TALLER DE PROGRAMACIÓN

Diseño y Programación orientada a objeto

Taller de Programación

Diseño y Programación orientada a objeto

ESCUELA DE CONSTRUCCIÓN E INGENIERÍA
Director de Escuela: Marcelo Lucero Yañez.

ELABORACIÓN
Experto disciplinar: Cristian Gandolfi.
Diseñador instruccional: Hernán Apablaza.

VALIDACIÓN
Experto disciplinar: Ricardo Olivares.
Jefa de Diseño Instruccional y Multimedia: Alejandra San Juan Reyes.

DISEÑO DOCUMENTO
Diseño Instruccional AIEP

Contenido

APRENDIZAJE ESPERADO

Analizan diseño orientado a objeto, considerando proceso de desarrollo de software.

3. METODOLOGÍA ORIENTADA A OBJETO

Una metodología orientada a objetos es un proceso para producir software de una manera organizada (usando convenciones) y técnicas de notación predefinidas.

Desde que la comunidad de programación orientada a objetos tuvo la noción de incorporar el pensamiento de que los objetos son entidades coherentes con entidad estado y conducta, estos objetos pueden ser organizados por sus similitudes y diferencias (herencia y polimorfismo).

Las metodologías orientadas a objetos incorporan estos conceptos para definir sus reglas, normas, procedimientos, guías y notaciones, para alcanzar un producto de calidad que satisfaga las necesidades del cliente. (Romero, 2015, p.1)

El lenguaje UML sirve para especificar, visualizar y documentar esquemas de sistemas de software orientados a objetos.

Como ya se ha revisado durante la semana anterior, UML no es un método de desarrollo, no permite como determinar que hacer o cómo diseñar el sistema, sino que simplemente le ayuda a visualizar el diseño y hacerlo más visible a otros. UML esta diseñado para ser usado con software oriento a objetos, contiene muchos elementos de esquematización que presentan las diferentes partes de un sistema de software. Así como lo indica el autor, los elementos UML se utilizan para crear diagramas, que representa alguna parte o punto de vista del sistema.

3.1. CONCEPTOS

Una instancia de una clase es otro término para un objeto real, o en otras palabras es la representación en memoria de una clase. La clase es una representación general de un objeto, una instancia es la representación concreta de objetos que se generen de acuerdo con esta.

A menudo se utiliza indistintamente la palabra objeto o instancia para referirse, precisamente a un objeto. En los lenguajes orientados a objetos, cada clase esta compuesta de dos cualidades, atributos y métodos. Los atributos determinan las características individuales que diferencian a un estado de otro, determinando con ello su apariencia, estado y otras cualidades de un objeto. Los atributos de un objeto incluyen información sobre su estado.

Los métodos de una clase determinan el comportamiento o conducta que requiere esta clase para que sus instancias puedan cambiar su estado

Los métodos de una clase determinan el comportamiento o conducta que requiere esta clase para que sus instancias puedan modificar su estado interno o cuando dichas instancias son llamadas para realizar algo por otra clase o instancia. (Academia EDU, 2015)

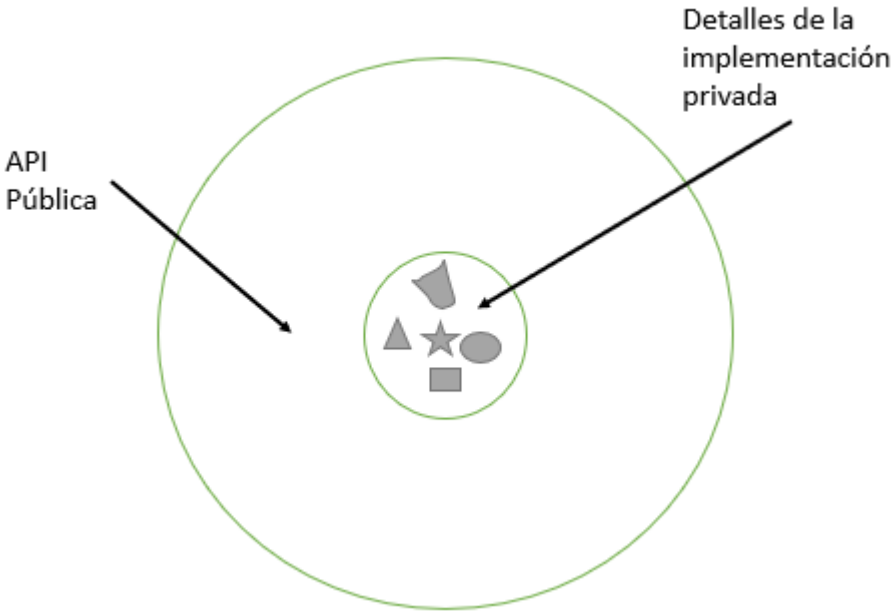

Figura 1. Implementaciones POO.
Fuente: Gandolfi, C. (2020) Módulo Taller de Programación, AIEP.

Para definir el comportamiento de un objeto, se crean métodos, los cuales tienen una apariencia y un comportamiento igual a las funciones en otros lenguajes de programación, la diferencia es que estos se escriben dentro de una clase. Los métodos no siempre afectan a un objeto; los métodos permiten además la comunicación entre objetos. Los métodos rodean y esconden el núcleo de un objeto, a esto también se le denomina empaquetamiento o encapsulamiento.

```
//Funcion suma recibe dos numeros enteros
//y retorna la suma de estos
int suma(int a, int b)
{
    return a + b;
}

//Llamada a la función suma enviando dos valores
Console.WriteLine(suma(1, 2));
Console.Read();
```

Figura 2. Uso de una función en C#.
Fuente: Gandolfi, C. (2020) Módulo Taller de Programación, AIEP.

```
//nombre de clase
2 references
class Operaciones
{
    //descripción de metodo que recibe dos
    //valores enteros como parametros y retorna
    //la suma de estos
    1 reference
    public int suma(int a, int b)
    {
        return a + b;
    }
}
}
```

Figura 3. Representación de una clase con un método en C#
Fuente: Gandolfi, C. (2020) Módulo Taller de Programación, AIEP.

El encapsulamiento de variables y métodos en un componente de software ordenado es, todavía, una simple idea poderosa que provee dos principales beneficios en los desarrollos de software:
Modularidad: Esto es, el código fuente de un objeto puede ser escrito, recibir mantenimiento, independiente del código fuente de otros objetos. Así mismo un objeto puede ser transferido alrededor del sistema sin afectar su conducta.

Ocultamiento de Información: un objeto tiene una interfaz pública que otros objetos pueden utilizar para comunicarse con él. Pero el objeto puede mantener información y métodos privados que pueden ser cambiados en cualquier tiempo sin afectar a los otros objetos que dependan de ello.
Los objetos proveen el beneficio del modularidad y el ocultamiento de la información. Las clases proveen el beneficio de la reutilización. Los programadores de software utilizan la misma clase, y por lo tanto el mismo código, una y otra vez para crear muchos objetos.

Otro concepto muy importante en la metodología orientada a objetos es el de herencia. La herencia es un mecanismo poderoso con el cual se puede definir una clase en términos de otra clase, significa que cuando se escribe una clase, solo se tiene que especificar la diferencia de esa clase con otra, con lo cual, la herencia dará acceso automático a la información contenida en esa otra clase. (Academia EDU, 2015)
Con la herencia todas las clases están arregladas dentro de una jerarquía estricta. Cada clase tiene una superclase (la clase superior en la jerarquía) y puede tener una o más subclases.

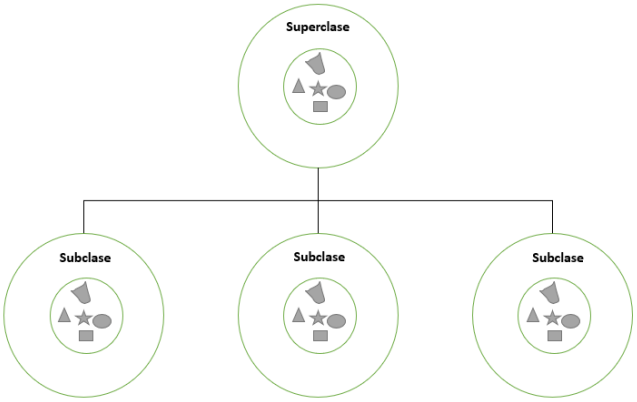

Figura 4. Herencia de clases
Fuente: Gandolfi, C. (2020) Módulo Taller de Programación, AIEP.

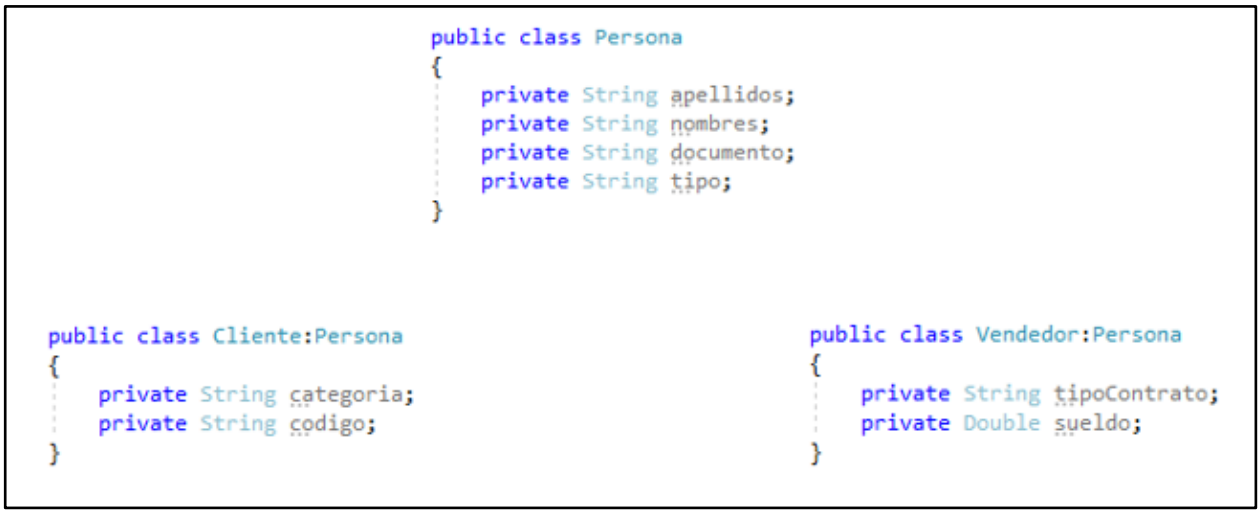

Figura 5. Herencia de clases en C#.
Fuente: Gandolfi, C. (2020) Módulo Taller de Programación, AIEP.

3.1.1. BENEFICIOS DE LA HERENCIA

Las subclases proveen conductas especializadas sobre la base de elementos comunes provistos por la superclase. A través del uso de herencia, los programadores pueden reutilizar el código de la superclase muchas veces.

- Los programadores pueden implementar superclases llamadas clases abstractas que definen conductas genéricas. Las superclases abstractas definen, y pueden implementar parcialmente, pero gran parte de la clase no esta definida ni implementada. Otros programadores concluirán esos detalles con subclases especializadas.

3.1.2. VENTAJAS DE LA METODOLOGÍA ORIENTADA A OBJETOS

Reutilización: Las clases están diseñadas para que se reutilicen en muchos sistemas, para maximizar la reutilización, las clases se construyen de manera que se pueden adaptar a los otros sistemas. Un objetivo fundamental de las técnicas orientadas a objetos es lograr la reutilización masiva al construir el software.
Estabilidad: Las clases diseñadas para una reutilización repetida se vuelven estables.

Calidad: Los diseños suelen tener más calidad, puesto que se integran a partir de componentes probados, que han sido verificados y pulidos varias veces.

Un diseño más rápido: Las aplicaciones se crean a partir de componentes ya existentes. Muchos de los componentes están construidos de modo que se pueden adaptar para un diseño particular.

Integridad: Las estructuras de datos (objetos) solo se pueden utilizar con métodos específicos. Esto tiene importancia en los sistemas Cliente/Servidor y los sistemas distribuidos, donde usuarios desconocidos podrían intentar el acceso al sistema.

Mantenimiento más sencillo: El programador encargado del mantenimiento tiene facilidad para el mantenimiento del código.

Independencia del diseño: Las clases están diseñadas para ser independientes del ambiente de plataformas, hardware y software. Utilizan solicitudes y respuestas con formato estándar, esto les permite ser utilizadas en múltiples sistemas operativos, controladores de bases de datos, controladores de red, interfaces de usuarios gráficas, etc. El creador del software no tiene que preocuparse por el ambiente o esperar a que este se especifique.

Interacción: El software de varios proveedores puede funcionar como conjunto, un proveedor utiliza clases de otros.

Computación Cliente-Servidor: En los sistemas Cliente/Servidor, las clases en el software cliente deben enviar solicitudes a las clases en el software servidor y recibir respuestas. Una clase servidor puede ser usada por clientes diferentes. Estos métodos solo pueden tener acceso a los datos del servidor a través de los métodos de la clase, los datos están protegidos contra su corrupción.

Computación de distribución masiva: Las redes a nivel mundial utilizaran directorios de software de objetos accesibles, el diseño orientado a objetos es clave para la computación distribuida masiva.

Mayor nivel de automatización de las bases de datos: Las estructuras de datos (objetos) en las bases de datos orientadas a objetos están ligadas a métodos que realizan acciones automáticas. Una base de datos OO tienen integrada una inteligencia, en forma de métodos, en tanto una base de datos relacional carece de ello.

Mejores herramientas CASE: Las herramientas Case utilizarán técnicas gráficas para diseñar las clases y sus interacciones, y para utilizar objetos existentes adaptados en nuevas aplicaciones. Las herramientas deberían facilitar el modelamiento en términos de eventos, triggers (iniciadores), estado de los objetos, etc. Las herramientas de los case orientados a objetos generan códigos tan pronto como una clase sea definida y permitirá al diseñador probar y utilizar el método creado.

3.1.3. PRINCIPIOS DE LA METODOLOGÍA ORIENTADA A OBJETOS

- La elección de qué modelos se creen influye directamente sobre como se enfoque el problema. Se debe seleccionar el modelo adecuado para cada momento y dependiendo del modelo escogido se obtendrán diferentes beneficios y distintos costes. Los softwares modelados con orientación a objetos proporcionan arquitecturas más flexibles y re adaptables.
- Todo modelo puede ser expresado a diferentes niveles de precisión. Es necesario poder seleccionar el nivel de detalle deseado, ya que en diferentes fases del proyecto se pueden tener necesidades distintas.
- Los mejores niveles están ligados a la realidad. Lo principal es tener modelos que permitan representar la realidad lo más claro posible, y estas deben adaptarse a la realidad para no ocultar detalles importantes.
- Un único modelo no es suficiente. Cualquier sistema que no sea trivial se afronta mejor desde pequeños modelos que se puedan construir de forma independiente. (Romero, 2015, p.11)

3.2. DISEÑO ORIENTADO A OBJETOS

El diseño Orientado a Objetos (DOO) difiere considerablemente del diseño estructurado ya que en DOO no se realiza un problema en términos de tareas (subrutinas) ni en términos de datos, sino que se analiza el problema como un sistema de objetos que interactúan entre sí.

Un problema desarrollado con técnicas orientadas a objetos requiere, en primer lugar, saber cuáles son los objetos del programa. Como tales objetos son instancias de clases, la primera etapa en el desarrollo orientado a objetos requiere de la identificación de dichas clases (atributos y comportamiento), así como las relaciones entre éstas y su posterior implementación en un lenguaje de programación.

Aunque no siempre están bien delimitadas las etapas de análisis y diseño en la OO (Orientación a objetos), se pueden sintetizar de alguna forma las ideas claves de las distintas tecnologías existentes dentro del desarrollo orientado a objetos al que denominaremos diseño.

En el diseño orientado a objetos, los objetos necesitan interactuar y comunicarse, para realizar dicha comunicación, los objetos utilizan su propia interfaz pública, dicha interfaz se compone principalmente de métodos y propiedades.

El diseño orientado a objetos transforma el modelo del análisis en un modelo de diseño que sirve como anteproyecto para la construcción de software. (ICTA, 2017)

3.2.1. CARACTERÍSTICAS PRINCIPALES DEL DISEÑO ORIENTADO A OBJETOS

Los objetos son abstracciones del mundo real o entidades del sistema que se administran entre ellas mismas.

- Los objetos son independientes y encapsulan el estado y la representación de información.
- La funcionalidad del sistema se expresa en términos de servicios de los objetos.
- Las áreas de datos compartidas son eliminadas. Los objetos se comunican mediante paso de parámetros.
- Los objetos pueden estar distribuidos y pueden ejecutarse en forma secuencial o en paralelo

3.2.2. VENTAJAS DEL DISEÑO ORIENTADO A OBJETOS

- Fácil de mantener, los objetos representan entidades autocontenidas.
- Los objetos son componentes reutilizables.
- Para algunos sistemas, puede haber un mapeo obvio entre las entidades del mundo real y los objetos del sistema.

3.2.3. MODELO DEL DISEÑO ORIENTADO A OBJETOS

El modelo del diseño puede verse en dos dimensiones distintas, como se ilustra en la figura. La dimensión del proceso indica la evolución del modelo del diseño conforme se ejecutan las tareas de éste como parte del proceso del software. La dimensión de la abstracción representa el nivel de detalle a medida que cada elemento del modelo de análisis se transforma en un equivalente de diseño y luego se mejora en forma iterativa. (ICTA, 2017)

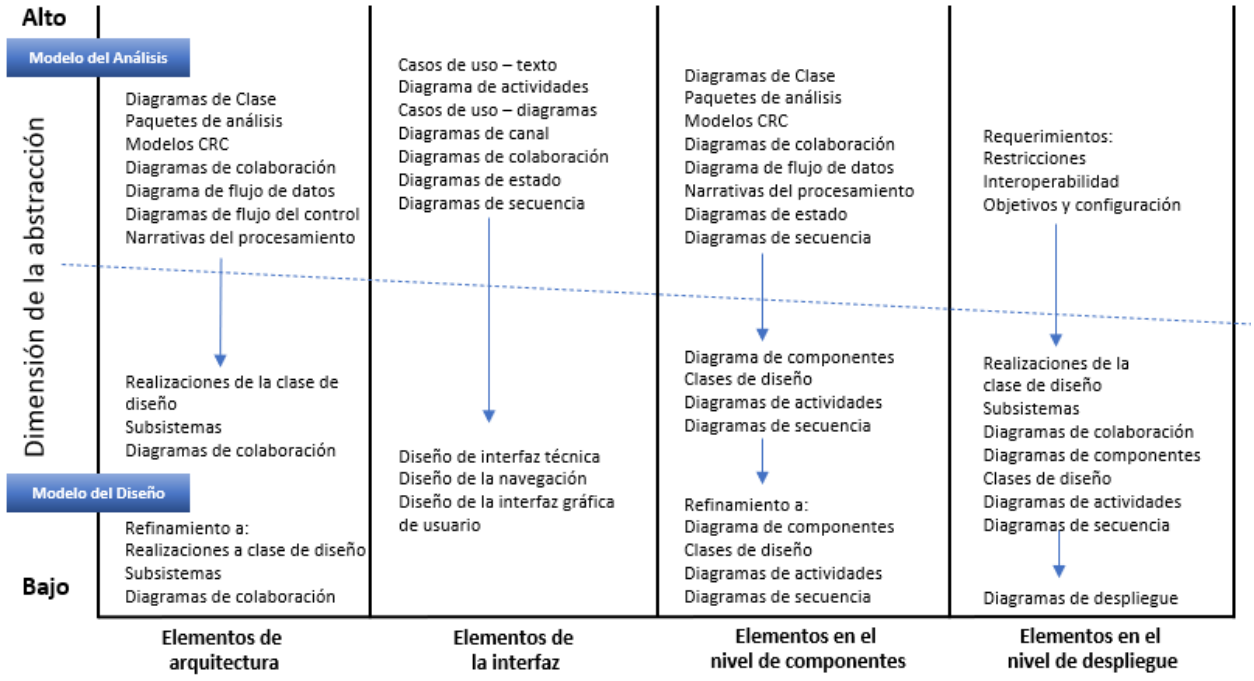

Figura 6. Herencia de clases en C#.
Fuente: ICTA, 2017.

En la figura, la línea punteada indica la frontera entre los modelos de análisis y de diseño. En ciertos casos, es posible hacer una distinción clara entre ambos modelos. En otros, el modelo de análisis se mezcla poco a poco con el de diseño y la distinción es menos obvia.

3.3. MODELO DEL DISEÑO ORIENTADO A OBJETOS

El diseño de software es un proceso iterativo por medio del cual se traducen los requerimientos en un plano para construir el software. Al principio, el plano ilustra una visión holística del software. Es decir, el diseño se representa en un nivel alto de abstracción, en el que se rastrea directamente el objetivo específico del sistema y los requerimientos más detallados de datos, funcionamiento y comportamiento. A medida que tienen lugar las iteraciones del diseño, las mejoras posteriores conducen a niveles menores de abstracción. (ICTA, 2017)

3.3.1. LINEAMIENTOS Y ATRIBUTOS DE LA CALIDAD DEL SOFTWARE

El diseño es importante porque permite que un equipo de software evalúe la calidad de éste antes de que se implemente, momento en el que es fácil y barato corregir errores, omisiones o inconsistencias.
Durante el diseño, la calidad se evalúa por medio de la realización de una serie de revisiones técnicas. Una revisión técnica es una reunión celebrada por miembros del equipo de software. Por lo general, participan dos, tres o cuatro personas, en función del alcance de la información del diseño que se revisará.

3.3.2. LINEAMIENTOS PARA EL DISEÑO

- Debe tener una arquitectura que se haya creado con el empleo de estilos o patrones arquitectónicos reconocibles, esté compuesta de componentes con buenas características de diseño y se implementen en forma evolutiva de modo que faciliten la implementación y las pruebas.
- Debe ser modular, es decir, el software debe estar dividido de manera lógica en elementos o subsistemas.
- Debe contener distintas representaciones de datos, arquitectura, interfaces y componentes.
- Debe conducir a estructuras de datos apropiadas para las clases que se van a implementar y que surjan de patrones reconocibles de datos.
- Debe llevar a componentes que tengan características funcionales independientes.
- Debe conducir a interfaces que reduzcan la complejidad de las conexiones entre los componentes y el ambiente externo.
- Debe obtenerse con el empleo de un método repetible motivado por la información obtenida durante el análisis de los requerimientos del software.
- Debe representarse con una notación que comunique con eficacia su significado.

3.4. PATRONES DEL DISEÑO ORIENTADO A OBJETOS

Son soluciones simples y elegantes a problemas específicos y comunes del diseño orientado a objetos. Son soluciones basadas en la experiencia y que se ha demostrado que funcionan, un patrón es una descripción del problema y la esencia de su solución, que se puede reutilizar en casos distintos. (ICTA, 2017)
Está claro que un equipo de desarrollo no se limita a aplicar los principios del diseño orientado a objetos una vez tras otra, lo más seguro es que se aprovechen de las soluciones aplicadas en el pasado para resolver los problemas del presente. La solución a algunos problemas del desarrollo viene en forma de bloques de código y de esqueletos de una solución, a estos bloques de código se les conoce como patrones.

Un patrón se puede definir de la siguiente manera:
"Un patrón describe un problema que ocurre de forma iterativa en nuestro entorno, también describe la solución a dicho problema de tal forma que podemos usar la misma solución todas las veces que sea necesario". (ICTA, 2017)

Los patrones de diseño son un conjunto de prácticas de óptimo diseño que se utilizan para abordar problemas recurrentes en la programación orientada a objetos.

3.4.1. OBJETIVOS DE LOS PATRONES

- Proporcionar catálogos de elementos reusables en el diseño de sistemas software.
- Evitar la reiteración en la búsqueda de soluciones a problemas ya conocidos y solucionados anteriormente.
- Formalizar un vocabulario común entre diseñadores.
- Estandarizar el modo en que se realiza el diseño.
- Facilitar el aprendizaje de las nuevas generaciones de diseñadores condensando conocimiento ya existente.

Los patrones no imponen alternativas de diseño frente a otras, tampoco eliminar la creatividad inherente al proceso de diseño.

3.5. COMPONENTES DEL DISEÑO ORIENTADO A OBJETOS

Para facilitar la comprensión del diseño y la forma de representar los componentes de la orientación a objetos, nos centraremos en la revisión de un caso básico utilizando los siguientes elementos:

3.5.1. CLASE

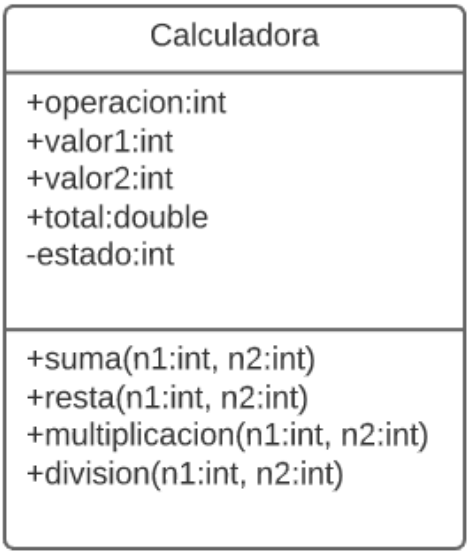

Figura 7. Clase Calculadora.
Fuente: Gandolfi, C. (2020) Módulo Taller de Programación, AIEP.

La clase Calculadora, está compuesta de cinco atributos y cuatro métodos, podemos observar que los símbolos + o – indican el ámbito de los atributos y métodos, + es público y – es privado.

3.5.1.1. DICCIONARIO DE ATRIBUTOS CLASE CALCULADORA

Atributo	Descripción
+operación	Atributo público de tipo entero que indicara la operación seleccionada
+valor1	Atributo público de tipo entero que contendrá un primer valor solicitado
+valor2	Atributo público de tipo entero que contendrá un segundo valor solicitado
+total	Atributo público de tipo entero que contendrá el resultado de la operación solicitada
-estado	Atributo privado de tipo entero que contendrá del objeto instanciado

3.5.1.2. DICCIONARIO DE MÉTODOS CLASE CALCULADORA

Atributo	Descripción
+suma(n1:int, n2:int)	Método público que recibe dos parámetros y devuelve el valor de una suma
+resta(n1:int, n2:int)	Método público que recibe dos parámetros y devuelve el valor de una resta
+multiplicación(n1:int, n2:int)	Método público que recibe dos parámetros y devuelve el valor de una multiplicación
+división(n1:int, n2:int)	Método público que recibe dos parámetros y devuelve el valor de una división

3.5.2. CASO DE USO

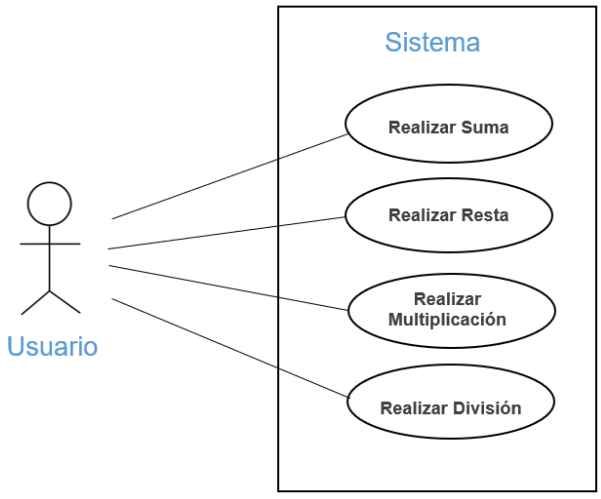

Figura 8. Caso de Uso.
Fuente: Gandolfi, C. (2020) Módulo Taller de Programación, AIEP.

Los casos de uso permiten describir las funcionalidades de un sistema, en este caso usamos las notaciones de usuario y usos para describir las acciones que se pueden realizar en el sistema, en este caso los métodos contenidos en la clase Calculadora, suma, resta, multiplicación y división.

3.6. CODIFICACIÓN EN C#

```
public class Calculadora
{
    //Atributos
    public int operacion;
    public int valor1;
    public int valor2;
    public double total;
    private int estado;

    //Metodos
    0 references
    public double suma(int n1, int n2)
    {
        total = n1 + n2;
        return Convert.ToInt32(total);
    }
    0 references
    public double resta(int n1, int n2)
    {
        total = n1 - n2;
        return Convert.ToInt32(total);
    }
    0 references
    public double multiplicacion(int n1, int n2)
    {
        total = n1 * n2;
        return Convert.ToInt32(total);
    }
    0 references
    public double division(int n1, int n2)
    {
        total = n1 / n2;
        return Convert.ToInt32(total);
    }
}
```

Figura 9. Codificación de clase en C#.
Fuente: Gandolfi, C. (2020) Módulo Taller de Programación, AIEP.

La clase Calculadora tiene ámbito público, tiene cinco atributos, cuatro públicos y uno privado. También posee cuatro métodos, todos de ellos públicos y que realizan las operaciones matemáticas básicas.

3.6.1. DESCRIPCIÓN DEL CÓDIGO C#

A continuación, se revisará cada línea de código de la clase Calculadora.

La siguiente imagen, contiene la descripción general de la clase calculadora, una clase de ámbito publico sin atributos ni métodos.

```
0 references
public class Calculadora
{

}
```

Figura 10. Estructura de la clase.
Fuente: Gandolfi, C. (2020) Módulo Taller de Programación, AIEP.

La siguiente imagen, contiene la descripción de los atributos de la clase Calculadora, son los mismos que se han descrito en el punto 2.5.1. correspondiente a la descripción de la clase. Podemos visualizar cuatro atributos públicos y uno privado, cuatro de tipo entero(tipo de dato que soporta valores numéricos enteros) y uno de tipo double (tipo de dato que soporta valores numéricos con decimales).

```
public class Calculadora
{
    //Atributos
    public int operacion;
    public int valor1;
    public int valor2;
    public double total;
    private int estado;
}
```

Figura 11. Atributos de clase Calculadora.
Fuente: Gandolfi, C. (2020) Módulo Taller de Programación, AIEP.

La siguiente imagen, contiene la descripción de los métodos de la clase calculadora, son los mismos que se han descrito en el punto 2.6. correspondiente a la descripción de casos de uso. Se puede visualizar cuatro métodos públicos.

```
public class Calculadora
{
    //Atributos
    public int operacion;
    public int valor1;
    public int valor2;
    public double total;
    private int estado;

    //Metodos
    0 references
    public double suma(int n1, int n2)
    {
        total = n1 + n2;
        return Convert.ToInt32(total);
    }
    0 references
    public double resta(int n1, int n2)
    {
        total = n1 - n2;
        return Convert.ToInt32(total);
    }
    0 references
    public double multiplicacion(int n1, int n2)
    {
        total = n1 * n2;
        return Convert.ToInt32(total);
    }
    0 references
    public double division(int n1, int n2)
    {
        total = n1 / n2;
        return Convert.ToInt32(total);
    }
}
```

Figura 12. Métodos de clase calculadora.
Fuente: Gandolfi, C. (2020) Módulo Taller de Programación, AIEP

Los métodos descritos en la clase Calculadora se definen de tal forma que retornaran un valor, todos contienen un tipo de datos antes de la descripción del método, para todos los métodos el tipo es double, reciben dos parámetros o argumentos de tipo entero (n1 y n2), dentro del código de instrucciones de cada método se le asigna a la variable total la operación correspondiente al método ejecutado, sea este suma, resta, multiplicación o división.

Con la palabra reservada *return*, cada método retorna el valor de la variable total de tipo double. Los métodos de la clase tienen relación con el diagrama de clases del punto 3.5.1 o figura 7 y también en las notaciones de casos de uso del punto 3.5.2 o figura 8

La siguiente imagen, contiene la definición de un formulario que solicitara a un usuario el ingreso de dos valores de tipo entero, y la operación matemática que se desea calcular:

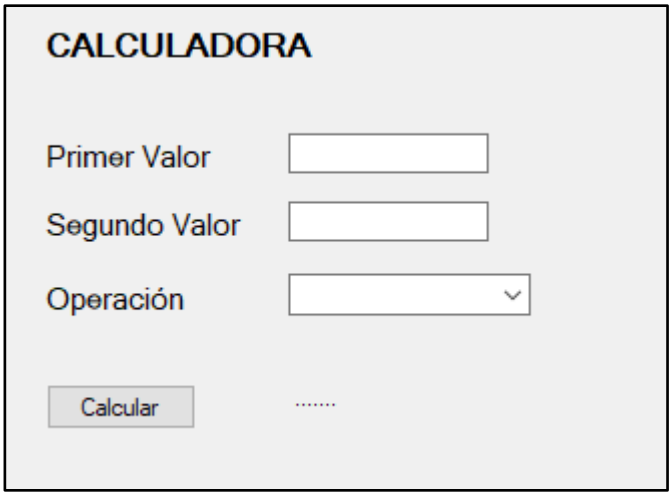

Figura 13. Formulario Calculadora.
Fuente: Gandolfi, C. (2020) Módulo Taller de Programación, AIEP

El formulario solicita al usuario dos valores de tipo entero y la posibilidad de seleccionar la operación deseada (suma, resta, multiplicación, división) para realizar el cálculo deseado.

La siguiente imagen, contiene una instancia (objeto) de la clase Calculadora, debemos recordar que una instancia es la representación de un objeto como tal, con propiedades y acciones disponibles para ser ejecutadas.

```
private void button1_Click(object sender, EventArgs e)
{
    //Creación de instancia
    Calculadora objeto1 = new Calculadora();
    //Ejecutar un metodo de acuerdo a la opción escogida
    if (opcion.Text == "1") //Suma
    {  men.Text = Convert.ToString(objeto1.suma(Convert.ToInt32(valor1.Text), Convert.ToInt32(valor2.Text))); }
    if (opcion.Text == "2") //Resta
    {  men.Text = Convert.ToString(objeto1.resta(Convert.ToInt32(valor1.Text), Convert.ToInt32(valor2.Text))); }
    if (opcion.Text == "3") //Multiplicación
    {  men.Text = Convert.ToString(objeto1.multiplicacion(Convert.ToInt32(valor1.Text), Convert.ToInt32(valor2.Text))); }
    if (opcion.Text == "4") //División
    {  men.Text = Convert.ToString(objeto1.division(Convert.ToInt32(valor1.Text), Convert.ToInt32(valor2.Text))); }
}
```

Figura 14. Crear instancia clase calculadora.
Fuente: Gandolfi, C. (2020) Módulo Taller de Programación, AIEP

En las siguientes líneas de código se realiza una instancia de la clase Calculadora:

```
//Creación de instancia
Calculadora objeto1 = new Calculadora();
```

Figura 15. Instancia clase calculadora.
Fuente: Gandolfi, C. (2020) Módulo Taller de Programación, AIEP

La sintaxis de instanciación es similar en todo lenguaje orientado a objetos, el nombre de la clase va seguido del nombre que le daremos al objeto, después un signo igual para generar la instancia con la palabra reservada new, seguido del nombre de la clase.

La siguiente imagen permite utilizar el objeto instanciado y ejecutar el método suma:

```
if (opcion.Text == "1") //Suma
{
    men.Text = Convert.ToString(objeto1.suma(Convert.ToInt32(valor1.Text), Convert.ToInt32(valor2.Text)));
}
```

Figura 16. Ejecución método suma.
Fuente: Gandolfi, C. (2020) Módulo Taller de Programación, AIEP

El objeto instanciado tiene como nombre objeto1, este a nivel de código seguido de un punto mostrara todos los métodos contenidos de la clase de ámbito público, y dentro de los paréntesis deben ir los parámetros con valores enteros esperados por el método.

Al elemento *Label* de nombre *men.Text* se le asigna el valor que devolverá el método utilizando las conversiones de tipos de datos correspondientes.

PALABRAS CLAVE

- Metodología
- Diseño
- Componentes
- Implementación
- Reutilizaición
- Integridad
- Patrones de diseño

CONCLUSIONES

Durante la tercera semana se ha profundizado en la programación orientada al objeto, revisando su metodología, las formas de diseño y sus principales componentes desde un diagrama de clases, casos de uso y la respectiva codificación en el lenguaje de programación C# utilizando el IDE Visual Studio 2019.

Se ha logrado conocer la importancia del Lenguaje UML (Unified Modeling Language) para el diseño de software orientados a objetos, documentando y especificando esquemas de los sistemas. Además se ha profundizado en los principios de herencia, polimorfismo, abstracción y encapsulamiento de la POO, conociendo sus ventajas para el desarrollo de software.

También hemos revisado los componentes de un pequeño desarrollo pasando por un diagrama de clases y sus respectivos casos de usos, terminando con una implementación en el lenguaje de programación orientado a objetos como C#.

REFERENCIAS

- *Academia EDU (2015), Metodología Orientada a Objetos – Introducción a UML [online]. Recuperado de <https://www.academia.edu/15943563/Que_es_la_Metodolog%C3%ADa_orientada_a_objetos_Introducci%C3%B3n_a_UML> [03 agosto 2020]*

- *ITCA (2017), Selección de Técnicas de Ingeniería de Software [online]. Recuperado de <https://virtual.itca.edu.sv/Mediadores/stis/2___diseo_orientado_a_objetos.html> [03 agosto 2020]*