**Presentación**

00009 Esta disciplina trasciende la actividad de programación, que es el pilar fundamental a la hora de crear una aplicación. El ingeniero de sistemas, de Datos y Software, y afines o tecnólogo, se encarga de toda la gestión del proyecto para que éste se pueda desarrollar en un plazo determinado y con el presupuesto previsto. La ingeniería de software, por lo tanto, incluye el análisis previo de la situación, el diseño del proyecto, el desarrollo del software, las pruebas necesarias para confirmar su correcto funcionamiento y la implementación del sistema.

## Red de conceptos

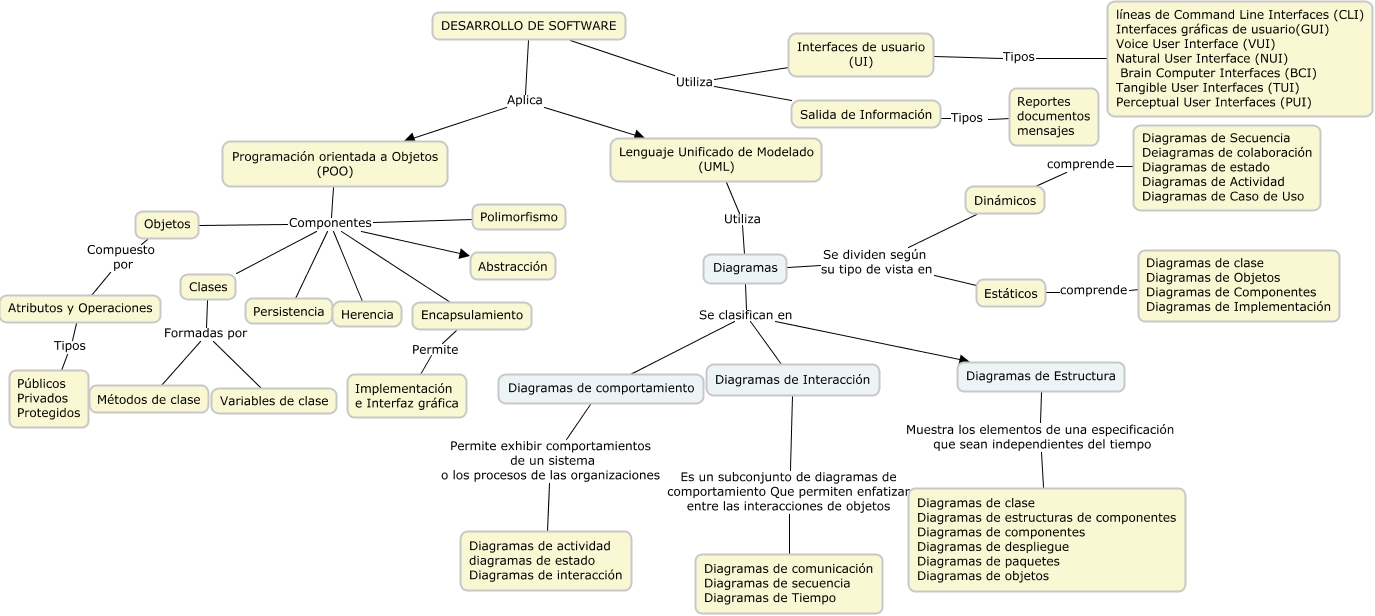


Figura 1. Metodologías y herramientas desarrollo de software

## Introducción

Ser programador no solo implica saber un lenguaje de programación, Hay mucho más en la programación profesional, principalmente se requiere habilidades para analizar los problemas o necesidades que se le plantean y dar una solución adecuada por medio de un programa informático, seguido a ello es fundamental dominar las herramientas y metodologías que las empresas utilizan en la actualidad para analizar , diagramar y documentar los procesos y que permite a los analistas de sistemas y analistas desarrolladores la implementación de una solución informática acorde a una necesidad específica.

En esta unidad el estudiante conocerá las etapas y las metodologías que los programadores deben aplicar para desarrollar soluciones a través de la implementación de programas informáticos, adicionalmente conoceremos los principios fundamentales sobre los cuales se basa la filosofía de la programación orientada a objetos.

# **Desarrollo temático**

# 1. Conceptos de diseño

## 1.1 fundamentos de la programación Orientada a objetos

La programación Orientada a objetos (POO, u OOP según sus siglas en inglés) es una técnica especial de programación, la podemos considerar como la mejor forma de representar y manejar las cosas de la vida real bajo un lenguaje de programación, lo cual sería difícil o complejo de manejar bajo otros modelos de programación.

La (POO) es un paradigma de programación que usa objetos en sus interacciones, para diseñar e implementar programas computacionales. Los paradigmas son estilos para el desarrollo de programas, es decir, un modelo para resolver problemas computacionales. Los lenguajes de programación siempre están enmarcados en uno o varios paradigmas.

La POO nos exige aprender a programar y a pensar las cosas de una forma distinta, muy diferente a la programación estructurada porque ya debemos desarrollar nuestros programas en términos de objetos, propiedades, métodos y otras cosas que veremos en el transcurso de esta lección.

Hoy en día la programación orientada a objetos es una de las formas más difundidas de programación y a nivel empresarial tiene una gran acogida en el desarrollo de proyectos de software. Esta acogida se debe a sus grandes capacidades y ventajas frente a las antiguas formas de programar. Este modelo de programación ofrece una mayor productividad del programador, mejor calidad de software y menor costo de mantenimiento.

## 1.2 Los objetos

Pensar en función de objetos es muy parecido a cómo percibimos el mundo real. Entender lo que es un objeto en programación es la clave para entender cualquier lenguaje orientado a objetos. Existe una gran diversidad de escritos sobre las definiciones de lo que es un objeto. Pero iniciemos entendiendo que **es un objeto del mundo real**.

Un objeto del mundo real es cualquier cosa que vemos a nuestro alrededor. Digamos que para movilizarnos lo hacemos a través del automóvil o una bicicleta, ambos son objetos, al igual que nuestro computador, el teléfono celular, un animal o una silla.

Es muy importante tener en cuenta que todos los elementos del mundo real tienen dos componentes que los identifican, el primero de ellos comprende **las** **características del objeto** y el segundo **el comportamiento** de este, por ejemplo un automóvil posee:

|  |  |
| --- | --- |
| CARACTERÍSTICAS | COMPORTAMIENTOS |
| ● Marca  ● Modelo  ● Color  ● Velocidad alcanzada  ● costo | ● Acelerar  ● Frenar  ● Retroceder  ● Girar  ● Cambiar llantas |

Al igual que en el mundo real, los objetos en los programas tienen unas características y unos comportamientos, Un objeto de software almacena sus características en una o más "**variables**", y sus comportamientos son definidos a través de lo que denominamos "**métodos**". *Un método básicamente es una función o subrutina asociada a un objeto.*



Es de aclarar que los métodos y los atributos se pueden compartir o se pueden restringir para ser accedidos desde afuera, esto dependiendo de la estructura que planteamos en su construcción. Por lo general los atributos siempre se ocultan al exterior y algunos métodos quedan expuestos al exterior para convertirse en la interfaz del objeto. A este ocultamiento de sus métodos y atributos se le denomina “**Encapsulamiento**”.

**Estados de un Objeto:** los objetos poseen estados y estos se dan cuando se le ponen valores a las características del objeto, las variables definidas como características de un objeto son las encargadas de almacenar los estados del objeto en un determinado momento.

Los objetos en la programación no son elementos “eternos”, es decir es necesario que estos sean creados, técnicamente a esta creación se le llama **Instanciar** y también se destruyen cuando han cumplido su misión.. Una vez que se se ha creado(instanciado) un objeto puede **recibir mensajes**. Los objetos pueden instanciarse de forma dinámica o estática. Un objeto estático comienza su existencia una vez es declarado (instanciación de objetos estáticos), sin embargo, un objeto dinámico no comienza su existencia al ser declarado, sino al recibir un espacio de memoria (instanciación dinámica de objetos).

Como la primera tarea que nos enfrentamos al desarrollar un programa bajo el paradigma de Programación orientada a objetos es la identificación de los objetos que hacen parte del problema que deseamos solucionar, para esto debemos tener en cuenta que los objetos los podemos identificar de acuerdo a las siguientes categorías:

● **Cosas Tangibles:** carro, casa, producto, juguete,computador.

● **Roles** : cliente, empleado, Jefe, vendedor, auxiliar,

● **Organizaciones o entidades:** Universidad, colegio, empresa, proveedor ,hospital.

● **Cosas intangibles:** Vuelos, Materias, cursos, Servicios.

## 1.3 Definición de clase

Las clases son un pilar fundamental de la programación orientada a objetos. Las clases son las plantillas para hacer objetos. Una clase sirve para definir una serie de objetos con propiedades (atributos), comportamientos (operaciones o métodos), y semántica comunes. Hay que pensar en una clase como un molde. A través de las clases se obtienen los objetos en sí, Es decir antes de poder utilizar un objeto se debe definir la clase a la que pertenece, esa definición incluye:

**SUS ATRIBUTOS**. Es decir, los datos miembros de esa clase. Los datos pueden ser accesibles desde otra clase (públicos) o sólo se puede acceder desde el código de su propia clase(privados). También se las llama campos.

**MÉTODOS:** Las funciones miembro de la clase. Son las operaciones o las acciones que puede realizar la clase.

**CÓDIGO DE INICIALIZACIÓN:** Son las operaciones previas necesarias para crear una clase a esto se le conoce como el constructor de la clase.

**OTRAS CLASES** Dentro de una clase se pueden definir otras clases.

|  |
| --- |
| **Cuando programamos un objeto y definimos sus características y funcionalidades en realidad lo que estamos haciendo es programar una clase** |

La siguiente imagen es la representación gráfica de una clase que se llama carro, a esta representación se le denomina diagrama de clase y ayuda al programador a visualizar cuales son las propiedades y métodos que contendrá una clase o conjunto de clases en particular.



Figura 2. representación gráfica de una clase

Los objetos son hijos de una clase cualquiera. Cuando creamos un objeto tenemos que especificar la clase a partir de la cual se creará. Recuerde que la acción de crear un objeto a partir de una clase se llama **instanciar**. Para crear un objeto se tiene que digitar una instrucción especial que puede ser distinta dependiendo del lenguaje de programación que se emplee, pero será algo parecido a la siguiente instrucción:

miautomovil = new automovil()

Con la palabra new especificamos que se tiene que crear una instancia de la clase que se nombra a continuación. Dentro de los paréntesis podríamos colocar parámetros con los que inicializar el objeto de la clase automóvil.

En el ejemplo anterior al objeto “miautomovil” podemos leer o asignar valores a sus propiedades. El valor concreto de una propiedad de un objeto se llama estado, para acceder a un estado de un objeto y ver su valor o para cambiarlo se utiliza el nombre del objeto y el operador punto.

Miautomovil.color=Azul

El objeto es Miautomovil, posteriormente le colocamos el operador punto y por último el nombre de la propiedad a la que deseamos conocer. En el ejemplo estamos modificando el valor del estado de la propiedad del objeto al valor de ”Azul” con una simple asignación.

Un mensaje en un objeto es la acción de efectuar una llamada a un método particular. Por ejemplo, cuando le indicamos a un objeto Miautomovil que se ponga en movimiento estamos pasándole el mensaje “”Muevete”, Para mandar mensajes a los objetos utilizamos el operador punto, seguido del método que deseamos invocar.

Miautomovil.muevete()

En este ejemplo pasamos el mensaje muevete(). Hay que colocar paréntesis igual que cualquier llamada a una función, dentro irían los parámetros que este requiere para su funcionamiento.

|  |
| --- |
| **Podemos concluir que: una instancia es un objeto de una clase en particular,** **los objetos y las instancias son ambas representación de una clase.** |

Si ya logramos entender lo son las clases y objetos, atributos y estados, métodos y mensajes, ahora podemos ampliar la información sobre otros conceptos de gran importancia en la POO.

## 1.4 La herencia

La Herencia es uno de los cuatro conceptos que son claves de la programación orientada a objetos (POO) junto con la **Abstracción**, **Encapsulación** y **Polimorfismo**. La herencia básicamente consiste en que una clase puede heredar sus variables y métodos a varias subclases. La clase que hereda es llamada superclase o clase padre, esto significa que una subclase, aparte de los atributos y métodos propios, tiene incorporados los atributos y métodos heredados de la superclase. De esta manera se crea una jerarquía de herencia.

**La Jerarquía** es un proceso por el cual se crean estructuras de elementos en distintos niveles. No es un concepto exclusivo de la POO, sino que es algo que vemos en la vida real en muchos ámbitos, algo propio a cualquier tipo de sistema. Puedo tener diversos tipos de jerarquías, como clasificación o composición.

● **Composición**: Es cuando unos elementos están compuestos de otros, o que unos elementos están presentes en otros, podemos tomar como ejemplo el sistema respiratorio, en este tenemos los pulmones los cuales están dentro del sistema respiratorio, dentro de los pulmones tenemos los alvéolos y bronquios. En esta jerarquía de elementos tenemos composición debido a que unos elementos forman parte de otros.

● **Clasificación**: Este tipo de jerarquía indica que unos elementos son una especialización de otros. Por ejemplo, los medios de transporte, en donde tenemos terrestre, aéreo y acuáticos luego dentro de los terrestres tenemos los trenes, autos, motos, bicicletas etc en los aéreos tenemos aviones, helicópteros, cohetes y en los acuáticos los barcos, submarinos etc.

La herencia es la transmisión del código desde unas clases a otras. Para soportar un mecanismo de herencia como vimos se deben tener dos clases: la clase padre y las clases hijas o una sola hija. La clase padre es la que transmite su código a las clases hijas. De igual forma que los humanos heredamos el código genético de nuestros padres, En diversos lenguajes de programación entre ellos Java se declara la herencia con la palabra "extends".

class Hija extends Padre{ }

Esto nos indica que todo el código de la clase padre se transmite, tal cual, a la clase hija. Es como tener escrito línea a línea el código de la class " Padre" dentro de la class " Hija". Esto nos evita tener que digitar de nuevo todo el contenido del código del padre dentro de la clases hijas, Es por Esto que la herencia es fundamental para reutilizar código, porque no necesitas volver a incorporar el código de una clase superior a una inferior, sino que al utilizar el "extends" se inserta automáticamente el código en las hijas.

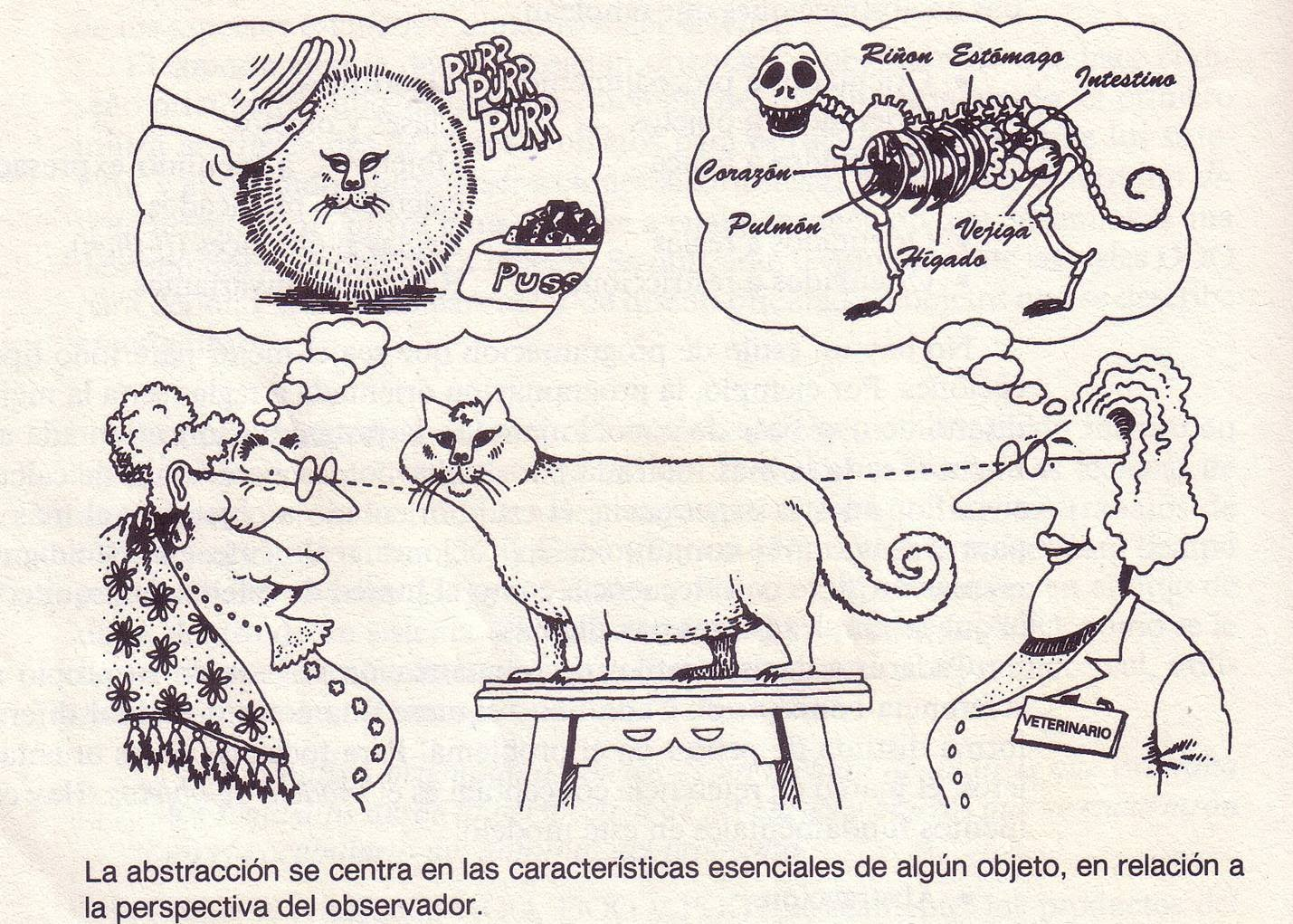
La herencia implica una relación de **especialización** en la que la subclase especializa el comportamiento o la estructura más general de sus superclases

## 1.4 Abstracción

**Es un proceso mental por el que se ignoran las características de algo, quedándonos con lo que realmente nos importa**. La clave para entenderlo es "proceso mental", así que nos tenemos que poner a pensar, extrayendo aquello que realmente nos importa e ignorando lo superfluo. (Fernández Muñoz & Alvarez, 2015)

Debido a la gran cantidad de información que manejamos los seres humanos nuestro cerebro está constantemente realizando una abstracción del mundo que nos rodea, sacando lo que verdaderamente nos interesa, sin esto, nuestro cerebro simplemente colapsaría. Por ejemplo, al tomar el metro único que nos interesa es la ruta, el costo del tiquete y el tiempo en desplazamiento, pero hay cosas que omitimos las cuales no nos interesa cómo quien conduce, cual es el costo del vagón, que potencia tiene, años que lleva funcionando etc.

Pero debemos aclarar que la abstracción es un proceso subjetivo. O sea que depende del punto de vista del observador, por ejemplo el mecánico del metro realizará otro tipo de abstracción seleccionando los detalles que por su profesión le interesan.



fuente: <https://sites.google.com/site/contenidospoo/abstracion.jpg>

Figura 3. representación de abstracción

|  |
| --- |
| **La abstracción consiste en captar las características esenciales de un objeto, así como su comportamiento.** |

## 1.5 Encapsulación

También conocida como **ocultamiento**, este es el proceso por el cual se oculta el funcionamiento y la estructura interna de los objetos, es como tener una caja negra en la cual no conozco su parte internan solo puedo acceder a lo que se me autoriza ver. Cuando encapsulamos estamos ocultando cómo se soporta algo, como se almacena, qué medio es, cuál es su nombre, etc

El encapsulamiento se logra gracias a la abstracción. La abstracción y la encapsulación no son lo mismo, pero si están relacionadas porque sin encapsulación no hay abstracción. La utilidad del encapsulamiento se da por la facilidad para manejar la complejidad, ya que tendremos las Clases como cajas negras donde sólo se conoce el comportamiento pero no los detalles internos, y esto es importante porque nos interesará conocer que hace la Clase pero no será necesario saber cómo lo hace.

Un ejemplo de encapsulación lo encontramos con nuestro televisor, el usuario conoce su apariencia y parte de su funcionamiento. Sólo le importa que funcionen el control remoto, el video y que se escuche; no le interesa saber cómo funciona cada una de las partes internas del aparato, ese detalle sólo le interesan al fabricante y al técnico de servicio.

La encapsulación da lugar a que las clases se dividan en dos partes:

**Interfaz**: son funciones definidas específicamente para modificar los atributos y evitar así que el usuario los modifique directamente, con estas interfaces se garantiza la integridad de los datos

**Implementación**: Comprende la representación de la abstracción, así como los mecanismos que conducen al comportamiento deseado.

La encapsulación permite definir los niveles de acceso para elementos de esa clase. Estos niveles de acceso definen los derechos de acceso para los datos, permitiéndonos el acceso a datos a través de un método de esa clase en particular, desde una clase heredada o incluso desde cualquier otra clase. Los siguientes son los niveles de acceso que se pueden definir en una clase

**Público**: funciones de toda clase pueden acceder a los datos o métodos de una clase que se define con el nivel de acceso público. Este es el nivel de protección de datos más bajo.

**Protegido**: el acceso a los datos está restringido a las funciones de clases heredadas, es decir, las funciones miembro de esa clase y todas las subclases.

**Privado**: el acceso a los datos está restringido a los métodos de esa clase en particular. Este es nivel más alto de protección de datos.

## 1.6 Polimorfismo

### En términos generales y de acuerdo a la definición de la Real academia española(RAE) el polimorfismo se refiere a “aquello que cuenta o que puede adoptar múltiples formas. El término también hace mención a una propiedad capaz de atravesar numerosos estados”. Esta es una palabra de origen griego que tiene como significado "muchas formas".

### El polimorfismo es uno de los principales conceptos de la programación orientada a objetos junto con la Abstracción, Encapsulación y Herencia. Para comprender qué es el polimorfismo es muy importante tener bien claro el concepto de la Herencia que vimos anteriormente.

### Debemos destacar en esta parte que cuando hablamos de herencia estamos haciendo relación a las clases y a su jerarquía, pero cuando hacemos referencia a polimorfismo, lo relacionamos con los métodos.

### En el contexto de la POO, el polimorfismo se refiere al hecho de que un “método” de una clase padre posea definiciones diferentes en clases hijas que las hereden. En otras palabras, es la capacidad que tienen los objetos de una clase de responder al mismo mensaje o evento de acuerdo a los parámetros utilizados durante su llamado. Un objeto polimórfico es una entidad que puede contener valores de diferentes tipos durante la ejecución del programa.

### En algunos lenguajes de programación el polimorfismo es también conocido como ‘Sobrecarga de parámetros’ ya que las características de los objetos permiten aceptar distintos parámetros para un mismo método. En el siguiente ejemplo veremos el método suma, el cual será invocado con diversos parámetros y el retornará unos resultados diversos de acuerdo a estos valores suministrados.

|  |
| --- |
| En este primer ejemplo el método suma() retorna la suma de los 2 valores ingresos como parámetrosFunction suma(int numero1,int numero2)En este segundo ejemplo utilizamos el mismo método pero con 3 párametros, el resultado será la suma de estos tres valoresFunction suma(int numero1,int numero2, int numero3)En el tercer ejemplo vemos que si pasamos valores de tipo carácter , estos no se suman pero si se unirán en una sola cadena lo que se denomina “concatenar”Function suma(String cadena1, String cadena2) |

### Como se aprecia el comportamiento del método depende de los parámetros de entrada.

### Un ejemplo más gráfico lo podemos apreciar mediante la siguiente imagen:

### 

fuente: <https://ortizol.blogspot.com.co/2013/12/polimorfismo-en-c.html>

Figura 4. representación gráfica herencia y de polimorfismo

### En la imagen se aprecia que las "Clases Hijas" Circulo, Triangulo y rectángulo heredan los métodos públicos o protegidos de su clase padre llamada “Figura” , En este caso el polimorfismo lo podemos aplicar invocando cualquiera de los tres métodos definidos en la clase “Figura”:

### · Dibujar()

### · EstablecerColor()

### · ObtenerColor()

### Como vemos el nombre de los métodos es el mismo en cada una de las clases hijas pero su funcionalidad interna no es la misma, por ejemplo se dibuja en forma diferente un triángulo a un circulo y a un rectángulo, es decir cada una de estas debe tener implementada la forma particular de dibujar. Al crear las diferentes instancias de los tipos de objetos Circulo, Triangulo, y Rectangulo y que cada uno de estas instancias invoque los métodos mencionados anteriormente, sólo se ejecutará la versión de la implementación no heredada sino la sobrescrita en cada uno de los tipos.

|  |
| --- |
| **Con el polimorfismo podemos obtener comportamientos diferentes, asociados a distintos objetos, y en donde los métodos comparten el mismo nombre, al llamarlos por ese nombre se utilizará el comportamiento correspondiente al objeto que se esté usando** |

### Como hemos podido ver el polimorfismo es un concepto mucho más avanzado que la herencia y puede ser muy útil en el momento de jerarquizar y querer dar un patrón de comportamiento común a un conjunto de objetos que heredan de la misma clase. En esta pequeña explicación no hemos visto todo lo referente al polimorfismo ya que nos tocaría ver conceptos mucho más avanzados los cuales se irán adquiriendo y practicando en el transcurso del aprendizaje de un lenguaje de programación como lo es Java.

### Es muy probable que en este momento no le encuentres sentido a este tema de polimorfismo, y esto es normal. Solo queda decir que a través de la experiencia que adquieras en el tiempo le encontrarás sentido a este importante concepto.

### 

## 1.7 La persistencia

### La persistencia es la propiedad de un objeto a través del cual su existencia trasciende en el tiempo y/o espacio. Esto significa que un objeto persistente sigue existiendo después de que ha finalizado el programa que le dio origen y que además puede ser movido de la localidad de memoria en la que fue creado.

para describirlo de otra forma podemos decir que la persistencia es la acción de mantener la información del objeto de una forma permanente (almacenada), la cual en el tiempo también se debe permitir su recuperación para nuevamente utilizarla.

Para la persistencia los objetos se pueden clasificar en dos tipos:

**Transitorios**: Son los objetos que en su tiempo de vida depende del espacio del proceso que lo creo.

**Persistentes**: Son aquellos que su estado es almacenado en un medio temporal para su posterior reconstrucción y utilización, por lo cuál el objeto no depende del proceso que lo creo.

### Para profundizar en el conocimiento de la Persistencia en POO te invitamos a leer el siguiente artículo: <http://webcem01.cem.itesm.mx:8005/apps/s201411/tc1018/notas_persistencia/>

### ¡Para Saber más!

Para continuar con el estudio de los conceptos planteados sobre la programación orientada a objetos, te recomendamos las siguientes lecturas:

* Fernández Muñoz, L., & Alvarez, M. (7 de Septiembre de 2015). Teoría de programación orientada a objetos. Recuperado el 18 de Junio de 2017, de Desarrollo Web:<https://desarrolloweb.com/manuales/teoria-programacion-orientada-objetos.html>
* Grupo de Investigación en Ingeniería de Software del Tecnológico de Antioquia(Octubre de 2009).Lógica y programación orientada a objetos: Un enfoque basado en problemas, Recuperado el 18 de Junio de 2017, de<https://issuu.com/carlosarturo.castrocastro/docs/libro_logica_y_programacion_de_objetos>

En los siguientes videos encontrarás ampliación de estos temas:

● FUNDAMENTOS PROGRAMACIÓN ORIENTADA OBJETOS, por Silvia Angelica Quintero, Recuperado el 15 de junio de 2017, de <https://www.youtube.com/watch?v=wuWWruFM_WQ>

● Introducción al Paradigma de la Programación Orientada a Objetos (JavaPOO,Recuperado el 15 de junio de 2017,<https://www.youtube.com/watch?v=wmECY8XCe4E>

# 2. El lenguaje de Modelado Unificado (UML)

El lenguaje unificado de modelado (UML, por sus siglas en inglés, Unified Modeling Language) es un conjunto de diagramas con su correspondiente descripción que permite especificar, construir, visualizar y documentar los elementos (artefactos) de un aplicativo de software construido con el paradigma orientado a objetos.

Uno de los diagramas UML utilizados en la fase de análisis del proceso de desarrollo de software, son los casos de usos trabajados en la unidad 2. En la la etapa de diseño el UML tiene entre otros usos, la especificación de estereotipos básicos de clases, diagrama de clases y diagrama de clases estereotipados.

Según la Real Academia de la lengua española, un estereotipo es "Imagen o idea aceptada comúnmente por un grupo o sociedad con carácter inmutable". Muchos estereotipos son infundados, por ejemplo que todos los hombres son valientes y no lloran.

En UML un estereotipo es una cadena de texto encerrada entre los símbolos de comillas francesas (« »), y se usa para especializar el significado o la funcionalidad de un elemento o artefacto del sistema definiendo de esta manera su categoría.

### 

## 2.1 Diagramas de clases

Un diagrama de clases, es un diagrama de estructura estática que describe un sistema mostrando las clases del sistema, sus atributos, operaciones (o métodos), y las relaciones entre las clases. Es un diagrama de estructura estática en el sentido de que estos diagramas muestran la estructura del sistema, pero no la secuencia de mensajes que se deben intercambiar entre los objetos creados con las clases (es decir las instancias).

Por ejemplo, un diagrama de clases para un sistema que genera y registra las facturas en un proceso de ventas es el siguiente:

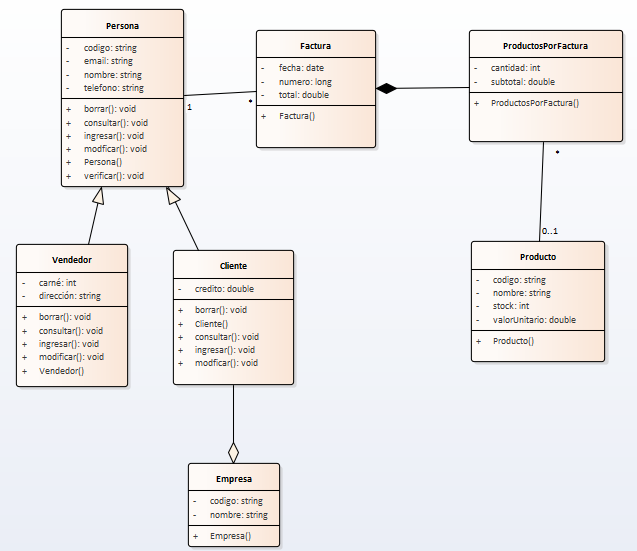


figura 5. Diagrama de Clases proceso de ventas

En este diagrama se presentan las clases, con sus correspondientes atributos y métodos que representa una estructura de un aplicativo de software para un proceso de ventas.

Entre la clase Cliente y la clase Persona se presenta una relación de herencia, es decir que las clases Cliente y Vendedor pueden usar los atributos y los métodos de la clase Persona. Por mecanismo de herencia, las clases Cliente y Vendedor, pueden usar los atributos código, email, nombre y teléfono de la Clase Persona. Igualmente, por el mecanismos de herencia, las clases Cliente y Vendedor pueden usar directamente el método verificar. Los métodos ingresar, consultar, modificar y borrar, se usan a través de polimorfismo.

La relación entre la clase Persona y la clase Factura, es una relación de asociación. En este caso una relación de uno a muchos (1…\*).

De igual manera, la relación entre la clase Producto y la clase ProductosPorFactura, es una relación de asociación uno a muchos.

La relación entre la clase Factura y la clase ProductosPorFactura, es una relación de composición, es decir, que un objeto de la clase Factura, está compuesto por uno o varios objetos de la clase ProductosPorFactura. Al borrar un objeto de la clase Factura, se eliminan los objetos de la clase ProductosPorFactura contenidos en la clase Factura.

La relación entre la clase Empresa y la clase Cliente, es una relación de agregación, es decir si se eliminan los objetos de la Clase Empresa, los objetos de la clase Cliente, se mantienen.

## 2.2 Diagramas de clases estereotipadas

Un estereotipo es una subclase con los mismos atributos y relaciones que la clase padre pero con la intención de especializar los objetos de la clase hija con restricciones adicionales. El estereotipo se representa con “ <<estereotipo>>”. En ocasiones no se usa directamente las comillas, sino una pequeña imagen para indicar el estereotipo.

Existen tres tipos de clases estereotipadas: clases borde o interfaz, clases entidad y clases control.

En el caso del proceso de facturas se seleccionó un modelo de tres capas para la solución. Es decir tres grupos o paquetes de clases especializadas en: la interfaz gráfica de usuario o capa vista, el modelo de datos, o capa modelo, y el controlador, es decir, clases que se encargan de velar por las reglas del negocio.

En caso del proceso de facturas, se elaboran tres grupos de clases estereotipadas. Un primer grupo de clases o paquete especializado en la interfaz gráfica. En este caso para representar el estereotipo, se seleccionó una imagen.

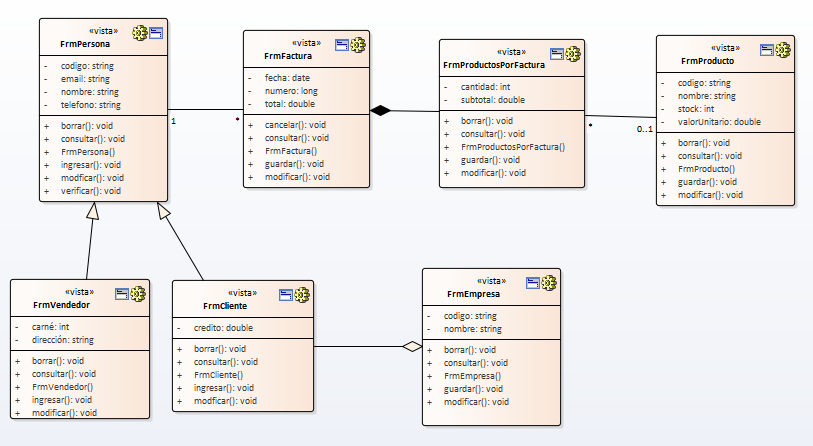
.

figura 6. Diagrama de Clases estereotipadas.capa vistas

### 

Para la capa controlador, se utilizó un diagrama de clases estereotipado como el que se presenta a continuación:

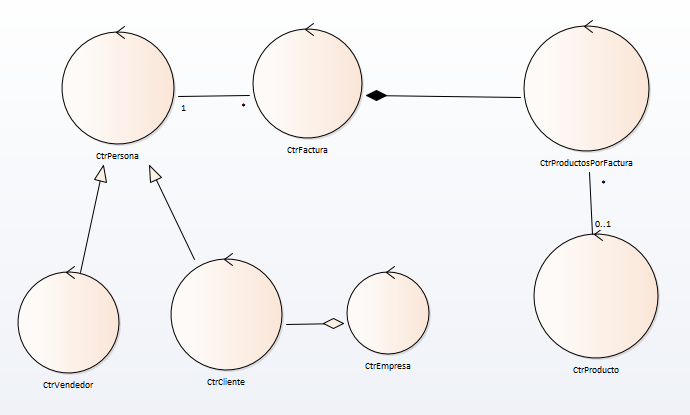


figura 6. Diagrama de Clases Estereotipadas. capa control

La capa modelo, es una representación de base de datos en memoria. A continuación se muestra el diagrama de clases estereotipado para la capa modelo:

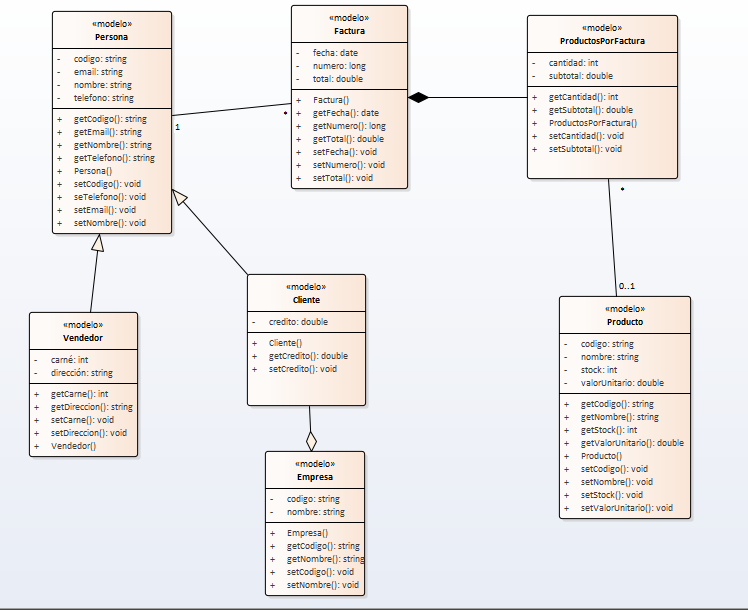


figura 7. Diagrama de Clases Estereotipadas. capa Modelo

### ¡Para Saber más!

Para continuar con el estudio de los conceptos planteados sobre Diagramas de Clase, te recomendamos las siguientes lecturas:

* <https://users.dcc.uchile.cl/~psalinas/uml/modelo.html>
* <http://eii.ucv.cl/pers/guidi/cursos/estructuras/pdf/SE-DiagramasDeClasesUML.pdf>
* <https://www.ecured.cu/Diagrama_de_Clase>
* <http://sevillajarquin.udem.edu.ni/wp-content/uploads/2014/01/Diagramas-de-Clases-y-Objetos.pdf>
* <http://www.hernanolivieri.com/Material/UML/ManualAlumno.pdf>
* <https://www.academia.edu/9732490/Instituto_Tecnol%C3%B3gico_De_Tijuana_Estereotipos_e_Interfaces_Integrantes>

En los siguientes videos encontrarás ampliación de estos temas:

* <https://www.youtube.com/watch?v=nFvUZ2Q0CFY&t=311s>
* <https://www.youtube.com/watch?v=7WRYH2ei5Rw&t=275s>
* <https://www.youtube.com/watch?v=gAc7j6gialE>

3. Principios de arquitectura y Diagrama de paquetes

### 

## 3.1 Arquitectura de software

### La arquitectura de software, es un conjunto de patrones con sus correspondientes restricciones que definen la estructura lógica que debe tener el aplicativo de software.

### Una arquitectura de software definida, permite que el equipo de desarrollo conformado por analistas, programadores, diseñadores, testers y arquitectos de software, tengan una idea común de cómo se debe elaborar el sistema.

Según David Garlan and Mary Shaw “más allá de los algoritmos y estructuras de datos de la computación; el diseño y la especificación de la estructura general del sistema emergen como una clase nueva de problema. Los aspectos estructurales incluyen la estructura global de control y la organización general; protocolos de comunicación, sincronización y acceso de datos; asignación de funciones para diseñar elementos; distribución física, composición de elementos de diseño; ajuste y rendimiento; y selección entre otras alternativas de diseño” ([Garlan & Shaw,199](http://cgrw01.cgr.go.cr/rup/RUP.es/LargeProjects/core.base_rup/customcategories/references_56F06DFD.html#GAR93)4).

### Según IEEE “la arquitectura de software es la organización fundamental de un sistema encarnada en sus componentes, las relaciones entre ellos y el ambiente y los principios que orientan su diseño y evolución” (IEEE, 2000).

La arquitectura de software es vista estructural de alto nivel de abstracción y se centra en los requerimientos funcionales.

Para especificar la arquitectura del software de un sistema, se emplean diagramas UML, así como cuerpos de código en algún lenguaje de programación.

Existen diversos paradigmas y frameworks que dirigen la forma y las herramientas que se deben utilizar para la construcción del aplicativo, sin embargo, todas los patrones de arquitectura de software usan al menos tres vistas o modelos para describir la arquitectura:

**La visión estática:** describe qué componentes tiene la arquitectura. Por ejemplo los diagramas de clases y de paquetes.

**La visión funcional:** describe qué hace cada componente. Por ejemplo los diagramas de casos de uso.

**La visión dinámica:** describe cómo se comportan los componentes a lo largo del tiempo y cómo interactúan entre sí. Por ejemplo los diagramas de secuencia y comunicación.

Algunos de los paradigmas más utilizados actualmente para definir la arquitectura del software son:

* [**D**](https://es.wikipedia.org/w/index.php?title=Dise%C3%B1o_Estructurado&action=edit&redlink=1)**escomposición Modular.** Donde cada módulo es una parte funcional de la solución completa, pero muy acoplados, es decir cada módulo depende mucho de los otros módulos para poder funcionar el aplicativo entero.
* **Cliente-servidor.** En el que se ejecutan una parte de las funcionalidades del lado de un computador central (servidor) y otra parte de las funcionalidades en el computador del usuario (cliente). El software reparte su carga de cómputo en dos partes independientes pero sin reparto claro de funciones.
* [**Arquitectura de tres niveles**](https://es.wikipedia.org/wiki/Arquitectura_de_tres_niveles)**.** Es una especialización de la arquitectura cliente-servidor donde la funcionalidad se divide en tres capas. una capa vista, para la presentación de los datos y resultados(interfaz de usuario), otra para los cálculos y acceso a las bases de datos (donde se encuentra modelado el negocio) y otra para el almacenamiento (persistencia). Una arquitectura que especializa el modelo multicapas es la arquitectura [Modelo, **Vista Controlador**](https://es.wikipedia.org/wiki/Modelo_Vista_Controlador)**.** En esta arquitectura, los métodos de las clases del paquete controlador, lanza mensajes (es decir, ejecuta los métodos) para encapsular datos tomados del modelo (repositorio de datos) y mostrar los resultados de los cambios en las vistas (formularios web por ejemplo).
* **Orientada a servicios** (SOA del inglés Service-Oriented Architecture). La Arquitectura Orientada a Servicios (SOA) es un arquitectura se apoya en la orientación a servicios, es decir componentes de software (interfaces) que permiten comunicar dos o más sistemas. Un servicio es una funcionalidad que ejecuta una actividad propia del negocio (ejemplo: comprobar que un estudiante si esté matriculado, que un cliente si reporte los datos completos o que la cuenta de un tarjetahabiente esté al día.

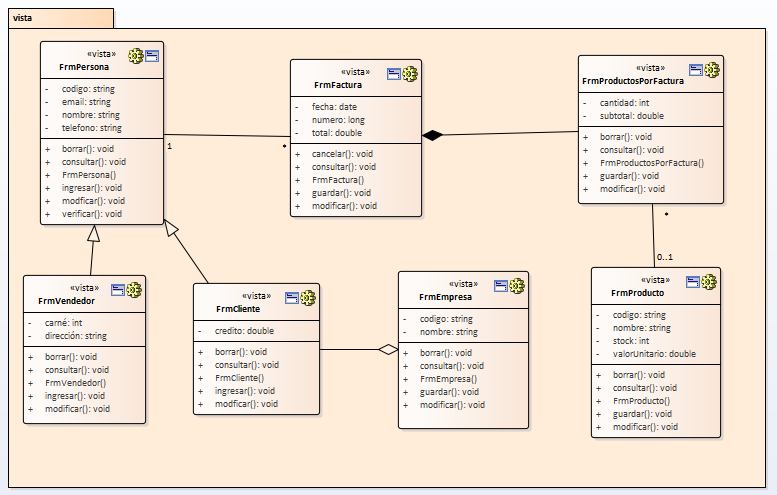
## 3.2 Los paquetes de clase

**Un paquete de clases**, es un conjunto de clases agrupados en un mismo directorio (o biblioteca de clases). Todas las clases de un paquete tienen un objetivo común. Por ejemplo las clases del **Paquete vista**, poseen clases cuyos métodos se encargan de procesar la interfaz gráfica para facilitar la comunicación con los usuarios.

**El paquete modelo**, poseen clases cuyos métodos se encargan de encapsular los datos obtenidos de la base de datos (repositorio de datos-persistencia de datos), por ejemplo, si se tienen código, nombre, teléfono e email de un cliente consultado de la base de datos, se crea un objeto de la clase Cliente para encapsular y así transportar el objeto con los datos incluidos y no los datos por separado.

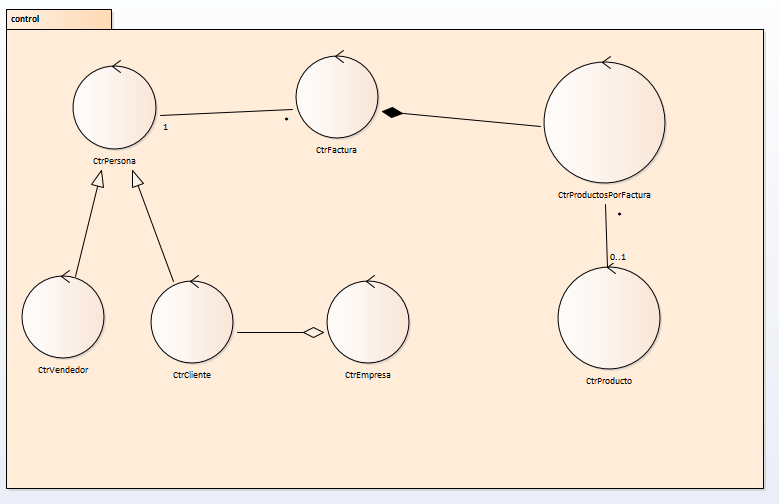
**Las clases del paquete control**, para el proceso de facturas, poseen métodos que se encargan de realizar los cálculos y las validaciones para asegurar que las reglas del negocio se cumplan a satisfacción. Por ejemplo guardar, consultar, modificar y borrar registros de clientes en la base de datos. Validar que no se dupliquen los datos de un cliente. Que los datos de los vendedores estén completos. Que se registren productos en las facturas, en las cantidades admitidas por la empresa entre otros.

Para el caso de un sistema de un proceso de facturas, se emplea un paradigma multicapas, que contempla tres paquetes de clases relacionados entre sí: paquete vista, paquete control y paquete modelo. Los diagramas de paquetes del ejemplo del proceso facturas se presentan a continuación:

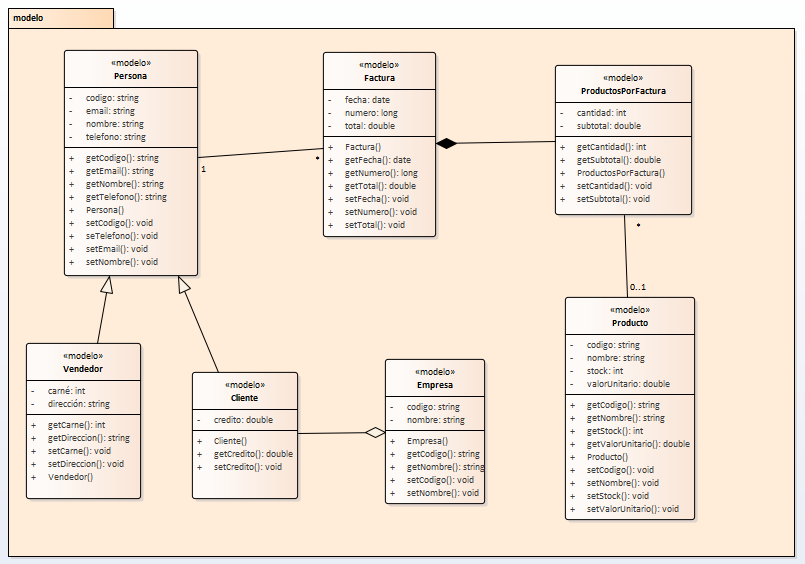


**figura 8. Paquete vista,proceso facturas**

### 



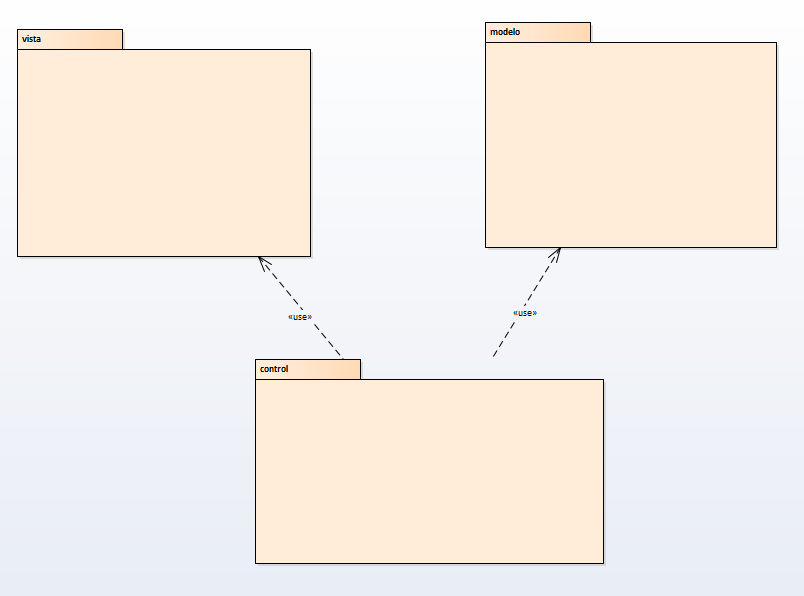
**figura 9. Paquete control, proceso facturas.**

****

**figura 10. Paquete modelo, proceso facturas.**

Las clases del paquete modelo, se especializan en encapsular los datos obtenidos mediante consultas a la bases de datos para mostrarlas al usuario mediante las clases del paquete vista , o también encapsular datos que serán modificados desde las clases del paquete control.

Las clases del paquete control poseen métodos que se comunican directamente con las clases del paquete vista y con las clases del paquete modelo. A continuación se presenta el diagrama de paquetes con sus interrelaciones, para el sistema proceso de facturas:



**figura 11. Diagrama de paquetes, proceso facturas**

### ¡Para Saber más!

Para continuar con el estudio de los conceptos planteados sobre Arquitectura de software y diagramas de paquetes , te recomendamos las siguientes lecturas:

* <https://es.slideshare.net/IsraelRey/qu-es-la-arquitectura-de-software>
* <http://campusvirtual.edu.uy/libro1/capitulo7.pdf>
* <http://jraquelm2.wixsite.com/ingenieriadesoftware/single-post/2015/07/08/-TEMA-9-DIAGRAMAS-DE-PAQUETES-Y-DE-SECUENCIAS>
* <http://www.codecompiling.net/files/slides/UML_clase_05_UML_paquetes.pdf>
* [http://exa.unne.edu.ar/informatica/anasistem1/public\_html/TUTORIAL\_UML[1].pdf](http://exa.unne.edu.ar/informatica/anasistem1/public_html/TUTORIAL_UML%5B1%5D.pdf)
* <https://www.ctr.unican.es/asignaturas/is1/is1-t11-trans.pdf>

En los siguientes videos encontrarás ampliación de estos temas:

* <https://www.youtube.com/watch?v=jOXdN0ZkZtM>
* <https://www.youtube.com/watch?v=ZjxTNGpgP4k>
* <https://www.youtube.com/watch?v=XR15vneBFoU>
* <https://www.youtube.com/watch?v=7vsPNUnhzTA>

# 4. Diagramas Dinámicos: Secuencias y Comunicación

Los diagramas UML, dinámicos se usan para modelar las interacciones entre los componentes del sistema a lo largo del tiempo. En un diagrama dinámico de secuencias o de comunicación (colaboración), se muestra la forma como los objetos colaboran entre sí, enviando mensajes (es decir ejecutando los métodos) para lograr una funcionalidad específica.

Hay varios tipos de diagramas UML, que permiten representar estas interacciones, por lo que se les llama diagramas dinámicos. Estos son:

* **Diagrama de estados**: muestra el conjunto de estados por los que pasa un objeto mientras esté activo en el sistema. El diagrama de estados muestra también los cambios que permiten pasar de un estado a otro. Cada objeto está en un estado en cierto instante. El estado en el que se encuentra un objeto determina su comportamiento, es decir los mensajes que puede enviar y los otros estados por los que puede pasar. Por ejemplo un objeto cuentaCorriente, puede pasar por los estados: saldo Rojo, Saldo Positivo o saldo Cero. Si está en estado saldoRojo, enviará un mensaje de alerta. Si está en estado saldoPositivo, puede enviar mensajes que permita retirar de la cuenta. Todo Objeto pasa por un estado inicial, que se presenta cuando se crea el objeto (es decir, cuando se instancia la clase) y un estado final, que se presenta cuando se destruye el objeto.

A continuación se presenta el diagrama de estados para el objeto cuentaCorriente:

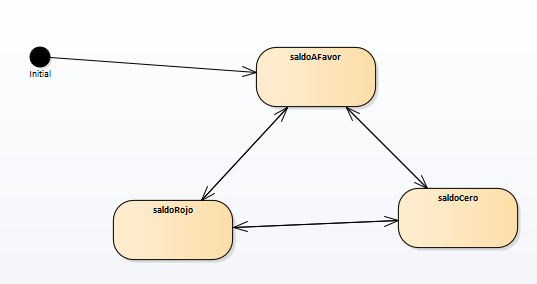


figura 12. Ejemplo de diagrama de estados

* **Diagrama de actividad:**los diagramas de actividad son utilizados para modelar el flujo de trabajo interno de una operación. Un diagrama de actividades podría reemplazar un diagrama BPMN, para modelar los procesos del negocio.

Un ejemplo de diagrama de actividades, es aquel que representa el proceso de un sistema de pedidos:

tomado de:

<https://www.docsity.com/es/un-proceso-basado-en-uml-para-sistemas-de-informacion-ejemplo-de-aplicacion-apuntes-ingenieria-del-software/329797/>

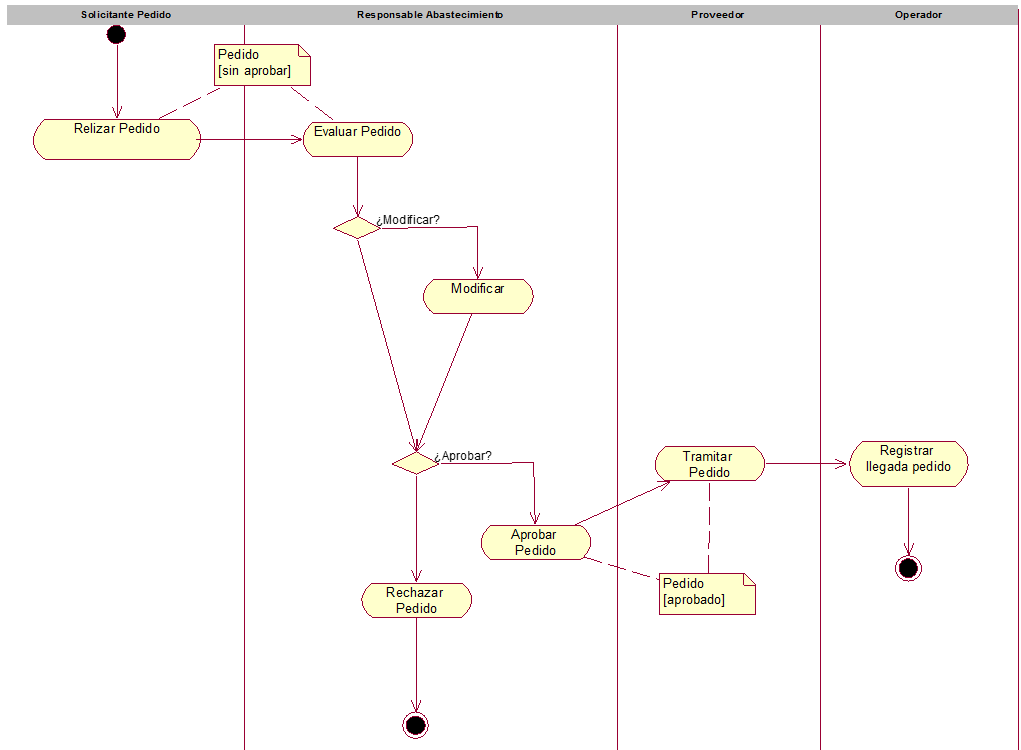


figura 13. Ejemplo de diagrama de actividades

El solicitante de pedido, realiza un pedido por algún medio (telefónico o por computador), se genera un pedido sin aprobar al responsable de abastecimiento, que se encarga de evaluar el pedido. Puede ser que el pedido requiera modificaciones. Se hagan modificaciones o no, el pedido puede ser o no aprobado. Si se toma la decisión de rechazar el pedido, se termina el proceso. Si se toma la decisión de aprobar el pedido, se envía el pedido aprobado al proveedor, que se encarga de tramitar el pedido. Una vez llegue el pedido al lugar solicitado, el operador se encarga de registrar el pedido y termina el proceso.

* **Diagramas de interacción**: Diagrama de secuencia y Diagrama de comunicación.

Los diagramas de interacción describen la forma como los objetos (es decir instancias de clases) colaboran entre sí para lograr la solución de una funcionalidad del sistema. En otras palabras, en los diagramas de interacción se muestran los mensajes que se envían los objetos para conseguir una funcionalidad del sistema especificada en uno o varios casos de uso. Por ejemplo sis se tiene un caso de uso en el que se especifica el ingreso, consulta, modificación y borrado de un cliente al sistema como el que se muestra a continuación:

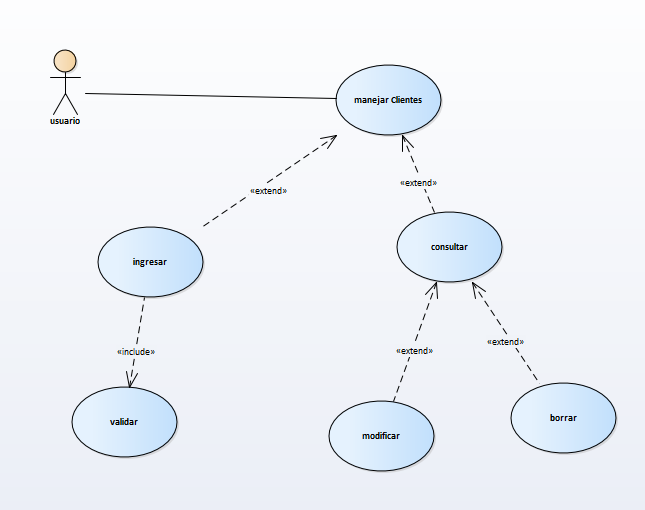


figura 14. Ejemplo de diagrama de casos de uso manejar cliente

### 

los diagramas de interacción (secuencia y/o comunicación), describen la forma como los objetos del sistema interactúan entre sí (enviando mensajes) a lo largo del tiempo, para lograr que cada funcionalidad sea posible (es decir ingresar, consultar, modificar y borrar).

Un diagrama de secuencia que describe el caso de uso ingresar cliente es el siguiente:



figura 15. Ejemplo diagrama de secuencia para ingresar Cliente

El diagrama de secuencia ingresar cliente, describe lo siguiente: el proceso comienza cuando un objeto de la clase del paquete control (un objeto de la clase CtrCliente), le envía un mensaje al objeto de la clase vista (FrmCliente). Este mensaje consiste en ejecutar el método ingresar. El usuario digita los datos del cliente. El objeto de la clase vista retorna un mensaje de confirmación. El objeto de la clase control envía un mensaje al objeto de la clase modelo (Cliente), para encapsular los datos que digitó el usuario (es decir crear un objeto objCliente con los datos del cliente) y retorna un mensaje de confirmación. El objeto de la clase control se envía a un mensaje así mismo, e invoca el método validar y envía el mensaje de validación al objeto de la clase que está en el paquete vista. Si el ingreso es correcto, debería mostrar el mensaje “cliente ingresado”, de lo contrario debe mostrar un mensaje de error. Nótese que el diagrama de secuencia muestra el envío de cada mensaje (es decir invocar métodos) sobre una línea de tiempo que comienza en la parte de arriba con el primer mensaje y termina en la parte de abajo del diagrama con el último mensaje. Es decir cada mensaje se envía de manera secuencial.

El diagrama de colaboración o comunicación, describe el mismo proceso expuesto anteriormente, sólo que se centran en las responsabilidades de cada objeto, en lugar del tiempo en que los mensajes son enviados.

A continuación se presenta un ejemplo de diagrama de comunicación para funcionalidad ingresar cliente:

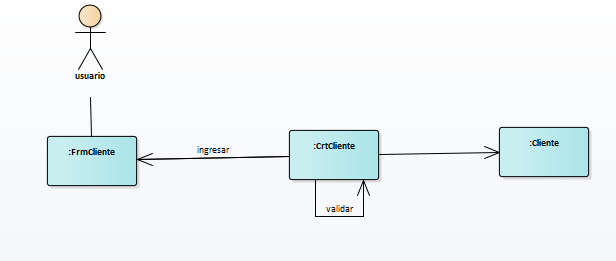


figura 16. Diagrama de comunicación para ingresar Cliente

### ¡Para Saber más!

Para continuar con el estudio de los conceptos planteados sobre Diagramas Dinámicos, de secuencia y comunicación, te recomendamos las siguientes lecturas:

* <http://www.elai.upm.es/webantigua/spain/Asignaturas/InfoInd/apuntesAOOD/cap5UMLDinamicoImpl.pdf>
* <https://es.scribd.com/doc/28016877/Diagramas-Dinamicos>
* <https://www.ctr.unican.es/asignaturas/MC_OO/Doc/M_dinamico.pdf>
* <https://www.ibm.com/support/knowledgecenter/es/SS6RBX_11.4.3/com.ibm.sa.tutorial.doc/topics/Less2.1_BuildASequenceDiagram.html>

En los siguientes videos encontrarás ampliación de estos temas:

* <https://www.youtube.com/watch?v=9sNGHPZPsZw>
* <https://www.youtube.com/watch?v=ruoylFi7Irs>
* <https://www.youtube.com/watch?v=jOXdN0ZkZtM&t=13s>
* <https://www.youtube.com/watch?v=7vsPNUnhzTA&t=217s>

# 5: Diseño de la interfaz de usuario, Diseño de pantallas, Diseño de reportes, Construcción de código

## 5.1 Diseño de la interfaz de usuario

### El diseño de interfaces de usuario también conocido como UI en español o por su sigla en ingles HCI (Human Computer Interfece), es una actividad que en la actualidad ha adquirido una gran importancia en el desarrollo de cualquier tipo de aplicativo informático. La calidad con la cual se implemente la interfaz de usuario puede llegar a ser uno de los motivos que lleven a un aplicativo al éxito o al fracaso. De aquí la importancia que los desarrolladores de software y los diseñadores de páginas web conozcan los conceptos básicos sobre los cuales se debe implementar una adecuada la interfaz de usuario, de acuerdo a la definición en Wikipedia “La interfaz de usuario es el medio con que el usuario puede comunicarse con una máquina, equipo, computadora o dispositivo, y comprende todos los puntos de contacto entre el usuario y el equipo. La interfaz de usuario es el espacio donde se producen las interacciones entre seres humanos y máquinas. El objetivo de esta interacción es permitir el funcionamiento y control más efectivo de la máquina desde la interacción con el humano ”

### En general una aplicación informática posee una parte invisible y otra parte visible, en la parte invisible podemos encontrar las actividades de cálculos, almacenamiento y recuperación de información entre otras actividades, y la parte visible de la aplicación es la interfaz de usuario también llamada front-end (inglés) con la que el usuario interactúa con la finalidad de usar el software. Los usuarios pueden controlar y manejar el software como también el hardware por intermedio de las interfaces de usuario. En la actualidad encontramos las interfaces de usuario casi en todos los lugares donde existe tecnología, la vemos en computadores, tablets, dispositivos móviles, televisores, electrodomésticos, autos, barcos, aviones, etc.

### La interfaz de usuario es parte del software, como norma general esta debe ser fácil de entender y fácil de manejar, en el lenguaje informático nos referimos a que una interfaz debe ser "amigable e intuitiva". La UI puede implementarse en forma gráfica, o solo texto, audiovisual, así mismo una interfaz puede ser un software, un hardware o una combinación de ambos.

### ¿Qué tipos de UI existen?

### En la actualidad existe una gran diversidad de UI, en los principios de la informática se utilizan las interfaces básicas que solo utilizaban textos en líneas de Command Line Interfaces (CLI), inclusive hoy se siguen utilizando en diversos aplicativos, pero en este tema se han logrado grandes avances, a continuación te presentamos algunas tipos de UI y sus principales características.

### Líneas de comando

### También llamadas CLI(Command Line Interfaces), estas consisten en básicamente una pantalla oscura en la cual se presenta el prompt o punto de espera de comandos, el cual consiste en una barral vertical que parpadea y que permite digitar textualmente los comandos que dan a la computadora las instrucciones a ejecutar. Una vez realizados los comandos presenta en pantalla en forma textual los resultados, El control mediante línea de comandos se hace por medio del teclado y sin necesidad de usar el Mouse.

### Los CLI se encuentran comúnmente en los sistemas operativos para interactuar con el kernel y se les denomina shell. Algunos ejemplos de estos shell son los utilizados por Unix (Bourne) y Windows(CMD). Algunas aplicaciones proporcionan tanto una CLI como una GUI , algunos ejemplos serían:el programa de CAD “AutoCAD” , Matlab, Python Shell y la línea de comando del video juego “Minecraft”.

### La siguiente es una imagen de línea de comandos utilizada en Windows en la cual se da el comando “dir” para que el sistema operativo suministre una lista de archivos y carpetas de la unidad de disco “C”:

### 

### figura 17. Interfaz línea de comandos

### Las interfaces gráficas de usuario

### Las Interfaces gráficas de usuario, comúnmente llamadas GUI (Graphical User Interfaces) son en la actualidad el estándar más utilizado en la industria informática. Estas se crean utilizando elementos gráficos de control (cajas de texto, listas desplegables, cajas de chequeo etc.) e imágenes simbólicas que se toman como referencia de objetos del “mundo real”.

### Para su utilización los usuarios requieren usar el ratón y el teclado como dispositivos de control, pero también cada vez son más utilizadas las pantallas táctiles que funcionan mediante toques con el dedo. El diseño gráfico GUI se basa en la mayoría de los casos en los elementos de una oficina, esto conlleva a que los elementos pueden identificarse de forma sencilla y su manejo resulta muy ser muy intuitivo, para que los usuarios con menos experiencia pueden entender la funcionalidad de los iconos con facilidad, muchos de estos iconos no requieren explicaciones para su comprensión. Este tipo de simbología se ha convertido en un factor primordial para las GUI.

### 

|  |
| --- |
| Las interfaces básicas de usuario son aquellas que incluyen elementos como menús, ventanas, contenido gráfico, cursor, los *beeps* y algunos otros sonidos que la computadora hace, y en general, todos aquellos canales por los cuales se permite la comunicación entre el ser humano y la computadora.(Wikipedia) |

### 

### Componentes básicos de la GUI

### Las GUI utilizan un conjunto de elementos básicos para interactuar con el software o con el hardware. Cada elemento gráfico permite una forma de trabajo con el sistema. En Windows un GUI contiene algunos de los siguientes elementos:

**f**

Figura 18. Interfaz gráfica

1.  **Menú de control:** es el icono ubicado en la parte superior izquierda de la barra de título de la ventana, ofrece un menú en el cual se encuentran acciones que se pueden realizar con la ventana.

2. **Barra de título:** Aquí se presenta el nombre de la ventana o aplicativo se está utilizando.

3. **Iconos de manipulación de la ventana**: son tres botones situados en la parte superior derecha de la ventana permiten minimizar, ocultar o mostrar la ventana.

4. **Barra de menús:** Presenta un conjunto de opciones que ofrecen las acciones que se pueden realizar con la aplicación. Dependiendo de la aplicación esta barra de menú es distinta.

5.  **Barras de herramientas:** son un conjunto de botones dispuestos en una barra horizontal o vertical, que permiten acceder fácilmente a diversas funcionalidades del programa, de esta forma los usuarios no requieren entrar a los menús para ejecutar una acción.

6. **Barra de dirección**: muestra la posición en la que estamos ubicados dentro del sistema.

7. **Panel de tareas:** ofrece acceso a las tareas más comunes que se pueden realizar con el objeto que este seleccionado.

8.  **Barras de desplazamiento y bordes:** estas barras aparecen cuando el contenido de la ventana no se puede visualizar en su totalidad y dependiendo del tamaño de la ventana.

### 

### Voice User Interface (VUI)

Esta es una nueva tecnología que permite la interacción con los dispositivos a través de la utilización de comandos de voz, son diversas las herramientas que se han implementado en los diversos sistemas operativos para ofrecer este tipo de interfaces, entre los más destacados encontramos SIRI de apple, amazon ha desarrollado ALEXA y microsoft ha implementado CORTANA. Este tipo de interfaces ofrecen tiene grandes ventajas entre ellas la accesibilidad, la cual permite a las personas con discapacidad la facilidad de manejo de los dispositivos sin necesidad de tener un contacto físico.

**Otras tecnologías de UI**

Además de las interfaces expuestas encontramos una gran variedad de tecnologías de interfaces de usuario las cuales en la actualidad se están implementando, algunas de estas las describimos a continuación para que investigues y profundices en ellas:

* Natural User Interface (NUI)
* Brain Computer Interfaces (BCI)
* Tangible User Interfaces (TUI)
* Perceptual User Interfaces (PUI)

## 5.2 Diseño de pantallas

Para el diseño de las pantalla por lo general se desarrollan bocetos o dibujos rápidos para tener una imagen visual de cómo será nuestra interfaz gráfica (GUI). A estos dibujos se les denomina sketchs, los cuales son los prototipos iniciales con los que diseñamos nuestra interfaz gráfica de nuestro sistema.. Mientras se diseña a lápiz, realizamos cambios, modificamos, tenemos mejor ideas y estos sketchs se convierten en el prototipos del proyecto final, en definitiva en diseños reales( Mockups). A este proceso se le denomina Wireframing que es la creación de la estructura de una interfaz.

Los elementos que hacen parte de una la interfaz gráfica son componentes visuales, y por medio de ellos los usuarios logran interactuar con la aplicación, En esta interacción los usuarios ingresan los datos que el programa utiliza realizar su funcionalidad y presenta los resultados de procesar estos datos. Los elementos gráficos están compuestos por las ventanas, imágenes .iconos, botones y otros tipos de elementos, detallamos a continuación algunos de los elementos gráficos que componen una aplicación:

### Ventanas de aplicación: Las ventanas de aplicación son aquellas que contienen a todos los elementos que hacen parte del aplicación.

### Ventanas: son elementos encargados de alojar a otros elementos gráficos.

### Cuadros de diálogo: Estos son ventanas que por lo general se presentan por un breve periodo de tiempo en la pantalla. Se utilizan para informar a los usuarios de alguna situación o solicitar datos en un determinado contexto.

### Componentes: Involucra a los elementos de una interfaz gráfica con entidad propia y una funcionalidad asociada a componentes. Ejemplos de estos son las imágenes, botones,, etiquetas, listas desplegables, menús, etc. No se consideran componentes, las letras, las líneas, los colores, los píxeles, etc

### Controles: son todos aquellos componentes que pueden recibir información de los usuarios cuando interactúan con el programa mediante el teclado o el mouse. Los controles más comunes son: los cuadros de texto, los botones, barras de desplazamiento, cuadros de chequeo, listas desplegables.

### Contenedores: estos son elementos que tienen la capacidad mostrar en su interior otros componentes.

### Menús: Los menús son elementos que contienen botones distribuidos verticalmente. La pulsación de uno de estos botones abrirá un nuevo menú o bien iniciará alguna acción de la aplicación. Los menús pueden aparecer al pulsar el botón secundario del ratón sobre algunos elementos de la interfaz. Si el contenido del menú depende del elemento pulsado, se denomina menú contextual.

### Barras de menús: Las barras de menús por lo general aparecen en la parte superior de las ventanas. Se componen de una barra horizontal con botones, que al ser pulsados despliega verticalmente un conjunto de opciones.

### La lista de controles que pueden utilizarse para confeccionar una interfaz de usuario es muy extensa, pero existe un grupo cuyo utilización es más frecuente para diversos ambientes como lo son las aplicaciones de escritorio, aplicaciones web y aplicativos para móviles, entre estos podemos destacar:

### Caja de texto : utilizadas para que el usuario e introduzca datos en texto.

### Botones: se usan para enviar entradas al software

### botón de Radio: presentan diversas opciones para selección. Sólo se puede seleccionar una de entre todas las que se ofrecen.

### Casilla de verificación : Las funciones son similares a los botones de radio. Pero permite seleccionar múltiples opciones representadas por casillas de verificación.

### Cuadro de lista : ofrecen una lista de los ítems disponibles para realizar la selección.

### Combo-box (casilla combinada)

### Data-grid (Cuadrícula de datos)

### Drop-down list (Lista desplegable)

### 

## 5.3 Diseño de reportes

### Una de las funcionalidades más importantes para los usuarios de un sistema informático es la salida de información. Si la salida no tiene calidad, entonces todo el sistema puede parecer a los usuarios poco útil y evitarán utilizarlo y esto indudablemente podrá conducir al fracaso de un proyecto informático. Los usuarios requieren de la salida de un sistema para realizar sus tareas, y por lo general juzgan el mérito de un sistema únicamente por su salida.

### Es por esto que es muy importante identificar, detallar y especificar las necesidades de los usuarios del sistema, con el objeto de implementar salidas eficientes y eficaces.

### Según Senn(1992) el término salida se utiliza para denotar “cualquier información producida por un sistema de información ya sea impresa o en una pantalla”.

### Para implementar en sistema la salida más útil posible, los desarrolladores del sistema deberán trabajan de cerca con los usuarios, por medio de un proceso interactivo y continuo hasta que el resultado se considera satisfactorio, Los desarrolladores del sistema deben decidir cuándo imprimir, desplegar en pantalla o presentar su salida en otros formatos. La salida impresa puede realizarse en papel en blanco o formas preimpresas, la salida visual puede utilizar una o varias pantallas para mostrar información.

### 

### Tipos de salida

### La salida de un sistema de información puede ser:

### Un reporte

### Un documento

### Un mensaje

### De acuerdo con las circunstancias y los contenidos, la salida puede ser impresa o presentada en una pantalla

### Salida tipo reportes

### Podemos considerar los reportes como informes que organizan y exhiben la información contenida en una base de datos. Estos son generados por las aplicaciones y nos muestra de forma organizada, estructurada y/o Resumida los datos guardados o generados por la misma aplicación de tal manera que se vuelvan útiles , Su función es aplicar un formato específico a los datos para ser presentados por medio de un diseño atractivo y que sea fácil de interpretar por los usuarios del sistema.

### Los reportes generalmente agrupan los datos de acuerdo a un interés específico; de acuerdo a las necesidades del usuario o acorde a los requisitos específicos de un área particular de la una empresa.

### 

### Aspectos importantes para la creación de reportes

### Quiénes recibirán el reporte: usuarios internos y externos.

### Cuál es el uso que se le pretende dar: el uso determina el contenido, la forma y medio a utilizarse para su generación.

### Cuántos detalles son necesarios: la cantidad de datos sugiere si deben emplear métodos de impresión o de presentación en una pantalla.

### Cuándo y con qué frecuencia es necesaria la salida:

### Qué método utilizar: impresa o presentada en pantalla

### Para la presentación de los informes se pueden realizar de las siguientes formas:

### Informes en formato Tabular

### En general, los usuarios de un sistema están acostumbrados a recibir la información en formas de tablas. El formato tabular debe utilizarse bajo las siguientes condiciones:

### Cuando los detalles son presentados en categorías discretas

### Cuando cada categoría debe tener una etiqueta.

### Cuando se deben obtener totales o realizar comparaciones entre diversos componentes.

### Se desea presentar la información en un formato que presente los detalles en un orden significativo (quizás de mayor a menor, por código o alfabéticamente), aquel donde estos sean más sencillos de localizar.

### La siguiente imagen es un ejemplo de una pantalla utilizada para la generación de informes:

|  |
| --- |
|  |

Figura 19. Interfaz gráfica para generador de reportes

### De acuerdo a lo que se aprecia en la imagen, la pantalla cuenta con los siguientes componentes:

* Título del informe
* Opciones de exportación del informe o de impresión(excel. word,pdf, css)
* Filtros: permite selección de la información de acuerdo a categorías
* Fechas de generación: por lo general los informes se realizan entre un rango de fechas.
* Opciones de búsqueda : utilizadas para consultas en pantalla de datos específicos.
* Tablas con columnas que presentan la información consultada.

### Formato Gráfico

### Existen en el mercado un sinnúmero de herramientas gráficas para diseñar de mejor manera la salida. El uso de gráficos ayudarán a mejorar la comprensión sobre ciertos temas, sobretodo estadísticos.

### Los tipos de gráficos comúnmente usadas:

### ·Gráfica de sectores

### ·Gráfica de curvas

### ·Gráfica de escalones o superficie

### ·Gráfica de barras y columnas

### ·Mapas

### La siguiente imagen representa un informe gráfico en el cual se consolida la información de acuerdo alas necesidades de los usuarios del sistema:

### 

|  |
| --- |
|  |

Figura 20. Reporte gráfico con datos estadísticos

## 5.4 Construcción de código

### Para indicar a las computadoras lo que deseamos que haga, requerimos de un lenguaje de programación, Un Lenguaje de Programación es básicamente un conjunto de instrucciones que son entendibles y ejecutables por un computador y las cuales deben tener unas reglas de sintaxis y reglas semánticas adecuadas para que puedan funcionar, estos lenguajes luego deben ser convertidos a códigos de máquina y códigos binarios los cuales son entendidos por la computadora.

### Los lenguajes utilizados para la programación se clasifican en: el de lenguajes de alto nivel, lenguajes de bajo nivel y lenguaje de máquina.

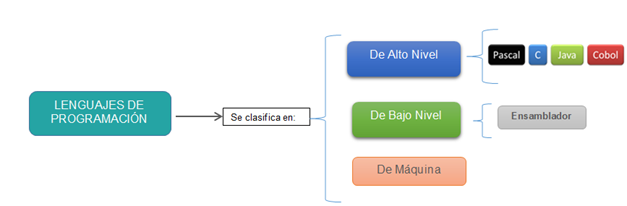


Figura 21. Clasificación de lenguajes

### Lenguaje de Máquina

### Cómo decíamos anteriormente el lenguaje máquina es el único lenguaje que puede ejecutar una computadora, este lenguaje solo utiliza dos símbolos: el cero (0) y el uno (1). Es por esto que al lenguaje máquina también se le denomina lenguaje binario, es un lenguaje específico en cada arquitectura, El lenguaje de máquina es un código que es interpretado directamente por el microprocesador. Es un lenguaje que los programadores no utilizamos directamente debido a su complejidad.

### Lenguaje de bajo nivel:

### Es el lenguaje de programación más entendible por las computadoras y los programadores, está compuesto por un conjunto de instrucciones que son extremadamente sencillas para ser entendidas por el computador, pero que son algo complejas de manejar para los humanos, este tipo de programación se conoce como Lenguaje “Ensamblador”, las instrucciones está basadas en una serie de mnemónicos (palabra que sustituye a un código que ejecuta una acción en la máquina) , Este Lenguaje es el más entendible por las computadoras. La programación en Assembler fue la que se utilizó en los primeros años de la historia de la Programación. Esta programación estaba sólo destinada a unos pocos sabios de la computación ya que era compleja de manejar, Cada vez los programadores tenían que ser mucho más especializados dada la complejidad de entendimiento de las instrucciones utilizadas en este lenguaje. Debido a esto se hace necesario desarrollar lenguajes de programación que estén más al alcance de todos y no de unos pocos, es así como surgieron los lenguajes de alto nivel.

### Lenguaje de alto nivel.

### Se consideran de alto nivel aquellos lenguajes que son más parecidos al lenguaje natural humano y más lejano al lenguaje de las máquinas. Un lenguaje de alto nivel permite al programador escribir las instrucciones de un programa utilizando palabras o expresiones sintácticas muy similares al inglés, Ésta es la razón por la que a estos lenguajes se les considera de alto nivel, porque se pueden utilizar palabras de muy fácil comprensión para el programador, es más flexible, fácil de aprender y más apropiado para corregir los errores de programación.

### Los lenguajes de alto nivel deben ser traducidos para que sean comprensibles para la máquina, este proceso se realiza en forma automática a través de programas especializados que se denominan ensambladores o compiladores.

### Los compiladores realizan la verificación que los programas escritos por los programadores cuenten con las reglas de sintaxis adecuada para que el computador pueda entendernos.

### 

### Figura 22: proceso de creación y compilación de programas.

### Es muy normal que al escribir un programa, omitamos algunas reglas y se nos vayan, sin querer, algunos errores de escritura. Por ejemplo que en alguna parte del programa abrimos un paréntesis que luego se nos olvidó cerrar o que no escribimos una palabra correctamente. Para ello el compilador revisa la sintaxis del programa, nos dice si tiene errores, nos orienta en qué líneas del programa están los errores y hasta nos sugiere la corrección, a este proceso se le llama compilación.

### Cuando se realiza el proceso de compilación se reportan los siguientes tipos de errores:

### Errores de Sintaxis.- Podríamos comparar los errores de sintaxis en un Lenguaje de Programación con los errores de Ortografía en nuestro idioma. Son todos aquellos errores que se presentan en la omisión de alguna o algunas reglas sintáctica, Por ejemplo cuando escribimos mal una palabra propia del lenguaje(Ej. Escribimos Exprimir en vez de Imprimir ) o es normal que en ocasiones, en las expresiones matemáticas, abramos un paréntesis que posteriormente se nos olvida cerrar... entonces al momento de compilar, el compilador nos indicará precisamente este error. Es de aclarar que mientras se reporten este tipo de errores el compilador no se creará el programa ejecutable. Es tarea del programador corregir estos errores en su código fuente para poder generar finalmente el programa ejecutable.

### Errores de Precaución.- son orientaciones o recomendaciones que el compilador realiza para mejorar nuestros programas.

### Es de anotar que existe otro tipo de error el cual no es detectado por el compilador y son los errores de lógica, estos se presentan cuando el programador escribe correctamente la sintaxis del programa pero las instrucciones dadas al programa son erróneas, por ejemplo al realizar un cálculo matemático escribimos mal la fórmula, esto conlleva a que el programa si funcione pero nos dará malos resultados en los resultados, es por esto que se requiere todo un proceso de pruebas de los programas para verificar que los resultados arrojados por los programas sean correctos.

### Ejemplos de lenguajes de programación de alto nivel

### En la actualidad existen gran cantidad de lenguajes de programación, muchos de ellos creado con fines específicos (empresariales, científicos, inteligencia artificial) ,entre los más destacados encontramos:

### 1. Java : Reconocido por su legibilidad y simplicidad, Java es uno de los lenguajes de programación más populares en el mundo.

### 2. C : Creado entre 1969 y 1972 en los Laboratorios Bell, es uno de los más utilizados en el mundo. es de propósito general, con lo cual es muy flexible. Es muy utilizado para el desarrollo de aplicaciones de escritorio.

### 3. C++ :Conocido por el nombre “C Plus Plus”, es un lenguaje de programación orientado a objetos surge como una continuación y ampliación del C.

### 4. C# :También llamado “C Sharp”, este lenguaje de programación desarrollado por Microsoft para ser empleado en una amplia gama de aplicaciones empresariales. C Sharp es una evolución del C y C++ que se destaca por su sencillez y modernidad.

### 5. Python : Un lenguaje de programación multiplataforma y multiparadigma, que también es de propósito general. Su simpleza, legibilidad y similitud con el idioma Inglés lo convierten en un gran lenguaje ideal para principiantes.

### 6. PHP : lenguaje especializado para la creación de aplicaciones para la Web. Es de fácil acceso para nuevos programadores .

### 7. Visual Basic. NET : desarrollado por Microsoft. Es utilizado por personas que no cuentan con conocimientos profundos como desarrolladores, su sintaxis es sencilla. Es visto como uno de los lenguajes más amigables para los que recién comienzan la programación.

### 

### 

### ¡Para Saber más!

Para continuar con el estudio de los conceptos planteados sobre interfaces gráficas, te recomendamos las siguientes lecturas:

* Pressman, R. (2010). INGENIERÍA DEL SOFTWARE. UN ENFOQUE PRÁCTICO: CAPITULO 11 D ISEÑO DE LA INTERFAZ DE USUARIO. México: McGRAW-HILL INTERAMERICANA EDITORES, S.A. DE C.V. En la web:

<http://cotana.informatica.edu.bo/downloads/ld-Ingenieria.de.software.enfoque.practico.7ed.Pressman.PDF>

* Sommerville, l. (2005). INGENIERÍA DEL SOFTWARE. Séptima edición: Diseño de interfaces de usuario capítulo 16. México: Pearson Educación, S.A. En laWeb:

<https://ulagos.files.wordpress.com/2010/07/ian-sommerville-ingenieria-de-software-7-ed.pdf>

Páginas Web

* <https://es.slideshare.net/ivancmontero/elemento-tipicos-de-las-interfaces-graficas-de-usuario>
* UI: qué define a una buena interfaz gráfica de usuario:

<https://www.1and1.es/digitalguide/paginas-web/diseno-web/ui-que-es-una-interfaz-de-usuario/>

* Estudio sobre el diseño de GUIs (III): La importancia del diseño: <http://blog.txipinet.com/2006/08/28/28-estudio-sobre-el-diseno-de-guis-iii-la-importancia-del-diseno/>

En el siguiente videos encontrarás ampliación de estes tema:

* <https://www.youtube.com/watch?v=xGzeEUHcsj8&t=162s>

**Referencias bibliográficas**

Pressman, R. (2010). INGENIERÍA DEL SOFTWARE. UN ENFOQUE PRÁCTICO . México: McGRAW-HILL INTERAMERICANA EDITORES, S.A. DE C.V.

Sommerville, l. (2011). INGENIERÍA DEL SOFTWARE. Séptima edición. México: Pearson Educación, S.A.

Fernández Muñoz, L., & Alvarez, M. (7 de Septiembre de 2015). Teoría de programación orientada a objetos. Recuperado el 18 de Junio de 2017, de Desarrollo Web:<https://desarrolloweb.com/manuales/teoria-programacion-orientada-objetos.html>

Grupo de Investigación en Ingeniería de Software del Tecnológico de Antioquia(Octubre de 2009).Lógica y programación orientada a objetos: Un enfoque basado en problemas, Recuperado el 18 de Junio de 2017, de<https://issuu.com/carlosarturo.castrocastro/docs/libro_logica_y_programacion_de_objetos>

lópez, P., & Ruiz, F. (s.f.). Lenguaje Unificado de Modelado - UML. Obtenido de Universidad Cantabria – Facultad de Ciencias: <https://ocw.unican.es/pluginfile.php/1403/course/section/1792/is1-t02-trans.pdf>

|  |
| --- |
|  |

**Glosario**

* ***Abstraction****: Propiedad y/o técnica de software que oculta los detalles de la implementación.*
* **Caso de uso:** especificación de un tipo de interacción con un sistema.
* **Clase de objetos:** una clase de objetos define los atributos y operaciones de los objetos. Los objetos se crean en tiempo de ejecución mediante la instanciación de la definición de la clase. El nombre de la clase de objetos se puede utilizar como un nombre de tipo en algunos lenguajes orientados a objetos.
* **Cornmon Request Broker Architecture** (CORBA) Conjunto de estándares propuesto porel OMG que define un modelo de objetos distribuido y las comunicaciones de los objetos.
* **Componente**: unidad de software independiente y desplegable que se ha definido completamente y a la que se accede a través de un conjunto de interfaces.
* **Desarrollo orientado a objetos**: enfoque para el desarrollo de software en el que las abstracciones fundamentales en el sistema son objetos independientes. Se utiliza el mismo tipo de abstracción durante la especificación, diseño y desarrollo.
* **Diagrama de secuencia:** diagrama que muestra la secuencia de interacciones necesarias para completar alguna operación. En UML. los diagramas de secuencias se pueden asociar con los casos de uso.
* **Diagrama de estado**: es un método de representación del comportamiento de un sistema que ilustra sus estados y los eventos que ocasionan que el sistema cambie de estado. Un estado es cualquier modo de comportamiento observable desde el exterior.
* **Diseño de enlaces de usuario**: proceso de diseñar el modo en el que los usuarios del sistema acceden a la funcionalidad del sistema y la forma en la que se visualiza la información producida por el sistema.
* **Interfaz**: especificación de los atributos y operaciones asociados con un componente software. La interfaz es utilizada como el medio de tener acceso a la funcionalidad del componente.
* **Lenguaje de Modelado Unificado (UML):** lenguaje gráfico utilizado en el desarrollo orientado a objelos que incluye varios tipos de modelos del sistema que proporcionan distintas vistas de un sistema. UML se ha convertido en un estándar de facto para el modelado orientado a objetos.
* **Lenguaje de restricciones de objetos (OCL):** lenguaje que forma parte de UML, utilizado para definir predicados que se aplican a las clases de objetos e interacciones en un modelo UML.
* **Lenguaje Estructurado de Consultas SQL):** lenguaje estándar utilizado para la programación de bases de datos relacionales.
* **Modelo de objetos:** modelo de un sistema software que se estructura y organiza como un conjunto de clases de objetos y las relaciones entre estas clases. Pueden existir varias perspectivas diferentes en el modelo, como una perspectiva del estado y una perspectiva de la secuencia.
* **Object Management Group (OMG):** grupo de compañías formado para desarrollar estándares para el desarrollo orientado a objetos. Ejemplos de estándares promovidos por el OMG son CORBA, UML y MDA.
* **Ocultación de información:** utilización de construcciones de lenguajes de programación para ocultar la representación de las estructuras de datos y controlar el acceso externo a estas estructuras.
* **Principios de diseño de las interfaces de usuario**: conjunto de principios que expresan buenas prácticas para el diseño de interfaces de usuario.
* **Prototipado Mago de Oz:** enfoque para el prototipado de las ¡nterracesde usuario enel que los comandos introducidos por los usuarios son interpretados por una persona quien responde como si fuera la computadora.
* **Sistema de objetos distribuidos:** sistema distribuido en el que los componentes ejecutables son objetos.
* **Tipo abstracto de datos**: tipo cuya representación se oculta y está definido por sus operaciones.

Fuente Glosario:

Sommerville, l. (2011). INGENIERÍA DEL SOFTWARE. Novena edición. México: Pearson Educación, S.A.