Análisis y Diseño Orientado a Objetos

Contenido

Fundamentos de POO

Modularidad

¿Qué es la modularidad? Es la propiedad que permite tener independencia entre las diferentes partes de un sistema. La modularidad consiste en dividir un programa en módulos o partes, que pueden ser compilados separadamente, pero que tienen conexiones con otros módulos. En un mismo módulo se suele colocar clases y objetos que guarden una estrecha relación.
La experiencia ha demostrado que la mejor manera de desarrollar y mantener un programa extenso es construirlo a partir de pequeñas piezas sencillas, o módulos. Cómo mencionamos anteriormente, a esta técnica se le llama “divide y vencerás”.

Principios de la Modularidad

1. **Capacidad de descomponer un sistema complejo.** Recuerda el principio de «Divide y Vencerás», en este procedimiento se realiza algo similar, ya que descompones un sistema en subprogramas (recuerda llamarlos módulos), el problema en general lo divides en problemas más pequeños.

2. **Capacidad de componer a través de sus módulos**. Indica la posibilidad de componer el programa desde los problemas más pequeños completando y resolviendo el problema en general, particularmente cuando se crea software se utilizan algunos módulos existentes para poder formar lo que nos solicitan, estos módulos que se integran a la aplicación deben de ser diseñados para ser reusables.

3. **Comprensión de sistema en partes.** El poder tener cada parte separada nos ayuda a la comprensión del código y del sistema, también a la modificación del mismo, recordemos que si el sistema necesita modificaciones y no hemos trabajado con módulos definitivamente eso será un caos.

Una de las razones por la cuál es útil hacerlo modular es debido a que podemos tener los límites bien definidos y es lo más valioso a la hora de leer el programa, recordemos una buena práctica en programación es hacerlo pensando en quien mantendrá el código.

Herencia

La herencia, es una forma de reutilización del software en la que se crea una nueva clase

absorbiendo los miembros de una clase existente. Además, se mejoran con nuevas capacidades, o con modificaciones en las capacidades ya existentes.

Con la herencia, los programadores vamos a ahorrar tiempo durante el desarrollo, al reutilizar software probado y depurado de alta calidad. Esto también aumenta la probabilidad de que un sistema se implemente con efectividad. Al crear una clase, en vez de declarar miembros completamente nuevos, podemos designar que la nueva clase herede los miembros de una clase existente. Esta clase existente como hemos mencionado antes se conoce como superclase (o clase base), y la nueva clase se conoce como subclase (o clase derivada). Cada subclase puede convertirse en la superclase de futuras subclases. Una subclase generalmente agrega sus propios campos y métodos.

Por lo tanto, una subclase es más específica que su superclase y representa a un grupo más especializado de objetos. Generalmente, la subclase exhibe los comportamientos de su superclase junto con comportamientos adicionales específicos de esta subclase. Es por ello que a la herencia se le conoce algunas veces como especialización.

Características:

- Es un mecanismo que sirve para reutilizar clases
- Se utiliza cuando existen clases que comparten muchas de sus características
- Se extiende la funcionalidad de clases más genéricas
- Se introducen los conceptos de superclase y subclase

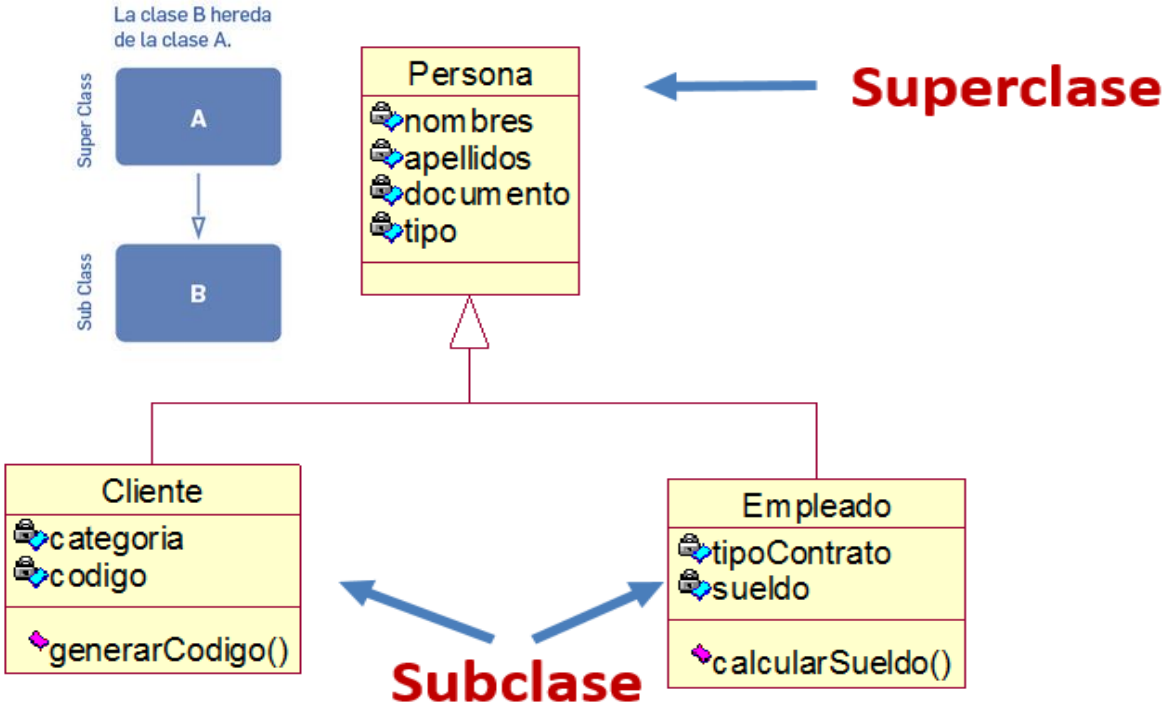
Jerarquía y Herencia

La jerarquía es una clasificación u ordenación de abstracciones. Las dos jerarquías más importantes en un sistema complejo son: su estructura de clases (la jerarquía "de clases") y su estructura de objetos (la jerarquía "de partes").

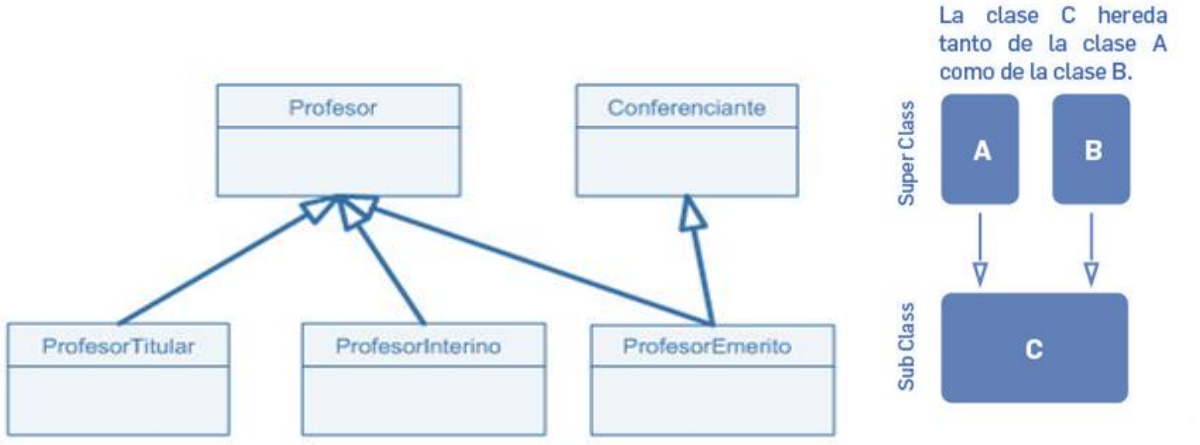
Las clases se encuentran relacionadas entre sí, formando una jerarquía de clasificación. La herencia es el medio para compartir en forma automática los métodos y los datos entre las clases y subclases de los objetos. Los objetos heredan las propiedades y el comportamiento de todas las clases a las que pertenecen. Existe la herencia Simple y Múltiple:

- **Herencia simple** (Una clase solo tiene una superclase. Un objeto pertenece a una sola clase)
- **Herencia múltiple** (Una clase tiene 2 o más superclases. Un objeto pertenece a más de una clase)

Herencia Simple



Herencia Múltiple



Entonces, la superclase directa es la superclase a partir de la cual la subclase hereda en forma explícita. Una superclase indirecta es cualquier clase arriba de la superclase directa en la jerarquía de clases, la cual define las relaciones de herencia entre las clases. En el caso de la herencia simple, una clase se deriva de una superclase directa.

Para la reutilización del código, se crean nuevas clases, pero utilizando las clases ya existentes y la estructura jerárquica de clases que se mencionó anteriormente. Para ello se utilizan los siguientes dos métodos más frecuentemente:

- **Composición**. Donde se crean objetos a partir de clases existentes, la nueva clase se compone de objetos de clases ya existentes.
- **Herencia**. Es como en la vida real, alguien hereda, adquiere algún bien o característica genética de su padre, por ejemplo, se puede decir: esta persona heredó hasta la forma de caminar de su padre, en POO esta clase es la clase padre, ya que conserva los atributos y métodos de la clase padre, pudiendo "alterar" (por medio de los modificadores de acceso), ya sea para quitar o agregar algunos de ellos en forma individual para esa clase en específico.

Polimorfismo

El polimorfismo nos permite "programar en forma general", en vez de "programar en forma específica". En especial, nos permite escribir programas que procesen objetos que compartan la misma superclase en una jerarquía de clases, como si todos fueran objetos de la superclase; esto puede simplificar la programación.

Definición de polimorfismo: El polimorfismo es un concepto de la Programación Orientada a Objetos (POO) que permite que un objeto pueda tomar múltiples formas. En Java, el polimorfismo se logra a través de la herencia y la implementación de interfaces.
Clases base y clases derivadas: El polimorfismo se basa en la relación de herencia entre una clase base y una o varias clases derivadas. La clase base define un conjunto común de métodos y propiedades, mientras que las clases derivadas heredan esos métodos y propiedades y pueden implementarlos de diferentes maneras.

Referencias y objetos polimórficos: En Java, se pueden usar referencias de la clase base para apuntar a objetos de la clase base o de cualquiera de las clases derivadas. Esto permite tratar a los objetos de diferentes clases derivadas de manera uniforme, ya que se pueden invocar los métodos comunes definidos en la clase base.

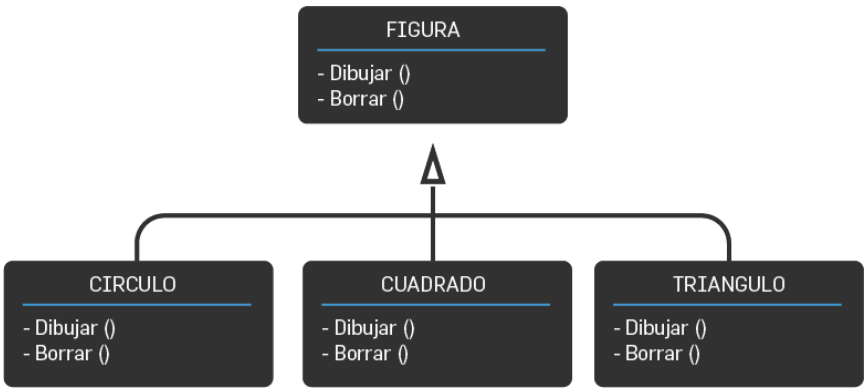
Métodos polimórficos: El polimorfismo se aplica principalmente a los métodos. Una clase derivada puede sobrescribir (***override***) un método de la clase base para proporcionar una implementación específica. Al utilizar una referencia de la clase base, el método invocado será el correspondiente a la clase derivada en tiempo de ejecución.

Ventajas del polimorfismo: El polimorfismo permite escribir código más flexible y extensible. Facilita la reutilización de código y promueve el diseño orientado a interfaces. Además, ayuda a lograr un código más legible y mantenible, al permitir trabajar con abstracciones en lugar de clases concretas.

Implementación de polimorfismo en Java: En Java, el polimorfismo se logra mediante

la herencia y la implementación de interfaces. Se debe utilizar la palabra clave ***extends*** para heredar de una clase base y ***implements*** para implementar una interfaz. Además, se debe utilizar la anotación ***@Override*** al sobrescribir un método de la clase base.

De manera resumida, el polimorfismo es una propiedad del EOO que permite que un método tenga múltiples implementaciones, que se seleccionan en base al tipo objeto indicado al solicitar la ejecución del método.



El polimorfismo operacional o sobrecarga operacional permite aplicar operaciones con igual nombre a diferentes clases o están relacionados en términos de inclusión. En este tipo de polimorfismo, los métodos son interpretados en el contexto del objeto particular, ya que los métodos con nombres comunes son implementados de diferente manera dependiendo de cada clase.

Clases diferentes (polimórficas) implementan métodos con el mismo nombre. Comportamientos diferentes, asociados a objetos distintos pueden compartir el mismo nombre; al llamarlos por ese nombre se utilizará el comportamiento correspondiente al objeto que se esté usando.

Esta característica facilita la implementación de varias formas de un mismo método, con lo cual se puede acceder a varios métodos distintos, que tienen el mismo nombre. Existen dos tipos principales de polimorfismo, que son:

1. Por reemplazo: dos o más clase diferentes con el mismo nombre del método, pero haciéndolo de forma diferente.

2. Por sobrecarga: es el mismo nombre de método ocupado varias veces, ejecutándolo de diferente forma y diferenciándose solamente por el argumento o parámetro.

- **Polimorfismo por herencia. Métodos virtuales**

La palabra clave virtual se usa para modificar una declaración de método, propiedad (no estática) o evento y permitir que, cualquier clase que herede este método pueda reemplazarlo.

- **Polimorfismo por abstracción: Clase Abstracta.**

Clase que no es posible crear instancias. Frecuentemente, están implementadas parcialmente o no están implementadas. Forzosamente se ha de derivar si se desea crear objetos de la misma.

- **Polimorfismo por interfaces: Interface.**

Es un CONTRATO. Define propiedades y métodos, pero no implementación.

¿Qué es Diseño de Software?

El diseño de software es una actividad previa a la programación que consiste en pensar como dividir el sistema o proceso en módulos o en grupos de líneas de código más pequeños, esto permite tener un modelo de los elementos de software que necesito antes de empezar a programar. Por último, poder planificar las tareas pudiendo asignar a distintos programadores cada módulo o fragmento de código.

Dividir el sistema en elementos de software: el objetivo es no hacer un programa como un solo bloque de código. El objetivo es separar en partes que se comunican o relacionan para poder unirlos como un rompecabezas, esto que nos permite poder desarrollar diferentes partes en paralelo. Estas partes de programas la llamaremos Elementos de Software.

Elementos de Software:

- **Módulos:** un módulo es una porción de un programa de ordenador. De las varias tareas que debe realizar un programa para cumplir con su función u objetivos, un módulo realizará, comúnmente, una de dichas tareas (o varias, en algún caso).
- **Programas:** Un programa es un conjunto de pasos lógicos escritos en un lenguaje de programación que nos permite realizar una tarea específica.
- **Procesos:** Un proceso es la ejecución de un programa, es decir, los datos e instrucciones están cargados en la memoria principal, ejecutándose o esperando a hacerlo.

- **Paquetes:** Cuando escribimos código, es posible que el resultado de nuestro programa sea bastante grande. Además, podríamos tener cientos o
- miles de clases distintas y relacionadas entre sí. Así que es normal agrupar estas clases dentro de paquetes, para hacer más fácil el mantenimiento. Los paquetes pueden contener clases, pero también puede contener otros paquetes, y así integrar una jerarquía entre ellos.
- **Componentes:** Un componente es una unidad modular de un programa con interfaces y dependencias bien definidas que permiten ofertar o solicitar un conjunto de servicios o funcionales.

Asignar una función específica a cada Elemento de Software consiste en describir con precisión la función que cumple cada elemento de software. Las relaciones entre los elementos del diseño son:

1. **Herencia:** es una forma de reutilización de software en la que se crea una nueva clase absorbiendo los miembros de una clase existente.

2. **Composición:** implica que tenemos una instancia de una clase que contiene instancias de otras clases que implementan las funciones deseadas. Es decir, estamos delegando las tareas que nos mandan a hacer a aquella pieza de código que sabe hacerlas. El código que ejecuta esa tarea concreta está sólo en esa pieza y todos delegan en ella para ejecutar dicha tarea.

3. **Inyección de dependencia:** La inyección de dependencias es un patrón de diseño de software usado en la Programación Orientada a Objetos, que trata de solucionar las necesidades de creación de los objetos de una manera práctica, útil, escalable y con una alta versatilidad del código.

4. **Interfaces:** una interfaz (también llamada protocolo) es un medio común para que los objetos no relacionados se comuniquen entre sí. Estas son definiciones de métodos y valores sobre los cuales los objetos están de acuerdo para cooperar.

Módulo

El concepto de módulo puede variar en diferentes documentaciones, pero pondremos varias definiciones que aplican para módulo:

1. Un módulo es una unidad de implementación de software que provee una funcionalidad definida o provee un conjunto de servicios definidos.
2. Un módulo es un conjunto de líneas de código agrupadas bajo un nombre, ese nombre puede ser reutilizado o referenciado en otros módulos de software.
3. Un módulo puede ser asignado a un programador para que diseño o implemente según las definiciones solicitadas.

La estructura de un módulo podemos pensar en dividir en 2 partes:

- **Interface:**

La interface es un conjunto mínimo de subrutinas que permite al módulo cumplir su función asignada, es decir cualquier tarea que el módulo deba hacer tiene que ser
a través de las subrutinas provistas por la interface. Es el único punto de interacción con el módulo, es decir, cualquier parte del sistema debe acceder al módulo únicamente por la interface.

La interface para ser definida debe asignarse un nombre, el tipo de datos de cada parámetro y el valor de retorno en caso que exista. La interface debe ser definida por el diseñador o arquitecto del sistema.

- **Implementación:**

La implementación la deben hacer los programadores.
El objetivo de dividir un sistema en módulos o partes más pequeñas es para tener un sistema “Diseñado para el Cambio”, el sistema estará sometido a cambios constantes por distintos factores al paso del tiempo, un Diseño en modulo permite agregar, modificar o reemplazar solamente el módulo sin afectar la funcionalidad del resto de los módulos o sistema. Los cambios dentro del sistema pueden ocurrir por muchos factores, pero listamos algunos:

1. Cambio a funcionalidades existentes ya programadas.
2. Cambiar la pantalla de usuario para agregarle un dato nuevo.
3. Cambiar una parte del cálculo que hace el sistema actualmente.
4. Cambios tecnológicos: (Necesidad de conectar con otros sistemas/ Necesidad de acceder desde internet / Necesidad de acceder desde celulares / Necesidad de conectar a una nueva base de datos).
5. Cambios socioculturales: (Cambio de moneda / Cambio la forma de calcular impuestos /Cambio de idiomas o internacionalizar).

Para profundizar más en tema podemos recomendarte los libros de David L. Parnas, Robert Martins

¿Qué es ADOO? Análisis y diseño orientado a objetos

Es un método de análisis que examina los requisitos desde la perspectiva de las clases objetos que se encuentran en el vocabulario del dominio del problema.

El Análisis orientado a objetos ofrece un enfoque nuevo para el análisis de requisitos de sistemas software. En lugar de considerar el software desde una perspectiva clásica de entrada/proceso/salida, como los métodos estructurados clásicos, se basa en modelar el sistema mediante los objetos que forman parte de él y las relaciones estáticas (herencia y composición) o dinámicas (uso) entre estos objetos.

El uso de Análisis orientado a objetos puede facilitar mucho la creación de prototipos, y las técnicas de desarrollo evolutivo de software. Los objetos son inherentemente reutilizables, y se puede crear un catálogo de objetos que podemos usar en sucesivas aplicaciones. De esta forma, podemos obtener rápidamente un prototipo del sistema, que pueda ser evaluado por el cliente, a partir de objetos analizados, diseñados e implementado en aplicaciones anteriores. Y lo que es más importante, dada la facilidad de reutilización de estos objetos, el prototipo puede ir evolucionando hacia convertirse en el sistema final, según vamos refinándolos objetos de acuerdo con un proceso de especificación incremental.

Marco de desarrollo de una aplicación software

El marco de desarrollo de una aplicación software estaría formado entoncespor tres fases: análisis, diseño e implementación.

1. El análisis es la fase cuyo objetivo es estudiar y comprender el dominio del problema, es decir, el análisis se centra en responder al interrogante ¿QUÉ HACER?
2. El diseño, por su parte, dirige sus esfuerzos en desarrollar la solución a los requisitos planteados en el análisis, esto es, el diseño se haya centrado enel espacio de la solución, intentando dar respuesta a la cuestión ¿CÓMO HACERLO?
3. Por último, la fase de implementación sería la encarga de la traducción del diseño de la aplicación al lenguaje de programación elegido, adaptando por tanto la solución a un entorno concreto.

Como se ha comentado en el apartado anterior, la transición entre las fases de análisis y diseño en la orientación al objeto es mucho más suave que en las metodologías estructuradas, no habiendo tanta diferencia entre las etapas. Esto implica que es difícil determinar dónde acaba el Análisis Orientado a Objeto (AOO)y donde comienza el Diseño Orientado a Objeto (DOO), siendo la frontera entre ambas fases totalmente inconsistente, de forma que lo que algunos autores incluyen en el AOO otros lo hacen en el DOO. Esto conduce a que sea frecuente eluso de las siglas ADOO para hacer referencia a ambas fases de forma conjunta.

El objetivo del AOO es modelar la semántica del problema en términos de objetos distintos pero relacionados. Por su parte, el DOO conlleva reexaminar las clases del dominio del

problema, refinándolas, extendiéndolas y reorganizándolas, para mejorar su reutilización y tomar ventaja de la herencia. El análisis se relaciona conel dominio del problema y el diseño con el dominio de la solución; por lo tanto, el AOO enfoca el problema en los objetos del dominio del problema y el DOO en los objetos del dominio de la solución.

Análisis Orientado a Objetos (AOO) y Diseño Orientado a Objetos (DOO)

Se puede definir AOO como el proceso que modela el dominio del problema identificando y especificando un conjunto de objetos semánticos que interaccionany se comportan de acuerdo con los requisitos del sistema. En el AOO deben llevarsea cabo las siguientes actividades:

- La identificación de clases semánticas, atributos y servicios
- Identificación de las relaciones entre clases (generalizaciones,agregaciones y asociaciones).
- El emplazamiento de las clases, atributos y servicios.
- La especificación del comportamiento dinámico mediante paso de mensajes

Se puede definir DOO como el proceso que modela el dominio de la solución, loque incluye a las clases semánticas con posibles añadidos, y las clases de interfaz, aplicación y utilidad identificadas durante el diseño. En el DOO deben llevarse a cabo las siguientes actividades:

- Añadir las clases interfaz, base y utilidad.
- Refinar las clases semánticas.

Como resumen final, se podría afirmar que el AOO y el DOO no deben verse comofases muy separadas, siendo recomendable llevarlas a cabo concurrentemente, así el modelo de análisis no puede completarse en ausencia de un modelo de diseño, ni viceversa. Uno de los aspectos más importantes del ADOO es la sinergia entre los dos conceptos.

Conceptos importantes de ADOO

En el caso de la POO, existen diferentes metodologías que consisten en construirun modelo (Representación de la realidad a través de diferentes variables) de un dominio de aplicación como:

OMT que es la Técnica del Modelado de Objetos, el cual a grandes rasgos cuentacon las siguientes cuatro fases: Análisis, Diseño del Sistema, Diseño de Objetos e Implementación.

UML (del que ya hablamos anteriormente), el cual es un Lenguaje Unificado deModelado y es una representación gráfica que sirve para modelar sistemas orientados a objetos, ya que permite manejar los elementos descritos en los apartados anteriores (por ejemplo, los

mensajes entre objetos, sincronización, etc.). Entre sus principales características se encuentran: su flexibilidad y que cuenta con muchos elementos gráficos.

Lenguajes orientados a objetos

Las técnicas, vistas anteriormente en las que se basa la programación orientada aobjetos (como el encapsulamiento, abstracción, etc.) ya eran conocidas, desde hace ya varios años, sin embargo, no todos los lenguajes proporcionan todas las facilidades para escribir programas orientados a objetos. Existen discrepancias de cuáles deben de ser estas facilidades y se pueden agrupar en las siguientes:

1. Manejo de memoria automático, incorporándose el concepto del recolector de basura, con lo que la memoria utilizada por objetos cuya utilidad ha terminado es liberada por mecanismos propios del lenguaje, sin intervención del programador.
2. Abstracción de datos a través del lenguaje.
3. Estructura modular en objetos, tomando como base sus estructuras de datos.
4. Clases, Herencia y polimorfismo que puedan manipuladas a través dellenguaje.

Tipos de datos

Los tipos de datos son los siguientes tres:

1. **Primitivos. -** (Son unidades de información tales como caracteres, números o valores boléanos (lógicos).
2. **Clases del sistema. -** No son clases y no tienen métodos propios.
3. **Clases definidas por el usuario. -** Como su nombre lo indica son hechaspor el usuario.

Los tipos primitivos son:

1. bolean.- Los cuales pueden tener valores de falso o verdadero (trae/false).
2. char.-Es un carácter de 16 bits.
3. byte, short.- Cuenta con 8, 16,32 y 64 bits
4. int, long.- Tiene valores entero y enteros largos.
5. Float, double.- Son números de punto flotante (notación científica) de 32 y64 bits.

Otra forma de entender la clasificación en tipos de datos es:

1. Tipos de datos primitivos: No son objetos) nos damos cuenta porque notienen Propiedades, métodos, y no necesitan instanciarse. además, quecomienzan con minúscula.

2. Tipos de datos que son Objetos: Los podemos reconocer porque podemos ver sus métodos y propiedades, estos objetos no necesitan ser INSTANCIADOS y se llaman objetos Estáticos. como por ejemplo los Integer y String, si bien es un tipo de datos similar a int notemos que empieza con mayúscula por ser una clase
3. También están los tipos de datos Objeto: Que SI NECESITAN SER INSTANCIADOS para poder ver sus métodos y propiedades.

¿Pero cómo vemos los métodos y propiedades de estos Objetos?Para poder ver los métodos y propiedades debemos:

- DECLARARLO (accedemos a sus métodos y propiedades)
- INSTANCIARLO (creando una nueva instancia o copia del objeto para poder usarlo.

Lenguaje Tipado y Tipificado

Los lenguajes tipificados (también llamados de tipado estático) son aquellos en los que una variable guarda siempre un mismo tipo de datos. En algunos lenguajes tipificados se exige al programador que declare el tipo de cada variable y en otros lo determina el compilador. Es bastante obvio que, si tienes que hacer un programa eficiente necesites un lenguaje de tipado, porque hace un buen uso de lamemoria RAM de tu computadora. ¿Pero, y con los lenguajes que no se usan para aplicaciones que necesiten esta eficiencia? Ahí es donde aparecen los lenguajes no tipados, donde no tienes que declarar explícitamente el tipo de dato que vas a usar, porque el lenguaje se hará cargo.

Entonces, la diferencia es que en uno tipado, tienes que manejar el tipo de dato detus variables, mientras que en uno no tipado no es que no se manejan los tipos dedatos, sino que dejas que el lenguaje de programación los maneje.

Cada uno tiene sus ventajas y desventajas, es obvio que al tener que manejar lostipos de datos, la programación es más compleja, pero es más eficiente, si no es tipado quizás tu aplicación no sea más rápida, pero será más fácil de programar.

Modelado de Objetos – UML

Los objetos identificados en la etapa de análisis se plasman en diferentes modelos, combatiendo de esta forma la complejidad inherente del problema al que nos estamos enfrentando. Para la construcción de modelos de objetos, y modelos software en general, se debe contar con un lenguaje de modelado, es decir, un lenguaje para especificar, visualizar, construir y documentar los componentes de los sistemas software.

Anteriormente dimos una pequeña introducción a UML. Este lenguaje de modeladoque se

había convertido en estándar de facto de las notaciones en orientación a objeto.

El hecho de la estandarización de la notación de UML nos conduce hacia una notación única en la que expresar las abstracciones identificadas en análisis, así como sus relaciones. Además, esos mismos modelos serán refinados durante la fase de análisis y diseño hasta que estén preparados para su implementación en el lenguaje de programación elegido.

A la hora de realizar un modelo de un sistema software, éste debe hacerse desde diferentes puntos de vista, de forma que recojan tanto la dimensión estática y estructural de los objetos como su componente dinámica. Un lenguaje de modelado debe aportar una serie de mecanismos gráficos que permitan captar laesencia de un sistema desde diversas perspectivas. Estos mecanismos gráficos suelen recibir el nombre de diagramas en la mayoría de los lenguajes de modelado.

UML (Lenguaje Unificado de Modelado)

Como ya hemos presentado antes, el Lenguaje Unificado de Modelado (UML) fue creado para forjar un lenguaje de modelado visual común y semántica y sintácticamente rico para la arquitectura, el diseño y la implementación de sistemas de software complejos, tanto en estructura como en comportamiento. Escomparable a los planos usados en otros campos y consiste en diferentes tipos dediagramas. En general, los diagramas UML describen los límites, la estructura y elcomportamiento del sistema y los objetos que contiene.

Hay muchos paradigmas o modelos para la resolución de problemas en la informática, que es el estudio de algoritmos y datos. Hay cuatro categorías de modelos para la resolución de problemas: lenguajes imperativos, funcionales, declarativos y orientados a objetos (OOP). Como ya vimos, en los lenguajes orientados a objetos, los algoritmos se expresan definiendo 'objetos' y haciendo que los objetos interactúen entre sí. Esos objetos son cosas que deben ser manipuladas y existen en el mundo real. Pueden ser edificios, artefactos sobre un escritorio o seres humanos. Los lenguajes orientados a objetos dominan el mundode la programación porque modelan los objetos del mundo real. UML es una combinación de varias notaciones orientadas a objetos:

1. Diseño orientado a objetos
2. Técnica de modelado de objetos
3. Ingeniería de software orientada a objetos

UML usa las fortalezas de estos tres enfoques para presentar una metodología más uniforme que sea más sencilla de usar. UML representa buenas prácticas para la construcción y documentación de diferentes aspectos del modelado de sistemas de software y de negocios.

Características de UML

1. Establecer una definición formal de un metamodelo común basado en el estándar MOF (Meta-Object Facility) que especifique la sintaxis abstracta del UML. La sintaxis abstracta define el conjunto de conceptos de modeladoUML, sus atributos y sus relaciones, así como las reglas de combinación deestos conceptos para construir modelos UML parciales o completos.

2. Brindar una explicación detallada de la semántica de cada concepto demodelado UML. La semántica define, de manera independiente a la tecnología, cómo los conceptos UML se habrán de desarrollar por las computadoras.

3. Especificar los elementos de notación de lectura humana para representarlos conceptos individuales de modelado UML, así como las reglas para combinarlos en una variedad de diferentes tipos de diagramas que corresponden a diferentes aspectos de los sistemas modelados.

4. Definir formas que permitan hacer que las herramientas UML cumplan conesta especificación. Esto se apoya (en una especificación independiente) con una especificación basada en XML.

Para más información **Sitio oficial UML: https://www.uml.org/**

Conceptos de Modelado Específico

El desarrollo de sistemas se centra en tres modelos generales de sistemasdiferentes:

1. **Funcionales:** Se trata de diagramas de casos de uso que describen lafuncionalidad del sistema desde el punto de vista del usuario.
2. **De objetos:** Se trata de diagramas de clases que describen la estructura del sistema en términos de objetos, atributos, asociaciones y operaciones.
3. **Dinámicos:** Los diagramas de interacción, los diagramas de máquina deestados y los diagramas de actividades se usan para describir el comportamiento interno del sistema.

Conceptos orientados a objetos UML

Los objetos en UML son entidades del mundo real que existen a nuestro alrededor.En el desarrollo de software, los objetos se pueden usar para describir, o modelar, el sistema que se está creando en términos que sean pertinentes para el dominio. Los objetos también permiten la descomposición de sistemas complejos en componentes comprensibles que permiten que se construya una pieza a la vez.

Estos son algunos conceptos fundamentales de un mundo orientado a objetos:

1. **Objetos:** Representan una entidad y el componente básico.
2. **Clase:** Plano/molde de un objeto.
3. **Abstracción:** Comportamiento de una entidad del mundo real.
4. **Encapsulación:** Mecanismo para enlazar los datos y ocultarlos del mundoexterior.
5. **Herencia:** Mecanismo para crear nuevas clases a partir de una existente.
6. **Polimorfismo:** Define el mecanismo para salidas en diferentes formas.