1 Programación Estructurada vs Orientación a Objetos.

1.1 Sumario

2 Programación Estructurada vs Programación Orientada a Objetos

(En elaboración)

2.1 Ingeniería de Software

La Ingeniería de Software es una disciplina que se ocupa de la construcción de aplicaciones para ordenador de una forma robusta y fiable. Así como los ingenieros civiles utilizan métodos probados para la construcción de edificios, los ingenieros de software van a utilizar métodos estandarizados para analizar un problema a resolver, un esquema o un plan para el diseño de la solución y un método de construcción que minimice el riesgo de errores . La disciplina ha evolucionado gracias a la expansión del uso de computadores. En particular, se han abordado cuestiones que han surgido como resultado de algunos fracasos catastróficos de proyectos de software entre equipos de programadores que escriben miles y miles de líneas de código en programas. Así como los ingenieros civiles han aprendido de sus fracasos también lo han hecho los ingenieros de software.

Un método particular o una **familia de métodos** que un ingeniero de software podría utilizar para resolver un problema es lo que se conoce como una **metodología**. Durante la década de 1970 y en los años 80, la metodología de ingeniería de software principal fue la **programación estructurada**. El enfoque de la programación estructurada para el diseño de programas se basó en el método siguiente:

- Para resolver un problema grande, hay que dividir el problema en varios trozos más pequeños y trabajar en cada pieza por separado.
- Para resolver cada pieza, se tratará como un nuevo problema que a su vez puede dividirse en problemas más pequeños.
- Repetir el proceso con cada nueva pieza hasta que cada una se puede resolver directamente, sin descomposición adicional.

Este método también se conoce como **diseño top-down**.

Lo siguiente es un ejemplo sencillo del enfoque de la programación estructurada para la resolución de problemas.

Escriba un programa para una computadora que muestre el promedio de dos números introducidos mediante teclado. El promedio se va a visualizar en una pantalla que también está conectada a este equipo.

La solución top-down a la que se llega será la siguiente:

Nivel superior:

- 0. Mostrar promedio de dos números introducidos a través del teclado.

Siguiente nivel:

- 0.1. Obtener dos números a través del teclado.
- 0.2. Calcular la media de estos dos números.
- 0.3. Mostrar la media en pantalla.

Los 3 niveles que hemos obtenido se podrán codificar en cualquier lenguaje de programación.

El diseño Top-down es un enfoque útil y de uso frecuente para la resolución de problemas. Sin embargo, tiene limitaciones:

- Se centra casi exclusivamente en la producción de las instrucciones necesarias para resolver un problema. El diseño de las estructuras de datos es una actividad que es muy importante pero que está en gran medida fuera del ámbito de aplicación del diseño descendente.

- Es difícil reutilizar el trabajo realizado para otros proyectos. Al comenzar con un problema particular y subdividirlo en partes más pequeñas, de arriba hacia abajo el diseño del programa tiende a producir un **diseño que es único para ese problema**. La adaptación de una parte de la programación de otro proyecto por lo general implicaría una gran cantidad de esfuerzo y tiempo.
- Algunos problemas por su propia naturaleza no encajan en el modelo top-down. La solución no se puede expresar fácilmente en una secuencia particular de instrucciones. Cuando el orden en que las instrucciones no pueden determinarse de antemano, es necesario pensar con un enfoque diferente.

2.2 Diseño top-down

En el diseño de abajo hacia arriba, el enfoque es empezar "por abajo", con problemas que ya han sido resueltos y para los que se dispone de un componente de software reutilizable. Es importante citar que en esta aproximación los componentes serán lo más **modulares** posible.

Un módulo es un componente de un sistema más amplio, que interactúa con el resto del sistema de una manera simple y bien definida.

La idea es que un módulo pueda ser "conectado a" un sistema. Los detalles de lo que ocurre dentro de un módulo no son importantes para el sistema en su conjunto, sólo que el módulo cumpla su función correctamente.

Por ejemplo, un módulo puede contener procedimientos para imprimir una lista de estudiantes, para añadir un nuevo estudiante, editar los detalles de un estudiante y para obtener una lista de estudiantes específicos. Cómo el módulo almacena los registros de estudiantes se oculta el sistema de almacenamiento de las aplicaciones / sistemas que utilizan este módulo. De manera similar, el detalle de cómo los diversos procedimientos se codifican también se oculta. Ésto se conoce como **ocultamiento de información**. Es uno de los principios más importantes de la ingeniería de software. Las aplicaciones sólo necesitan conocer los procedimientos que están disponibles en el módulo y los datos a los que pueden ser accedidos. Esta información se publica. A menudo se denomina el **interfaz del módulo o interfaces**.

Un formato común para un módulo de software podría ser un módulo que contiene algunos datos, junto con algunas subrutinas (subprogramas / procedimientos / funciones) para manipular los datos. Los propios datos a menudo se ocultan de la vista, en el interior del módulo, forzando al programa a utilizar el módulo para manipular los datos indirectamente, llamando a las subrutinas proporcionadas por el módulo para este propósito. Las ventajas de este enfoque son las siguientes:

- Los datos están protegidos, ya que sólo pueden ser manipulados utilizando métodos bien definidos.
- Es más fácil para escribir programas utilizar un módulo porque los detalles de cómo se representan los datos y se almacenan, no es necesario conocerlos.
- La estructura de almacenamiento de los datos y el código para las subrutinas en un módulo pueden ser alteradas sin afectar a los programas que hacen uso del módulo, siempre y cuando las interfaces publicadas y la funcionalidad del módulo sigan siendo los mismas.

Los módulos que podrían apoyar este tipo de ocultación de información se hicieron muy comunes en los lenguajes de programación a principios de 1980. El concepto ha evolucionado desde entonces convirtiéndose en una plataforma central de la ingeniería de software llamada **programación orientada a objetos**, cuya abreviatura es **OOP**.

El concepto central de la programación orientada a objetos es el **objeto**, que es un tipo de módulo que contiene los datos y las subrutinas.

Un objeto es una especie de entidad auto-suficiente que tiene un **estado interno** (los datos que contiene) y que responde a los **mensajes** (llamadas a sus subrutinas). El objeto estudiante, por ejemplo, tiene un estado que comprende los datos de todos los estudiantes registrados. Si se le envía un mensaje diciéndole que añada los detalles de un nuevo estudiante, responderá modificando su estado para reflejar el cambio. Si se envía un mensaje diciéndole que imprima, responderá imprimiendo una lista con los datos de todos los estudiantes registrados.

El enfoque de la programación orientada a objetos comienza por:

- La identificación de los objetos implicados en un problema.
- Identificar los mensajes a los que estos objetos deben responder.

La solución resultante es una colección de objetos, cada uno con sus propios datos y su propio conjunto de responsabilidades. Los objetos interactúan entre sí mediante el envío de mensajes. No hay mucho diseño top-down en esta aproximación.

Al principio la gente encuentra difícil la programación OOP, sin embargo desde el momento en que se empieza a conocer dicha metodología , los programas tienden a modelar mucho mejor el mundo y los comportamientos reales que deseamos programar. Se suele decir que producen soluciones que son:

- Fáciles de escribir.

- Fáciles de comprender.
- Contienen menores errores.

2.3 Aproximación a Orientación a Objetos

¿Por qué la aproximación de orientación a objetos se considera una aproximación más realista a la forma en la que el mundo funciona?

Consideremos el siguiente ejemplo:

- Primera persona: Vicente, tienes hambre?
- Segunda persona: Si, tengo hambre Antonio.

Clasificando este intercambio en **orientación a objetos**: hay dos mensajes que se envían entre dos objetos.

Los objetos son Vicente y Antonio. Los mensajes son "tienes hambre?" y "sí, tengo hambre". Vicente sabe cómo responder al mensaje que recibe. Su conjunto de respuestas pueden ser (Sí, tengo hambre, No, no tengo hambre). Vicente elige la respuesta adecuada mediante la comprobación de su estado interno, es decir si su estómago está o no vacío.

Los objetos se crean sabiendo cómo responder a determinados mensajes. Diferentes objetos pueden responder al mismo mensaje de diferentes formas.

Si por ejemplo Vicente es un robot, la respuesta al mensaje de "Vicente, ¿tienes hambre?" podría ser "esa es una pregunta tonta, yo no tengo características humanas".

Esta propiedad de los objetos, que permite que objetos diferentes puedan responder al mismo mensaje de diferentes formas - se denomina **polimorfismo**.

Otro concepto importante en programación orientada a objetos es el concepto de **clase**.

Es bastante común tener objetos que pertenecen a la misma familia. Estos son los objetos que contienen el mismo tipo de datos y que responden a los mismos mensajes de la misma forma. Tales objetos se dice que pertenecen todos a la misma clase. En términos de programación, se crea una clase y luego uno o más objetos son creados usando esa clase como una plantilla.

Por ejemplo, la familia reloj consta de dispositivos que realizan el seguimiento del paso del tiempo. Cuando su estado interno cambia, actualizan una pantalla mostrando horas, minutos y segundos. El único mensaje que un miembro de la familia de reloj necesita saber es cómo resetear su hora interna a la hora y minutos facilitados. Hay muchos tipos de relojes actualmente. Todos ellos pertenecerían a la familia reloj o **clase reloj** en términos de programación.

La clasificación de todos los relojes en una gran familia se denomina la **clase reloj**. Podemos dividir la clase reloj en dos subclases reloj llamadas **relojes analógicos** y relojes digitales. Ambas están relacionadas, pero se diferencian en la forma en la que muestran la hora.

Ambas **subclases** han heredado el comportamiento básico de la clase reloj, pero cada una tiene su forma particular de mostrar la hora. Una subclase hereda el comportamiento y las características de una clase. La subclase puede añadir a su herencia e incluso puede "anular" una parte de la herencia mediante la definición de una respuesta diferente a algún mensaje que es conocido en la clase padre.

La **herencia** es una potente herramienta de programación. También es compatible con la reutilización de componentes de software. Una clase es el último componente reutilizable. Puede ser reutilizada directamente, si se ajusta exactamente al nuevo programa que estemos desarrollando. Si casi se ajusta, todavía se podría reutilizar, mediante la definición de una subclase y luego hacer los cambios necesarios en la subclase para que se ajuste exactamente.

Una buena ilustración de jerarquías de clases - la clase, subclase, etc. - es un sencillo programa de dibujo que permite a sus usuarios dibujar lineas, rectángulos, polígonos y curvas con un grosor de pincel en la pantalla. Cada objeto visible en la pantalla podría estar representado en software por un objeto en el programa. Habría cuatro clases de objetos en el programa, una para cada tipo de objeto visible que se puede extraer. Todas las líneas pertenecerían a una clase, todos los rectángulos a otra clase y así sucesivamente.

¿Cuál es la relación entre estas clases?

Todas estas clases representan "Objetos de Dibujo". Todas ellos sabrían cómo responder a un mensaje de "dibujar".

¿Cuál es la relación entre la clase línea y rectángulo?

Las líneas y los rectángulos necesitan solamente dos puntos para definirlos. Una línea necesita dos puntos y un rectángulo otros dos (esquina superior izquierda y esquina inferior derecha).

¿Cuál es la relación entre las clases polígono y curva?

Ambas clases representan Objetos de Múltiples Puntos.

Podemos ver que hay una jerarquía de clases que pertenecen al programa de dibujo simple. El Objeto de Dos Puntos y el Objeto de Múltiples Puntos, son subclases de la clase objeto Dibujo. Las clases Línea y Rectángulo son subclases de la clase Objeto de Dos puntos. Las clases Polígono y Curva son subclases de la subclase Objeto de Múltiples Puntos. Las relaciones entre clases se muestran en la figura siguiente:

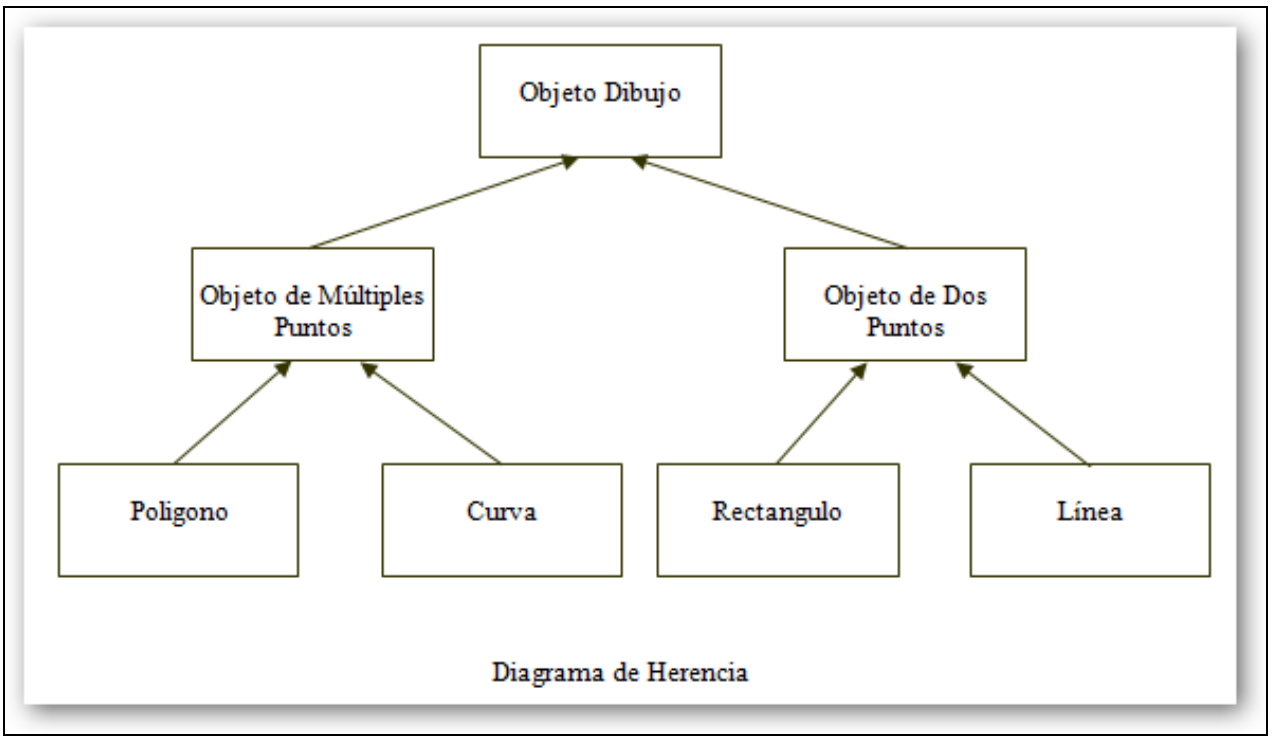

Diagrama de Herencia

2.4 Reusabilidad

Uno de las bases principales de la Programación Orientada a Objetos es la reutilización de código. En lugar de empezar desde cero con cada nueva aplicación, un programador consultará las bibliotecas de componentes existentes para ver si se pueden utilizar o son adecuados como puntos de partida para el diseño de una nueva aplicación.

Estos componentes existen en las bibliotecas como definiciones de clase. Un programador seleccionará una definición de clase apropiada de una biblioteca y luego creará una subclase de la aplicación. La subclase hereda los métodos y propiedades de la biblioteca de clases, y añadirá algunas nuevas funcionalidades o bien redefinirá algunas ya diseñadas anteriormente.

La **reusabilidad** es la capacidad de los elementos de software de poder utilizarse para la construcción de diferentes aplicaciones.

En lenguajes de programación visuales como Delphi, Visual C++ y Visual Basic las bibliotecas almacenan las definiciones de clase de los componentes que permiten que las interfaces gráficas de usuario (GUI) puedan construirse.

Se muestra a continuación un diagrama de herencia de clases.

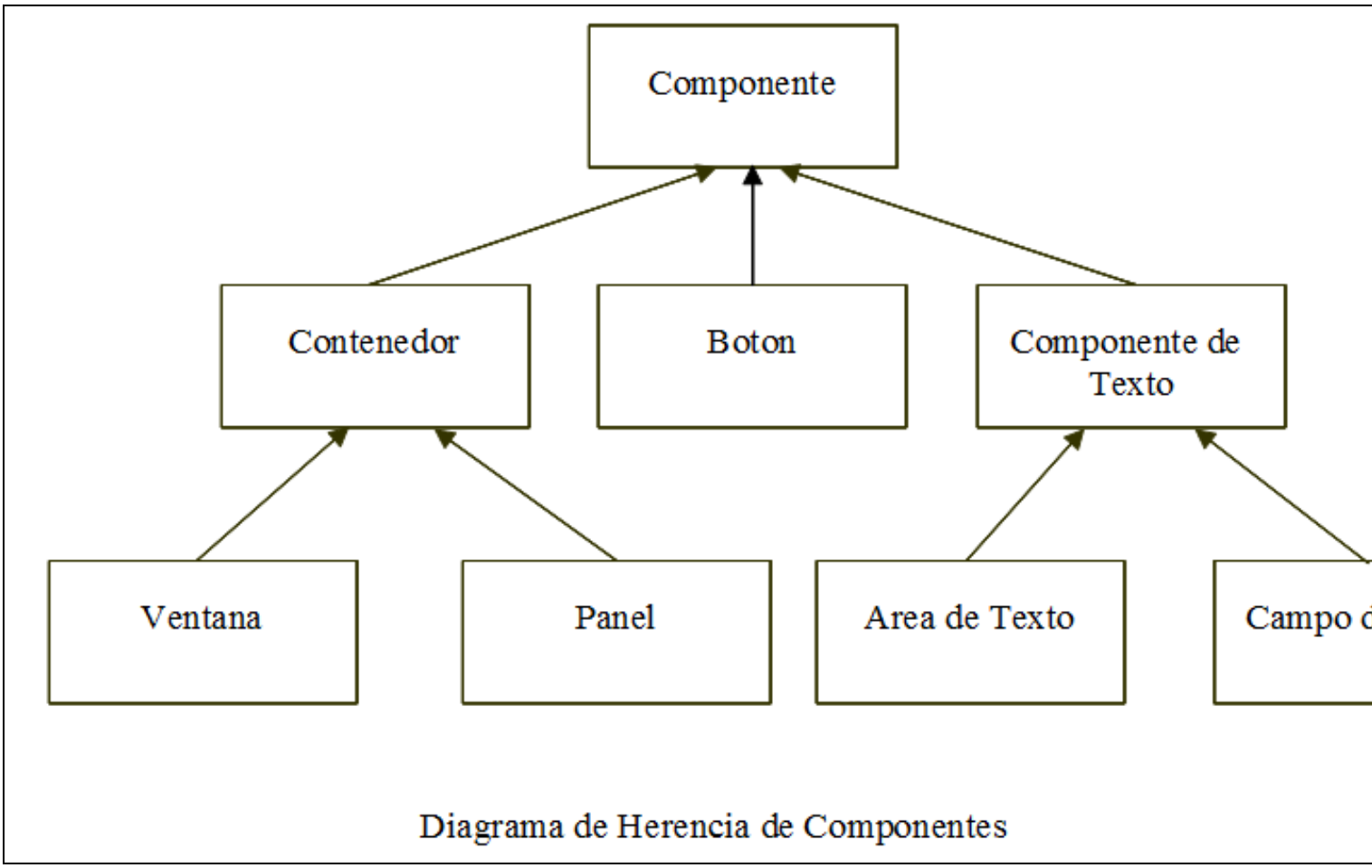

Diagrama de Herencia de Componentes

La popularidad de los sistemas orientados a objetos se basa en la cultura de los programadores en la que el software se desarrolla en base a componentes que serán reutilizados.

2.4.1 Beneficios de la reusabilidad

- **Confiabilidad**: los componentes creados por especialistas en su campo son más propensos a ser diseñados correctamente y con fiabilidad. La reutilización de estos componentes a través de muchas aplicaciones ha dado a los desarrolladores de los componentes la reacción suficiente para hacer frente a cualquier error y corregir sus posibles fallos.
- **Eficiencia**: los desarrolladores de componentes suelen ser expertos en su campo y han utilizado los mejores algoritmos y estructuras de datos posibles.
- **Ahorro de tiempo**: al confiar en los componentes existentes hay que desarrollar menos software y por lo tanto, las aplicaciones se pueden construir más rápido.
- **Disminución del esfuerzo de mantenimiento**: el uso de componentes desarrollados por otra persona disminuye la cantidad de esfuerzo de mantenimiento que el desarrollador de aplicaciones tiene que realizar. El mantenimiento de los componentes reutilizados es responsabilidad del proveedor de componentes.
- **La consistencia**: la dependencia de una biblioteca de componentes estándar tiende a difundir el mensaje de consistencia en nuestro diseño a través de del equipo de programadores que trabajan en una aplicación. La biblioteca es la base del estándar que dará coherencia y conformidad durante el proceso de diseño.
- **Inversión**: la reutilización de software le ahorrará el coste de desarrollar un software similar desde cero. La inversión en el desarrollo original se mantiene si el software desarrollado puede ser utilizado en otro proyecto. La mayoría del software reutilizable tiende a ser producido por los mejores desarrolladores. La reutilización de software es, pues, una forma de preservar el conocimiento y las creaciones de los mejores programadores.

2.5 Características principales de un lenguaje de programación orientado a objetos

- Encapsulación: al ocultamiento del estado, es decir, de los datos miembro, de un objeto de manera que sólo se puede cambiar mediante las operaciones definidas para ese objeto.

- Herencia: mecanismo de los lenguajes de programación orientada a objetos basados en clases, por medio del cual una clase se deriva de otra de manera que extiende su funcionalidad. La clase de la que se hereda se suele denominar clase base, clase padre, superclase, clase ancestro (el vocabulario que se utiliza suele depender en gran medida del lenguaje de programación).

- Polimorfismo: posibilidad de enviar un mensaje a un grupo de objetos cuya naturaleza puede ser heterogénea. El único requisito que deben cumplir los objetos que se utilizan de manera polimórfica es saber responder al mensaje que se les envía.

(Traducción de Dr.K.R. Bond 2000)

--Veiga (discusión) 09:31 3 feb 2014 (CET)