Desarrollo de contendido

Unidad 1

Programación Orientada a Objetos II

IU Digital de Antioquia
INSTITUCIÓN UNIVERSITARIA DIGITAL DE ANTIOQUIA

ORD No 74 de 2017 · VIGILADA MinEducación

www.iudigital.edu.co

El lenguaje de modelado unificado (UML) y paquetes en JAVA

Presentación

En este curso aprenderás a establecer las bases de la aplicación de los paradigmas de la Programación Orientada a Objetos, a través de herramientas que te permitan aplicar los principios y características de todos estos paradigmas, de tal manera que puedas fortalecer todas tus habilidades en el desarrollo de programas computacionales.

Antes de continuar con la Actividad de Conocimientos Previos, analiza el video de presentación, los objetivos y el mapa del curso, estos insumos te darán un panorama general sobre las temáticas que estudiarás y las competencias que debes desarrollar al finalizar el estudio del curso.

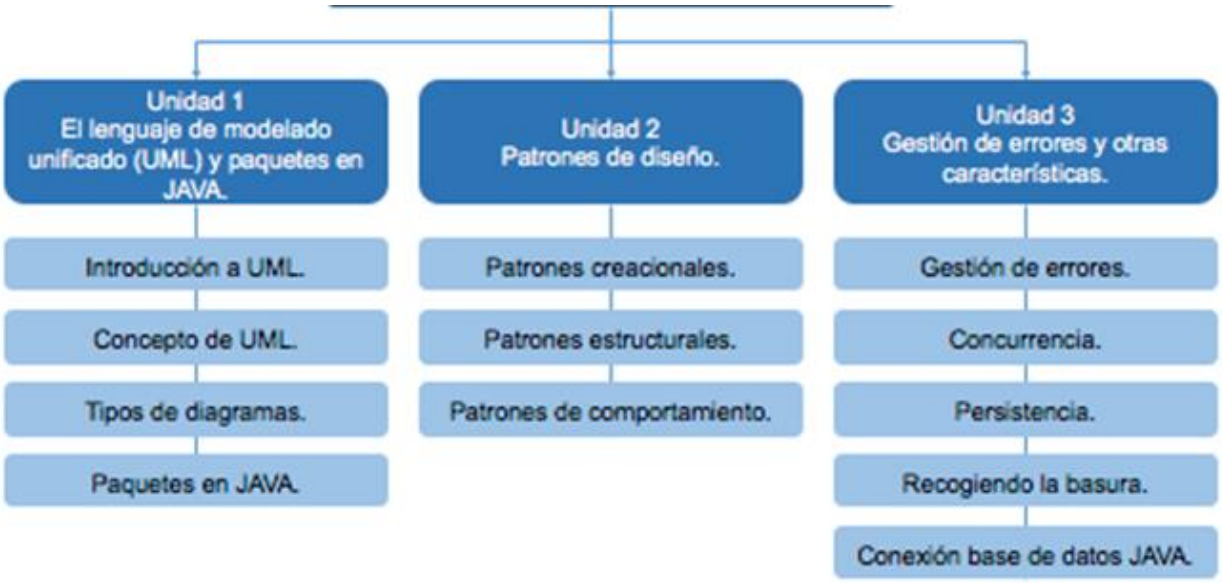
Objetivo general

Comprender los principales conceptos y aplicaciones de la Programación Orientada a Objetos, al identificar y aplicar sus ventajas en el desarrollo de software, a través de la solución de problemas algorítmicos por medio de la arquitectura empresarial JAVA.

Objetivos específicos

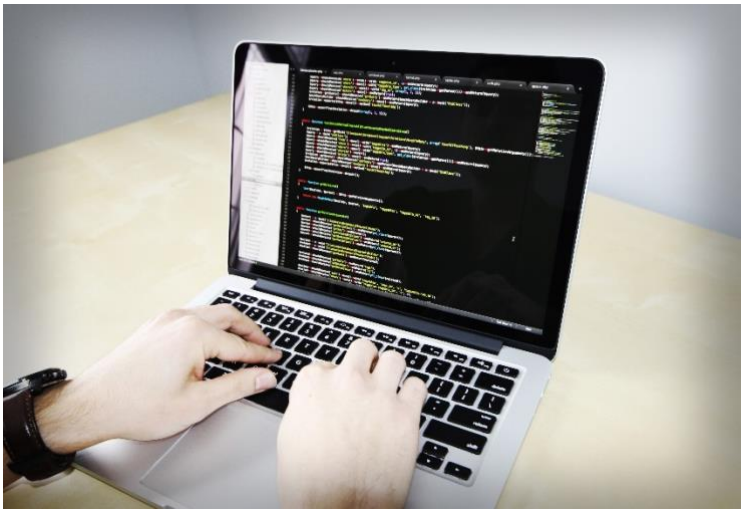
- Realizar el planteamiento y modelamiento por medio de expresiones de la Programación Orientada a Objetos, para la descripción de situaciones y optimización de procesos.
- Utilizar diferentes estructuras fundamentales en la Programación Orientada a Objetos.
- Identificar las principales propiedades y relacionarlas entre los lenguajes de programación.
- Desarrollar aplicaciones en lenguaje de alto nivel.

Mapa del curso



Al final del semestre, en caso que no apruebes el curso, tienes el derecho de solicitar el Proceso de Recuperación y la nota que obtengas será tu calificación final. Dada la situación que no hayas aprobado el Proceso de Recuperación o hayas decidido no presentarlo, deberás volver a cursar la materia.

Unidad 1. El lenguaje de modelado unificado (UML) y paquetes en JAVA



UML (Lenguaje de Modelado Unificado), es una herramienta en el área de desarrollo de sistemas, con la cual, los desarrolladores pueden crear diseños convencionales y de fácil manejo, que permitan una comunicación más clara para las demás personas. Cuando nació la programación, los análisis y los problemas para dar solución no eran tan detallados, y algunas de las herramientas que usaban se basaban en dibujar garabatos en hojas de papel, que permitieran identificar qué era lo que se deseaba desarrollar. Estos desarrollos se hacían de una vez, realizando ajustes conforme se iban necesitando.

Hoy en día, los desarrolladores requieren de un plan de trabajo que les permita realizar un análisis y diseño detallado de la solución, de tal manera que el usuario final puede identificar y conocer qué es lo que se va a desarrollar y emitir cambios, si así se requiere.

UML está conformado por diversos elementos gráficos, que al combinarse conforman un diagrama. Al tratarse de un lenguaje, se cuenta con normas y reglas que permiten combinar estos elementos.

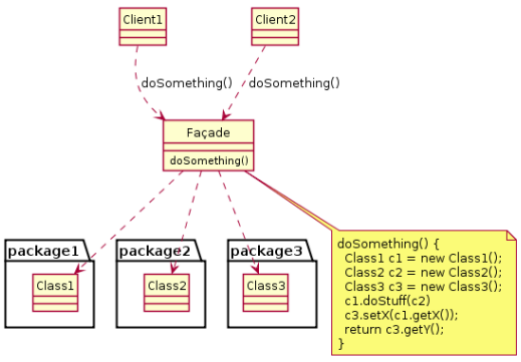
Con los diagramas se pretende representar diversas perspectivas de los sistemas. Esto se conoce como modelo, entendiéndose como modelo la representación simplificada de una realidad; con el modelo UML representamos supuestamente lo que hará el sistema, pero no nos dice cómo ha de ser implementado este sistema.

Dentro de los conceptos de UML tenemos: paquetes, notas, estereotipos y diagramas.Entre los diagramas más comunes y que se describirán con más detalle durante el transcurso de la Unidad, tenemos:

1. Diagramas de despliegue.
2. Diagramas de objetos.
3. Diagramas de paquetes.
4. Diagramas de comportamiento.
5. Diagramas de actividades.
6. Diagramas de comunicación.
7. Diagramas de estado.
8. Diagramas de casos de uso.
9. Diagramas de componentes.
10. Diagramas de secuencia.
11. Diagramas de clases.

Tema 1. Introducción al UML

Para iniciar, es importante que sepas qué es UML.



El lenguaje de modelado unificado (UML por sus siglas en inglés), es una herramienta en el área de desarrollo de sistemas, que permite a los desarrolladores elaborar diseños que capturan ideas de manera convencional y de fácil manejo, de tal manera que permite una comunicación clara y concisa con las demás personas.

NOTA: Debido a que la actividad del desarrollo del sistema es netamente humana, existe mucha posibilidad de cometer errores en cualquier etapa del desarrollo de

software, lo que implica que la generación de los programas se vuelva aún más difícil, y no se obtengan los resultados esperados.

En los inicios de la programación, no se realizaba un análisis detallado sobre los problemas a resolver, y lo más que se podía hacer era, en una hoja de papel, dibujar garabatos que permitieran identificar qué se iba a desarrollar; lo que principalmente se hacía era generar de una vez y, conforme se iban necesitando elementos, se realizaban los ajustes.

Actualmente, para desarrollar programas de software, es necesario realizar un plan de trabajo bien analizado y detallado, con el fin que el cliente identifique y conozca qué es lo que el desarrollador va a realizar y, de esta manera, pueda emitir cambios si las necesidades no están claras en este plan, o si existe algún cambio con respecto a lo planeado en sus inicios.

A medida que crece la complejidad de los problemas del mundo actual, los sistemas deben buscar adaptarse a estas dificultades; deben también crecer en complejidad de sus hardware y software que les permitan comunicarse a grandes distancias por medio de redes las cuales, a su vez, se comunican con bases de datos que contienen gran cantidad de información.

De lo dicho anteriormente, podemos decir que se desprende la siguiente pregunta:

¿Cómo adaptarse o manejar toda esta complejidad?
Para dar solución a este interrogante, la clave se encuentra en la organización de los procesos de diseño, de tal manera que todos los actores involucrados en el desarrollo de sistemas, comprendan e interactúen con él; de ahí se origina la necesidad de hacer uso de UML, que nos permite llevar a cabo una organización de todos estos procesos.

Cuando se va a construir un edificio de 21 pisos, un arquitecto no puede concebir esta compleja estructura sin antes crear un anteproyecto; así mismo, nosotros como desarrolladores no podemos dar inicio a un proceso sin antes tener un plan de diseño detallado del problema, el cual será el resultado de un análisis cuidadoso de todas las necesidades que tiene el cliente.

NOTA: Si se cuenta con un plan bien detallado, se tiene un tiempo estimado de desarrollo, lo que permite saber con detalle qué actividades se están ejecutando y cuáles presentan atrasos o adelantos.

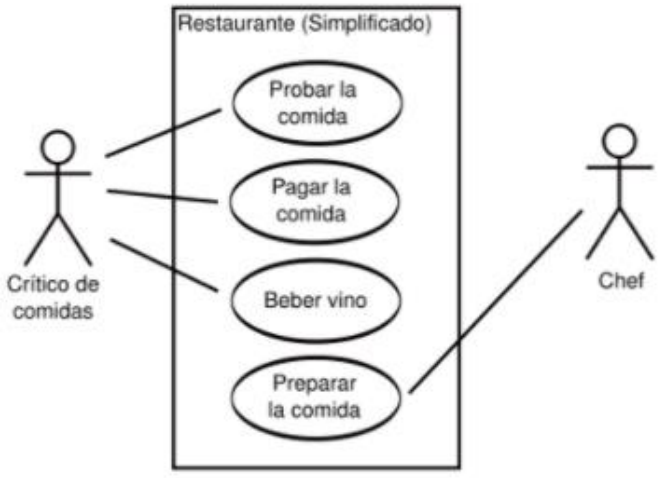
En la historia del UML podemos observar 3 fechas importante. Haz clic en cada una para conocerlas.

A. 1990: UML nace a principio de los años 90 cuando, tres amigos, Grady Booch, James Rumbaugh e Ivar Jacoboson, diseñaron su propia metodología para el análisis y diseño orientado a objeto, y luego decidieron unir esfuerzos y crear una única metodología.

B. 1994: En el año 1994, Rumbaugh ingresó a laborar a la empresa Rational Software Corporation, compañía donde ya Booch se encontraba trabajando como fundador y un año después, Jacobson se incorporó a la misma empresa.

C. 1997: Muchas empresas vieron en UML un propósito útil y fue así como crearon el Consorcio UML; entre algunos de sus miembros se encuentran Hewlett-Packard, Intellicorp, Microsoft, Oracle, y fue así como en el año 1997 crearon la primera versión de UML, el cual se convirtió en un Lenguaje de Modelado Estándar. Actualmente se cuenta con la versión 2.5.

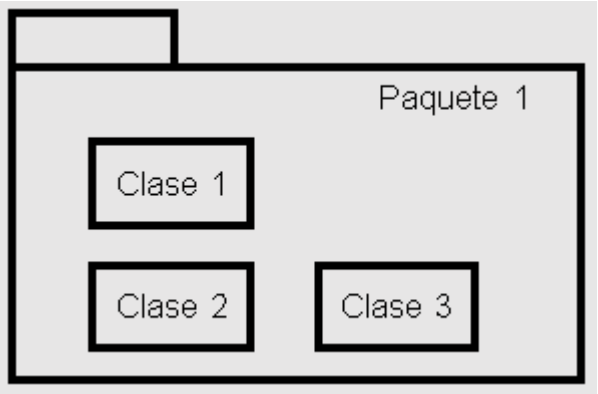
Tema 2. Concepto de UML



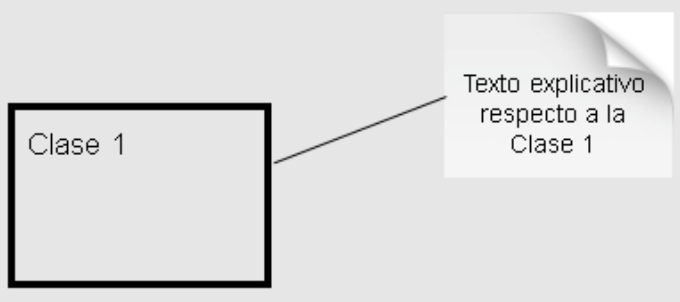
UML está conformado por distintos elementos gráficos que, al combinarse, conforman un diagrama y como UML es un lenguaje de modelado, se trabaja con normas y reglas que permitan combinar estos elementos.

Con los diagramas, se pretende presentar diferentes perspectivas de los sistemas y a esta representación se le conoce como modelo.

Paquetes: Los **paquetes** nos permitirán organizar los elementos de un diagrama en un grupo; cuando se requiere mostrar que una clase o un componente hacen parte de un subsistema cualquiera, se agrupan en paquetes, que son representados por una carpeta.



Las **notas** nos permiten representar una explicación clara de alguna parte del diagrama; estas son representadas gráficamente como un papel adhesivo en forma de rectángulo con una esquina doblada y, dentro de este, se escribe toda la nota explicativa la cual se conecta al elemento a través de una línea discontinua.



Los **estereotipos** nos permiten tomar elementos propios de UML y, a partir de estos, crear componentes que se adapten a las exigencias; esto se realiza para ajustar algún elemento a la medida del requerimiento a diseñar, y está representado con un nombre entre dos pares de paréntesis. Por lo general se crean estereotipos para modelar interfaces, debido a que estas no tienen atributos; es decir, representan un conjunto de atributos que se pueden usar una y otra vez en el modelo. (Tomar una interfaz de Visio). Los **estereotipos** nos permiten tomar elementos propios de UML y, a partir de estos, crear componentes que se adapten a las exigencias; esto se realiza para ajustar algún elemento a la medida del requerimiento a diseñar, y está representado con un nombre entre dos pares de paréntesis. Por lo general se crean estereotipos para modelar interfaces, debido a que estas no tienen atributos; es decir, representan un conjunto de atributos que se pueden usar una y otra vez en el modelo.

Los **diagramas** de UML permiten detallar un sistema a partir de diferentes puntos de vista; no en todos los modelos UML aparecen todos los diagramas, y la aplicación de cada modelo depende de las necesidades planteadas en el problema a diseñar.

Entre los diagramas más comunes y que se describirán con más detalle durante el transcurso de la Unidad tenemos:

- Diagramas de despliegue.
- Diagramas de objetos.
- Diagramas de paquetes.
- Diagramas de comportamiento.
- Diagramas de actividades.
- Diagramas de comunicación.
- Diagramas de estado.
- Diagramas de casos de uso.
- Diagramas de componentes.
- Diagramas de secuencia.
- Diagramas de clases.

Microsoft Visio es un software (dibujo vectorial) para dibujar una variedad de diagramas. Entre ellos se incluyen diagramas de flujo, organigramas, planos de construcción, planos de planta, diagramas de flujo de datos, diagramas de flujo de procesos, modelado de procesos de negocios, diagramas de carriles, mapas 3D y mucho más. Es un producto Microsoft que se vende como agregado de MS Office.

Mas información ver enlace: https://support.microsoft.com/es-es/office/tutorial-para-principiantes-de-visio-bc1605de-d9f3-4c3a-970c-19876386047c?ui=es-es&rs=es-es&ad=es

Tema 3. Tipos de diagramas

El Lenguaje Unificado de Modelado (UML) establece un conjunto de notaciones y diagramas estándar que nos permiten modelar sistemas orientados a objetos; de igual manera, describe una semántica del significado de los diagramas y los símbolos.

UML se utiliza para el modelado de diferentes tipos de sistemas, ya sean software o hardware y situaciones del mundo real.

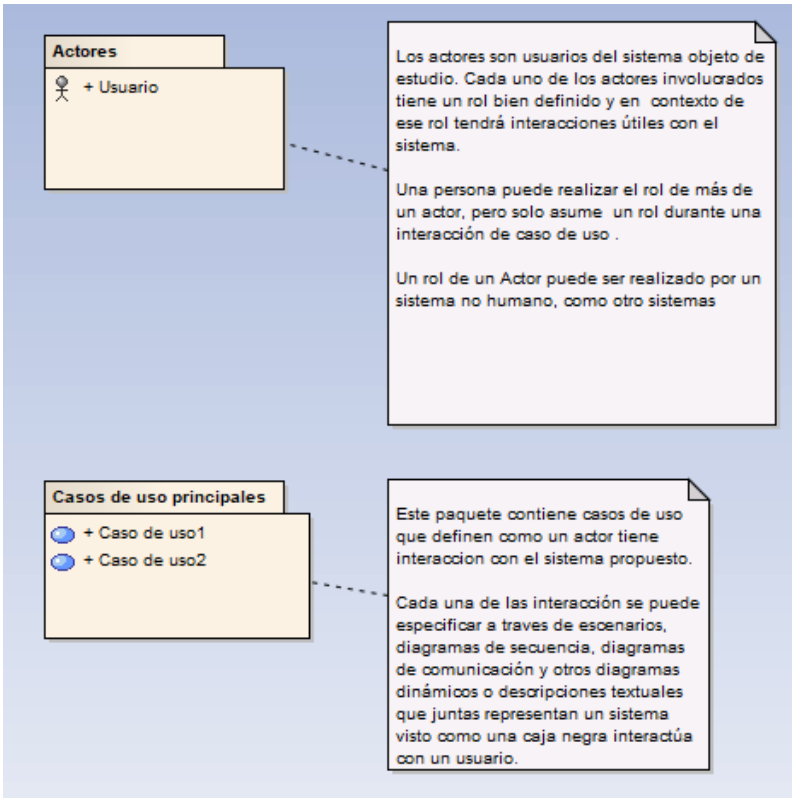
1. Los diagramas de casos de uso nos sirven para modelar los procesos de negocio.
2. Los diagramas de secuencia nos sirven para modelar la comunicación de mensajes entre objetos.
3. Con los diagramas de colaboración modelamos las interacciones entre objetos.
4. A través de los diagramas de estado se modela el comportamiento de los objetos en el sistema.
5. Los diagramas de actividad nos permiten modelar el comportamiento de los casos de uso, objetos u operaciones.
6. Los diagramas de clases nos ayudan a modelar la estructura estática de las clases en el sistema
7. Con los diagramas de objetos se modela la estructura estática de los objetos en el sistema.
8. Los diagramas de componentes nos permiten modelar los componentes.
9. Con los diagramas de implementación se modela la distribución del sistema.

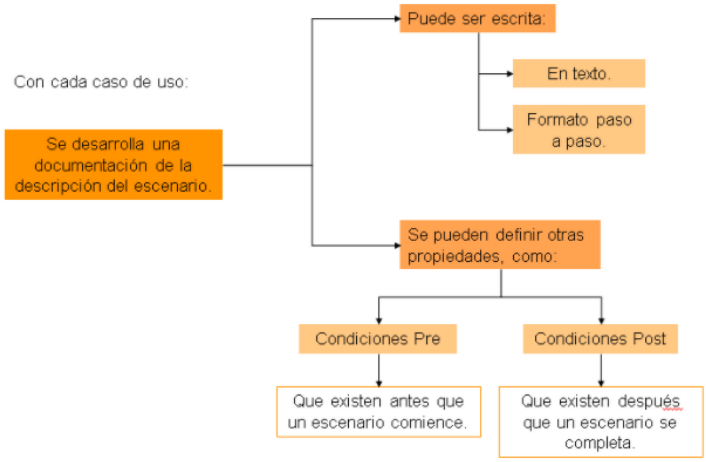
Modelado de casos de uso

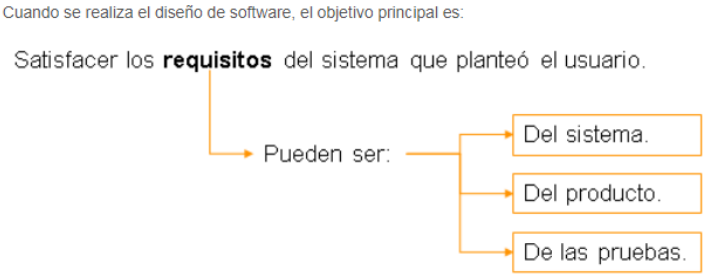
El modelado de casos de uso, es una de las técnicas más efectiva y a la vez más sencilla, para modelar requisitos del sistema desde una vista del usuario. Con este modelo se busca detallar cómo es el funcionamiento de un sistema o negocio actualmente, o cómo los usuarios desean que esto funcione. Es importante entender que no es una aproximación a la orientación de objetos, sino una forma de cómo se pueden modelar procesos, lo que nos permite realizar análisis de sistemas orientados a objetos.

Los **casos de uso** son el punto de partida para realizar análisis con UML. Está conformado por actores y casos de usos; los actores representan a un usuario o a otros sistemas que tienen interacciones con el sistema objeto de estudio, está representado por un “muñeco de palo”, y en este modelo los actores representan el tipo de usuario, no una instancia del usuario. Los casos de uso representan el

comportamiento que tiene el sistema, y los escenarios que conforman el sistema en función de una actividad realizada por el actor. Estos son representados como una elipse.







Para entender mejor el tema de casos de uso, imaginemos que compramos una computadora; existen un sinnúmero de cualidades y gamas, y es ahí donde nos hacemos varias preguntas.

Cuando vamos a realizar una compra o alguna actividad que tenemos planeada, siempre seguimos un procedimiento. Entonces, lo primero que hacemos es un **análisis de caso de uso**; este proceso lo llevamos a cabo a través de las preguntas sobre cómo utilizaremos el producto que vamos adquirir, de tal manera que este producto supla todas las necesidades que tenemos, y para ello es importante conocer todos estos requisitos. La forma como los usuarios hacen uso de los sistemas, nos da las pautas para diseñar y crear de una manera óptima.

- ¿Cuál computador voy a comprar?
- ¿Qué actividades quiero realizar con este computador?
- ¿Cuáles son las características que quiero para el computador?
- ¿Cuáles son las principales funciones que quiero que realice?
- ¿Lo utilizaré para diseño o para trabajos normales?

Modeleado de casos de uso

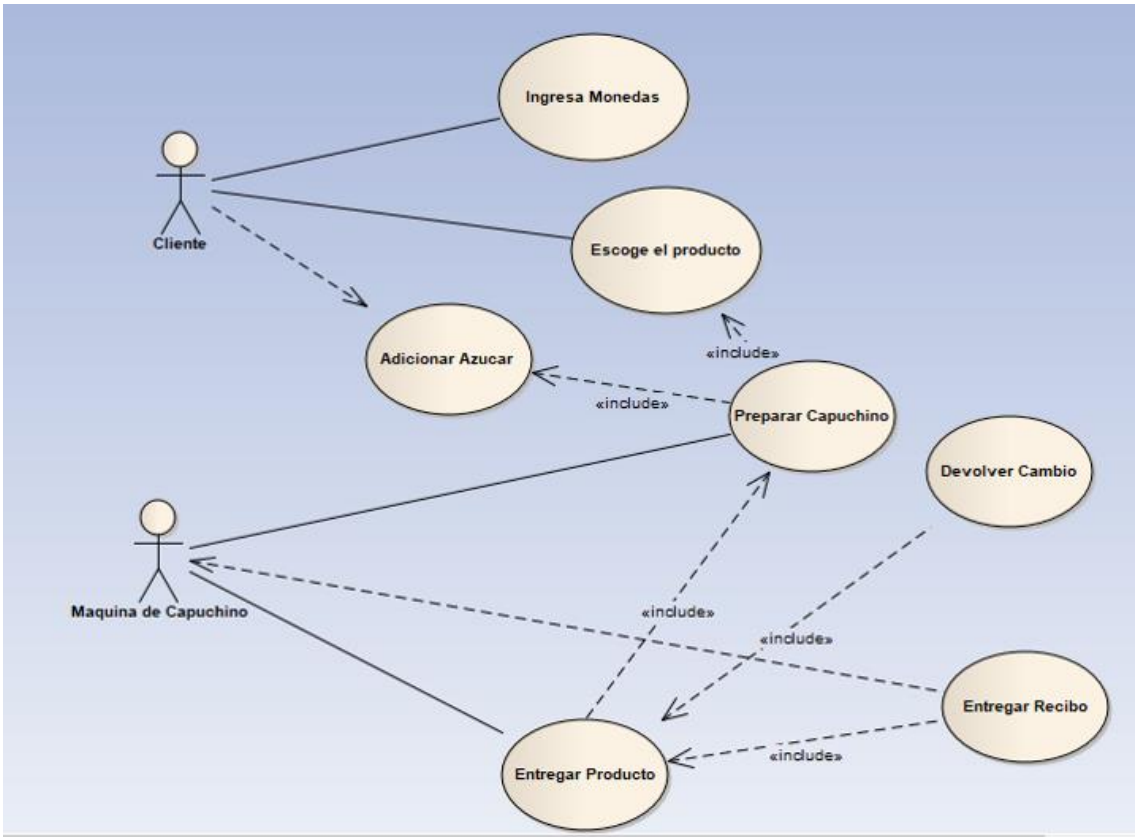
Podemos indicar entonces que los casos de uso son una colección de situaciones o acciones respecto a cómo utilizo un sistema. Cada uno de los escenarios que tenemos nos describe un evento a ejecutar o ejecutado, y estos eventos se inician por una persona, otro sistema, un hardware o con el paso del tiempo.

Una de las técnicas que más nos ayudan en la elaboración de este modelo, es la entrevista directa con los usuarios; cuando se tienen casos de usos derivados, es importante identificar estas condiciones para dar inicio al caso de uso.

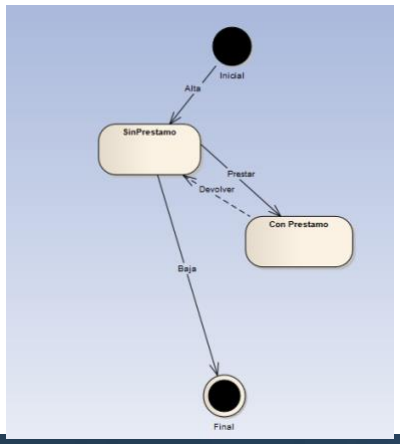
¿Sabías que?

Podemos reutilizar un caso de uso, y a este proceso lo llamamos inclusión, es decir, hacer uso de los pasos de un caso de uso como parte de la secuencia de pasos de otro caso de uso; otra forma es la extensión, es decir, se crea un nuevo caso de uso a partir de la adición de pasos a un caso de uso existente.

En conclusión, podemos decir que el caso de uso es una estructura que nos permite describir la forma como los sistemas se mostrarán a los usuarios finales. Es una agrupación de escenarios que son iniciados por una entidad que conocemos como actor; como resultado, los casos de uso deben entregar algo de valor ya sea para el actor o para otra persona o sistema.



Diagramas de estado



Un objeto siempre tendrá un estado particular y existen muchos ejemplos como: el nacimiento de una persona, el movimiento de un automóvil, los ciclos de lavado de una lavadora, entre otros. Con el diagrama de estado de UML, se pretende capturar todos los estados que tienen los objetos, y siempre se cuenta con un estado **inicial** y un estado **final**,

para caracterizar un cambio en el sistema; es decir, que sus objetos modificaron su **estado** como consecuencia de su suceso o el tiempo.

Un diagrama de estado se modela para todas las clases que tienen un comportamiento dinámico, en el que se modelan las secuencias del estado que un objeto de la clase tiene durante su vida, en respuesta al estímulo recibido, con sus propias respuestas y acciones.

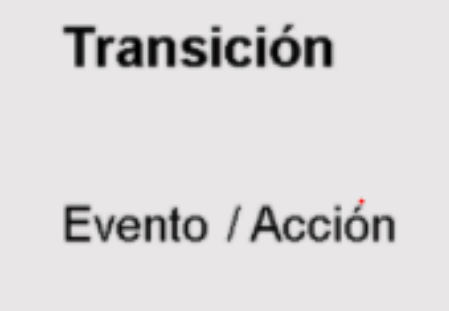
Estado

Con los estados representamos condiciones de objetos de acuerdo con su punto en el tiempo, o dicho de otro modo, situaciones durante la vida de un objeto. Se representa con un rectángulo con esquinas ovaladas.

Transición

Los eventos nos indican qué es lo que realizan los objetos cuando pasan de un estado a otro. Con la línea de transición, describimos los movimientos de un estado a otro. Estas líneas son nombradas con los eventos.

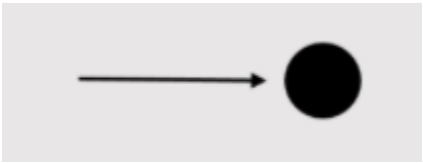
Se representa por medio de una flecha, que indica la transición entre diferentes estados; se etiqueta con el evento que provocó y con la acción que resulta.



Estado inicial

Representa el estado inicial de un objeto

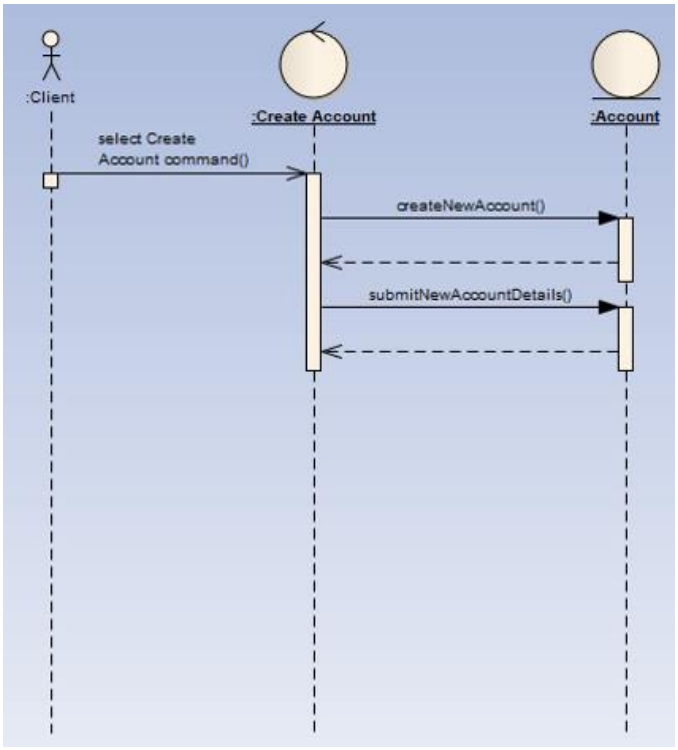
Estado final



Representa el estado final de un objeto.

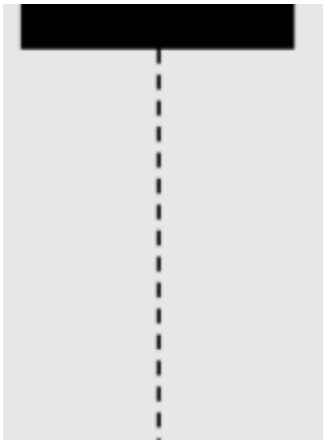
Diagramas de secuencia.

El diagrama de secuencia nos permite modelar las interacciones entre los objetos del sistema. Estos son modelados, por cada caso de uso, mientras que con los casos de uso, modelamos los escenarios de las vistas de negocios; el diagrama de secuencia contiene el detalle de implementación del escenario, teniendo en cuenta los objetos y las clases que se usarán en la implementación del desarrollo.



Cuando se examina la descripción de un caso de uso, se determina qué objetos son necesarios para la implementación del escenario. Este diagrama describe qué objetos son los que intervienen en el escenario, y se representan con líneas verticales discontinuas y los mensajes que son pasados entre los objetos se representan como vectores horizontales. Los mensajes deben dibujarse de manera cronológica desde la parte superior hasta la inferior del diagrama, porque la distribución horizontal de los objetos se realiza de manera arbitraria. En el análisis inicial, por lo general se coloca el nombre de un mensaje en la línea de mensaje, y luego en la etapa de diseño el nombre es reemplazado con el nombre del método que está llamando el objeto. El método invocado, hace parte de la definición de la clase instanciada por el objeto en la recepción final del mensaje.

Objetos

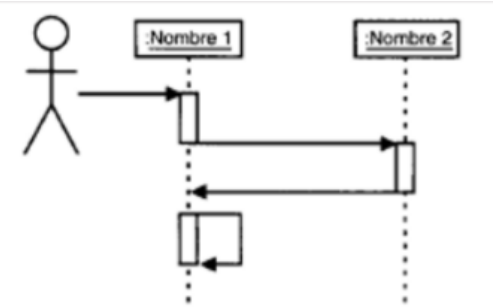


Estos son ubicados en la parte superior de izquierda a derecha y se colocan de manera que se simplifiquen en el diagrama. La extensión que está debajo de cada objeto se dibuja como una línea discontinua que es conocida como la línea de la vida del objeto; también encontramos un rectángulo pequeño que representa la activación, o ejecución de la operación que realiza el objeto. El tamaño del rectángulo representa la duración de la actividad.

Mensaje

El mensaje que va de un objeto a otro pasa de la línea de vida del objeto a la de otro. Un objeto puede enviar un mensaje a sí mismo. Estos mensajes pueden ser simples, sincrónicos o asincrónicos.

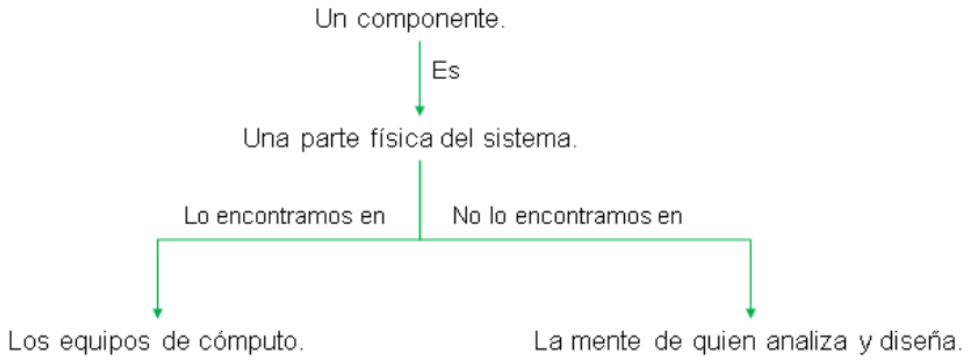
Tiempo



El tiempo es representado en forma de dirección vertical. Este se inicia en la parte superior y avanza hasta la parte inferior. Si el mensaje está más cerca de la parte superior, ocurrirá antes de uno que esté cerca de la parte inferior.

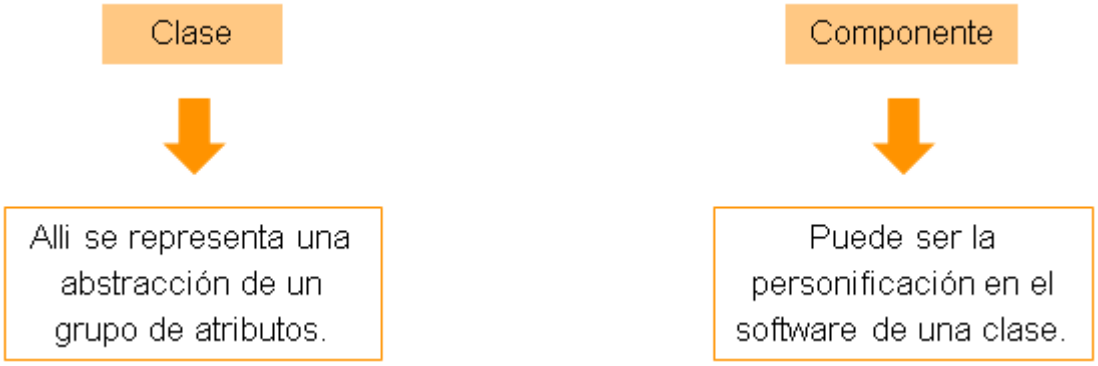
Diagramas de componentes

En software:



Entonces podemos decir que un componente es una tabla, un archivo de datos, un ejecutable, etc. Ahora, hagámonos la siguiente pregunta: ¿cuál es la relación que existe entre un componente y una clase?

Imaginar el componente como la personificación en el software de una clase.



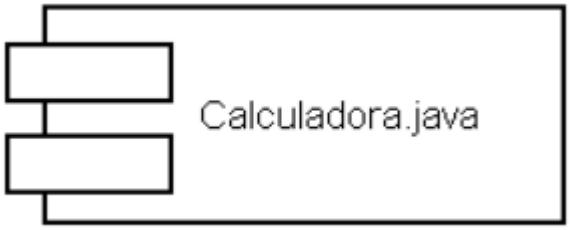
En la clase se representa una abstracción de un grupo de atributos y operaciones, y ahora un componente puede ser la implementación de más de una clase. Cuando se trata de un componente, entonces decimos que estamos tratando con una interfaz; una interfaz puede ser física o conceptual, y puede ser la misma que usa una clase o una implementación de software (componentes), es decir, tal cual como se modela una interfaz para una clase, asimismo se representa una interfaz para un componente.

NOTA: Tener en cuenta que UML distingue entre una clase y un componente, pero no hace distinción entre una interfaz conceptual y una física.

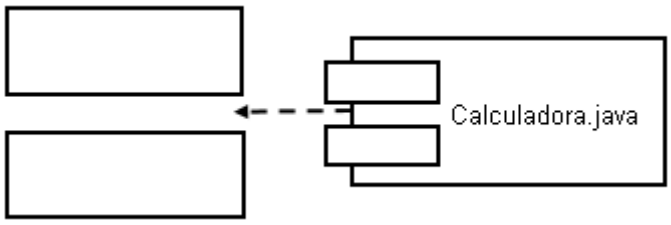
Símbolos del diagrama de componente:

El diagrama de componente incluye componentes, interfaz y relaciones, incluso también otros símbolos que se utilizan en otros diagramas. Haz clic sobre los títulos para conocerlos

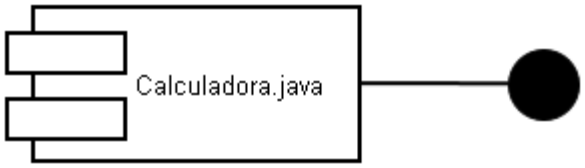
Componente: Está representado por un rectángulo, que tiene otros dos rectángulos sobrepuestos al lado izquierdo, y se coloca el nombre del componente dentro del símbolo.



Interfaz: Está dado por un rectángulo que tiene toda la información a relacionar y se conecta por medio de una línea discontinua y una flecha representada por un triángulo.

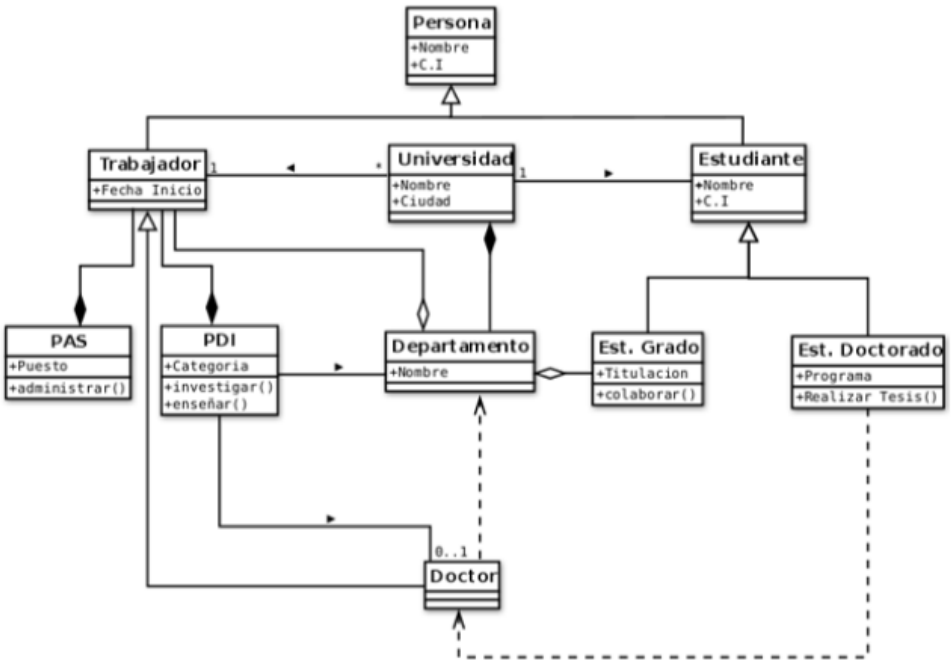


Aquí se representa como un componente y su interfaz, como un círculo pequeño que nos indica la interfaz y que es conectada a un componente por medio de una línea continua.



Diagramas de clase

El diagrama de clase es el diagrama principal en el diseño y análisis de sistemas; aquí se especifica la estructura de la clase, sus relaciones y toda la estructura de herencia. Cuando realizamos el análisis de sistemas, el diagrama es desarrollado buscando una solución ideal a la problemática presentada. En la etapa de diseño, se utiliza el mismo diagrama y es modificado para satisfacer los detalles de la implementación.



NOTA: Los casos de uso nos permiten un análisis orientado a objetos y nos ayudan con el desarrollo del diagrama de clase; los objetos que se identificaron en este diagrama se modelan en términos de la clase que es solicitada y las interacciones entre los diferentes objetos relacionados entre las clases gestionadas. En el diseño, el diagrama de clase se elabora con el fin de obtener un detalle de toda la implementación del sistema.

Los diagramas de intención y colaboración, modelan secuencias dinámicas de acciones ente objetos de un sistema; el diagrama de estado es utilizado para el modelado de comportamientos dinámicos de un objeto en particular, o de una clase de objetos; estos diagramas se modelan para todas las clases que se consideran con comportamientos dinámicos. Los estados representan las condiciones de un objeto en cierta parte del tiempo. Los eventos representan incidentes que hacen que los objetos pasen de un estado a otro. Las líneas de transición nos describen los movimientos desde un estado a otro, y estas son nombradas con el nombre del evento.

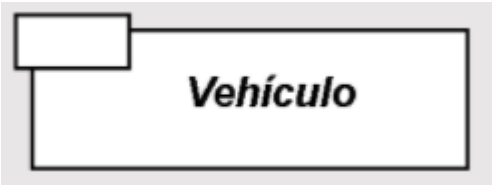
Explicación

Clase

Las clases son representadas por medio de un rectángulo, el nombre de la clase debe ir con la primera letra en mayúscula.

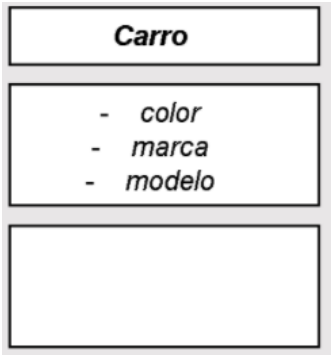
Paquetes

Un paquete es la manera en que UML organiza los elementos de un diagrama. Se representa por una carpeta tabular



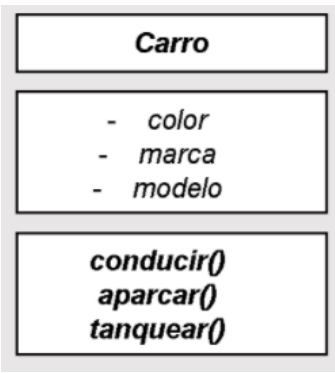
Atributos

Un atributo es una propiedad o característica de una clase y describe un rango de valores que la propiedad podrá obtener en los objetos de una clase. El nombre se escribe en minúscula.



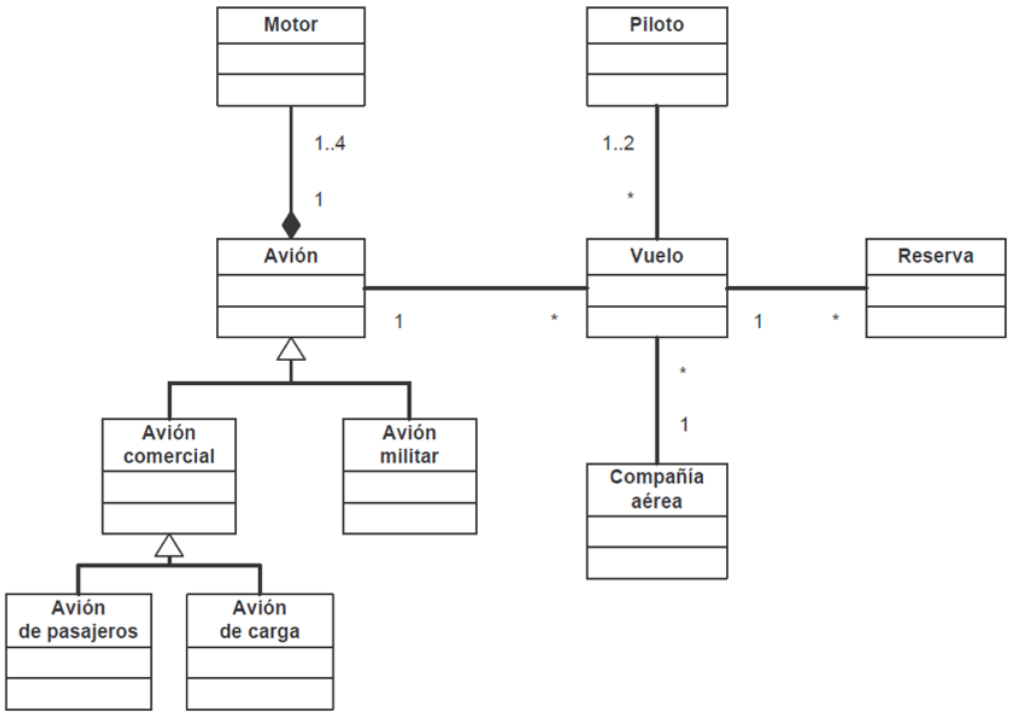
Operación

Una operación es una actividad que una clase puede realizar, o que se puede hacer en una clase; el nombre se escribe en minúscula y al final unos paréntesis.

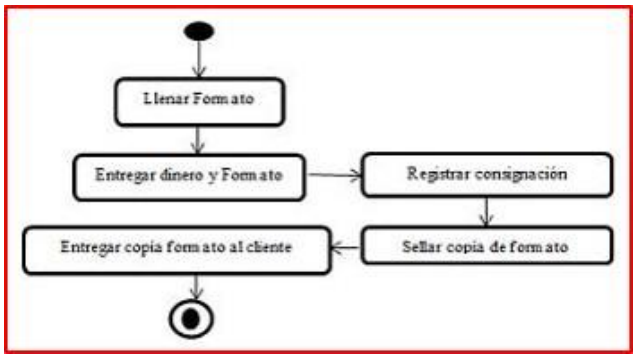


Modelo

Con este diagrama se puede modelar un conjunto de clase con sus respectivas relaciones. Observa el siguiente ejemplo.



Diagramas de actividad



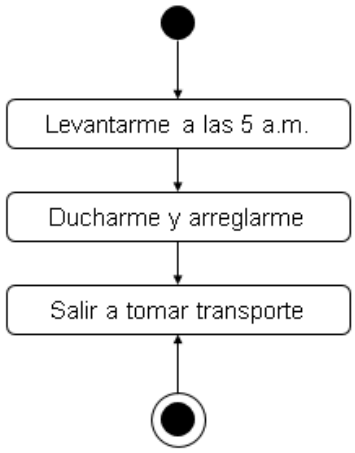
Este diagrama es un flujo de procesos multipropósito que se usa para modelar el comportamiento que tiene el sistema. Estos se pueden modelar para un caso de uso, una clase o un método complicado.

Es parecido a un diagrama de flujo, con la única diferencia que el diagrama de actividades puede mostrar procesos en paralelos. Ofrece una herramienta gráfica que permite modelar el proceso de caso de uso, una descripción textual, código, y otro diagrama de actividad que puede permitir detallar mejor la actividad.

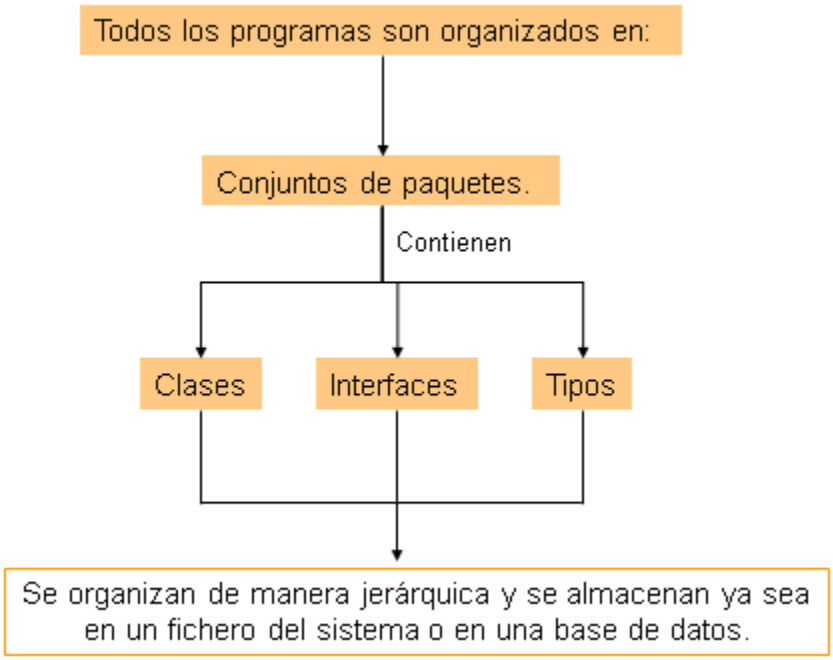
El diagrama de actividad, la mayoría de veces, se usa para representar los elementos en el desarrollo de los pasos dados por las acciones generadas internamente, y es así como se asignan actividades a la clase antes de culminar el diagrama de actividad.

La secuencia de actividades llegará a un punto donde se va a llevar a cabo alguna actividad. Algunas tomarán un camino y las otras tomarán otro, pero ambas son mutuamente exclusivas.

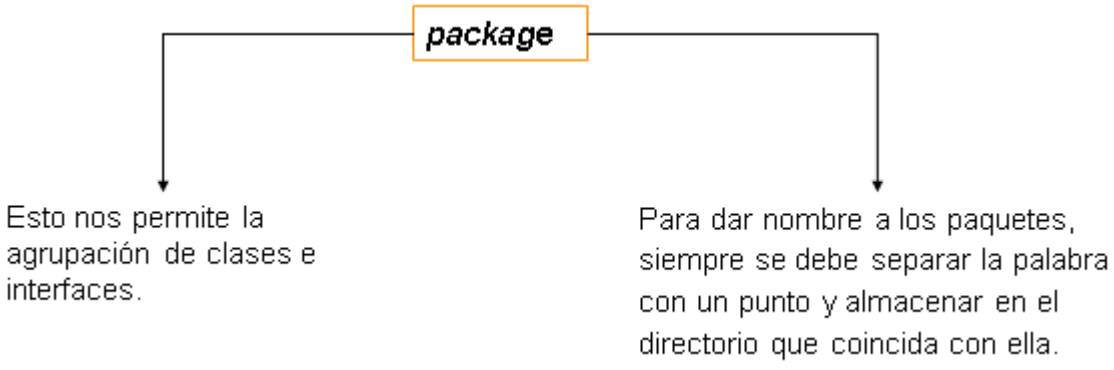
Una representación de este diagrama está dada por:



Tema 4. Paquetes en JAVA



La palabra clave para identificar un paquete en JAVA es:



Podemos observar una mayor utilidad del uso de paquetes, cuando estamos usando una gran cantidad de clases en un programa o cuando estamos usando clases desarrolladas por otro desarrollador en la que pueden coincidir algunos nombres de clases, ya que los paquetes son referencia para identificar el origen de la clase. Con los paquetes se puede controlar la visibilidad de las clases y sus respectivos métodos.

Por ejemplo:
Estos ficheros:
AppletContext.java
AudioClip.java

Tiene en su primera línea lo siguiente:
Package java.applet;
Si queremos ver el nombre de la clase, en los ficheros anteriores encontramos algo cómo:
nombre_de_la_clase.class

Todas las clases deben pertenecer a un paquete; si se omite la palabra package se considera que pertenece a un paquete por defecto, es decir, a un paquete que no tiene nombre, y los ficheros que se obtengan al compilar esta clase, se almacenan en este directorio. Se puede tener una jerarquía de paquetes, de tal manera que el paquete contenga subpaquetes que dependen de un paquete principal; de esta manera, podemos tener una mayor agrupación de nuestro código.

La finalidad de utilizar paquetes está dada, primero en la agrupación de clases, interfaces y tipos de una unidad de librería; de esta manera tenemos una mayor organización de las clases y por otro lado podemos emplear de manera homogénea, estas unidades de librería en cualquier aplicación que estemos desarrollando, de tal manera que podamos reutilizar código, y esto lo podemos realizar haciendo uso de la siguiente sentencia:

```
import java.util.*;
```

La función del import es cargar toda la biblioteca de utilidades, que para el ejemplo es la distribución estándar de JAVA. También podemos llamar solo una clase de esta biblioteca, lo que nos indica que no podemos usar las demás de la biblioteca.

```
import java.util.ArrayList;
```

Estructura de directorios asociados.

La palabra clave package, permite indicar que una clase hace parte de un paquete.

```
package mipaquete;

class miclasea

 {

….

}

classs mi clase

{

….

}
```

Se observa que se crea un paquete que se llama “mipaquete”, y todas aquellas clases que sean definidas en este código, hacen parte del paquete que se creó, es decir miclaseA y miclaseB.

Si tenemos el siguiente código:

```
package mipaqueteA;

package mipaqueteB;

class miclasea

{
```

….

}

classs mi clase

{

….

}

Generaría un error, toda vez que se debe usar la palabra package por fichero; ahora, si queremos que esta clase pertenezca a los dos paquetes, debemos declararla en ficheros independientes, con su correspondiente sentencia package-

Si queremos ejecutar cualquier clase por línea de comando, entonces debemos usar la siguiente sentencia.

java mipaquete.miclasea

Las siguientes son algunas bibliotecas de clases estándar de Java.

Java.lang

Es la clase central de JAVA, se manejan números, cadenas y objetos.

Cuando se utiliza esta sentencia, no es necesario hacer uso de *import* cuando se usan clases de este paquete.

Java.awt

Clase que nos permite la creación de interfaces gráficas, dibujar figuras e imagen.
Java.applet

Clase que se utiliza para crear Applets.

Java.io

Con esta clase se realizan operaciones de entrada y de salida.

Java.util

Tiene diferentes utilidades como: fecha, generación de números aleatorios, vectores dinámicos, etc.

Java.net

Con esta clase se pueden implementar aplicaciones distribuidas, es decir funciones en redes de ordenadores.

Java.sql

Clase para poder acceder a las bases de datos.

Javax.swing

Con esta clase se pueden crear interfaces gráficas de usuarios con componentes 100% Java.

Java.rmi

Clases para la creación de aplicaciones *Remote Method Invocation*.

Importar paquetes JAVA

Si queremos importar una clase de un paquete en JAVA, entonces debemos hacer uso de la palabra reservada import; podemos hacer esta importación ya sea de manera individual o completa.

Individual: import java.awt.Font;

Completa: import java.awt.*;

Ahora, supongamos que queremos crear un objeto fuente (Letra); para ello hacemos uso de la clase Font, y podemos usar dos alternativas:

1.import java.awt.Font;

```
Font fuente = new Font("Monospaced", Font.BOLD, 36);
```

2.No hacemos uso de la sentencia import

```
java.awt.Font fuente=new java.awt.Font("Monospaced", Font.BOLD, 36);
```

Por economía de código, es recomendable usar la primera alternativa, ya que no necesitamos crear varias fuentes de texto. Ahora, también podemos y se pueden usar las dos formas:

```
import java.awt.*;

public class miClase extends Panel implements java.io.Serializable{

//...

}
```

Si nos damos cuenta, Panel que pertenece a la clase del paquete java.awt, y Serializable, es una interfaz que está en el paquete java.io.

Compilar y ejecutar clases dentro de los paquetes.

Si tenemos dos paquetes que están dentro del mismo directorio y estos se llaman “miprimerprograma”, en este paquete tenemos una clase que se llama miPrimProg y el código de esta clase es:

```
package miPrimProg;

public class miPrimProg {



  public static void miPrimProg ()

  {

     System.out.println("Mi Primer Programa!");

  }



}
```

El paquete programa_prueba tiene la clase prueba; aquí tenemos un método estático que hace parte de la clase miPrimProg del paquete miprimerprograma, y es así como usaremos la sentencia de importación:

```
package programa_prueba;

import miprimerprograma.*;

public class  prueba{

  public static void main(String[] args)
```

```
{
  miprimerprograma. miPrimProg ();
 }
}
```

IU Digital de Antioquia
INSTITUCIÓN UNIVERSITARIA DIGITAL DE ANTIOQUIA

ORD No 74 de 2017 · VIGILADA MinEducación

www.iudigital.edu.co