dan la posibilidad de no utilizarla (lenguajes híbridos), lo cual quiere decir que no siempre es una solución óptima.

7.1. Relación del tema con el sistema educativo actual

Este tema puede ser desarrollado en los siguientes módulos formativos (para atribución docente de PES)

- Bachillerato – Tecnologías de la Información y la Comunicación II (PES)
- GS- GS – DAW – Desarrollo Web en Entorno Servidor DAW/DAM – Programación (PES)

Aunque los profesores de SAI no tienen una atribución docente en ningún módulo con aplicación directa del tema, siempre es probable que se diseñe algún pequeño programa en clase, para lo que recurrir a la notación de pseudocódigo o de ordinograma sería un recurso esclarecedor.

8. Bibliografía

- Introduction to Java Programming and Data Structures, Comprehensive Version. Y. Daniel Liang. Pearson, 11^a edición. 2017.
- Java 9. Francisco Javier Moldes Teo. Anaya. 2017.
- Introducción a la computación. J, Glenn Brookshear, Pearson, 11^º edición. 2012
- Fundamentos de programación. Algoritmos, estructuras de datos y objetos. Luis Joyanes Aguilar, McGraw-Hill, 4^º edición. 2008.
- Langsam, Augenstein y Tanembaum: “Estructuras de Datos con C y C++”, Prentice-Hall 1997
- Prieto A., Lloris A. y Torres J.C.: Introducción a la Informática, 4^a ed (2006) McGraw-Hill
- Lenguajes de programación, principios y práctica. 2^a Ed (2004). Kenneth C.Louden

