1. **Programación Orientada a Objetos: Elaboración de diagramas de clases.**
   1. **Introducción**

La **Programación Orientada a Objetos** (**POO**) es una herramienta que permite reducir la complejidad de los sistemas de software. Usando la POO, un sistema que ocupa miles o millones de líneas de código puede organizarse como una colección de unidades (clases) mucho más pequeñas, cada una con cierta independencia y responsabilidades.

Esto permite que el mantenimiento y la ampliación o reutilización del software sea mucho más sencilla, de hecho, gracias a las herramientas que proporciona la POO, como la herencia o el polimorfismo, es mucho más fácil aprovechar el código existente, ampliándolo o modificándolo para nuevas o diferentes funcionalidades.

En el diseño orientado a objetos, un sistema se entiende como un conjunto de objetos que tienen ciertas propiedades y comportamientos. Dichos objetos son definidos o agrupados mediante las clases. Siendo así, podemos definir:

Se entiende por **objeto** a una entidad caracterizada por:

Una identidad que lo distingue de los demás.

Un estado que comprende todas sus características o atributos con valores que pueden ser constantes o variar a lo largo del tiempo.

Un comportamiento, definido por una serie de métodos que definen la evolución del objeto en función de la interacción con otros objetos o consigo mismo así como la posible variación de sus atributos.

Se entiende por **clase** a la descripción de un conjunto de objetos con propiedades y comportamientos similares, así como con relaciones comunes con otros objetos. Otra posible forma, más pragmática, de entender una **clase** es como una plantilla para la creación de objetos; Plantilla que describe las características y el comportamiento de un conjunto de objetos similares.

Según la definición de clase anterior, se puede redefinir **objeto** o **instancia** de una clase como cada una de las representaciones concretas y específicas de dicha clase.

Se habla de **paquete** de clases a una agrupación de clases que mantienen una relación estrecha entre ellas, y al igual que pueden existir relaciones entre las clases de un paquete, un paquete puede a su vez relacionarse con otros paquetes (en Java se puede declarar la pertenencia a un determinado paquete con la orden package que deberá ocupar la primera línea de código, antes de las posibles líneas de import y de la declaración de la clase).

* 1. **Conceptos OO**

Los **atributos** de un objeto son las características individuales que pueden diferenciar un objeto de otro, determinando su apariencia, estado y otras cualidades que lo definan. Los atributos se almacenan en las denominadas **variables de instancia**. Las variables de instancia son declaradas en la clase, pero sus valores son fijados y cambiados en el objeto. Por último, hay otro tipo de variables, denominadas **variables de clase**, las cuales se aplican a la clase y a todas sus instancias.

El comportamiento de los objetos de una clase se implementa mediante las llamadas **funciones miembro** o **métodos**. Un método es un conjunto de instrucciones que realizan una determinada tarea (hay cierta similitud entre los métodos de una clase y las funciones de un lenguaje de programación estructurado). Al igual que con los atributos, podemos hablar de **métodos de instancia** y **métodos de clase**.

En la POO se puede hablar de varios principios fundamentales:

* Abstracción, denota las características esenciales de un objeto en las cuales se describe sus comportamientos. La abstracción es clave en el proceso de análisis y diseño de la POO.
* Encapsulación, es el proceso por el cual se ocultan todos los detalles de un objeto que no contribuyen a sus características esenciales. Es decir, se separan los aspectos externos accesibles por otros objetos de los aspectos internos e inaccesibles. Los atributos y métodos internos del objeto, se denominan atributos y métodos privados.
* Modularidad, es la propiedad por la cual un sistema o aplicación se descompone en un conjunto de módulos o partes más pequeñas, coherentes e independientes.
* Herencia, es la propiedad que permite extender una clase, de forma que se puedan definir nuevas clases que heredan todo el comportamiento y atributos de la clase extendida, pero añadiendo nuevos comportamientos y atributos o modificando los de la clase extendida.
* Polimorfismo, consiste en reunir con el mismo nombre comportamientos diferentes, es decir un mismo mensaje (o denominación de método) puede dar resultados diferentes al ser recibido por objetos diferentes de clases distintas.

En menor grado también se puede hablar de:

* Tipificación, es la definición precisa de un objeto, objetos de diferentes tipos no pueden ser intercambiados, o solamente de forma muy restringida.
* Concurrencia, propiedad que permite distinguir entre los objetos activos y los que no lo están, será importante en los casos de programación concurrente o multihilo.
* Persistencia, por esta propiedad, un objeto puede trascender en el tiempo (un objeto puede seguir existiendo, incluso después de que su creador ya no lo haga).
  1. **UML**

El Lenguaje de Modelado Unificado (UML) es un lenguaje gráfico para visualizar, especificar y documentar cada una de las partes que comprenden el desarrollo de software. En particular el UML se ha convertido, de facto, en el lenguaje de modelado usado para la POO.

UML ha ido evolucionando a lo largo del tiempo, siendo la versión 2.5 la última aprobada en Mayo de 2015.

En UML se definen diferentes tipos de diagramas, en concreto en la versión 2.0 se habla de 13 tipos de diagramas según las siguiente estructura.

* Estática: Diagramas de Estructura
  + Diagrama de Clases
  + Diagrama de Estructuras Compuestas
  + Diagrama de Componentes
  + Diagrama de Despliegue
  + Diagrama de Objetos
  + Diagrama de Paquetes
* Dinámica: Diagramas de Comportamiento
  + Diagramas de interacción
    - Diagramas de Secuencia
    - Diagramas Resumen de Interacción
    - Diagramas de Comunicación
    - Diagramas de Tiempo
  + Diagramas de Actividad
  + Diagramas de Casos de Uso
  + Diagramas de Maquinas de Estado.

De dichos tipos de diagramas, los más utilizados son:[[1]](#endnote-1)

Diagrama de Clases, es el tipo de diagrama que más nos interesa, permite representar las diferentes clases que componen un sistema y las relaciones entre las mismas.

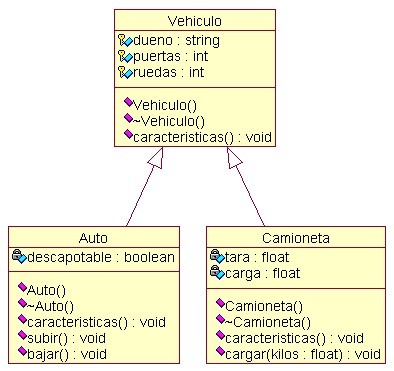


Diagrama de Objetos, este tipo de diagramas, permite representar los objetos (o instancias de clase) con sus correspondientes relaciones en un determinado momento de la evolución del sistema.

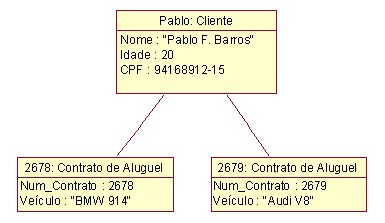
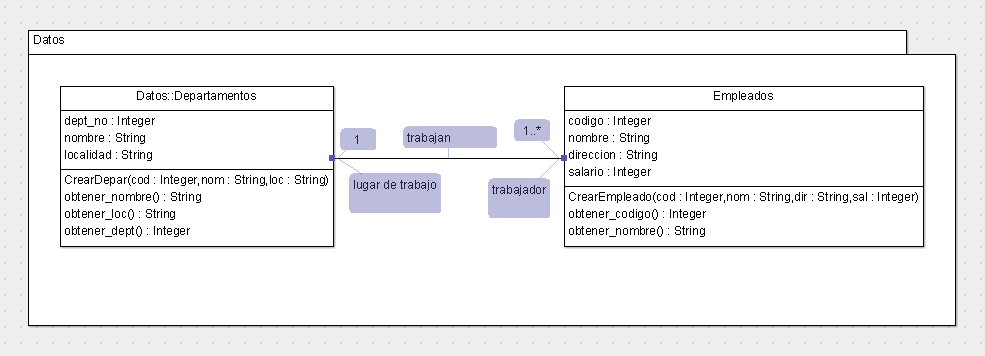
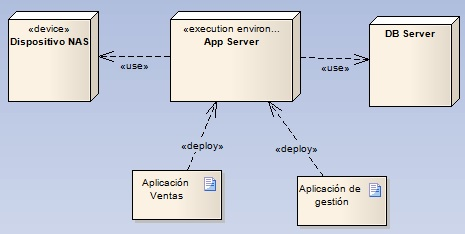
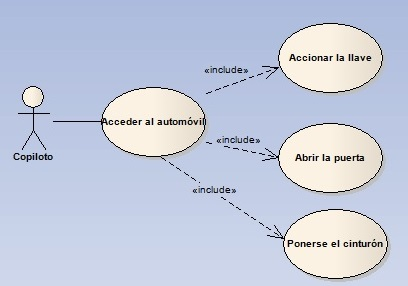


Diagrama de Paquetes, sirven para representar la organización de los paquetes y sus elementos (las distintas clases que los componen, así como las relaciones entre las mismas).

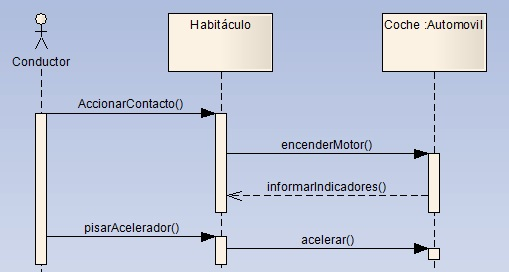
Diagrama de Despliegue, detalla el hardware físico sobre el que se ejecutará el sistema de software y también detalla como se despliega dicho software sobre ese hardware.



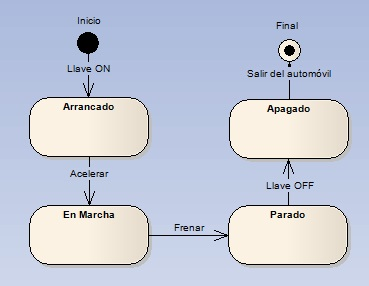
Diagramas de Casos de Uso, se trata de un diagrama de comportamiento y sirve para entender el uso del sistema, representando los actores (como los usuarios de un sistema) y las acciones (o casos de uso) del sistema.



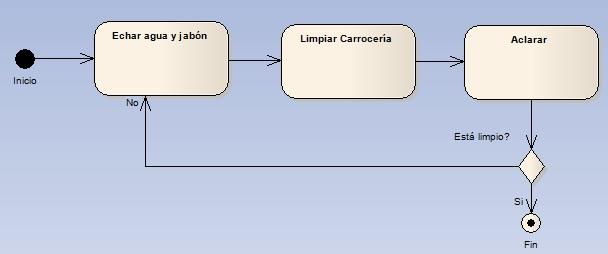
Diagramas de Secuencia, se podría ver como un particular de los diagramas de uso, un tipo de diagrama de interacción en el que se enfatiza la interacción entre los distintos actores y objetos.



Diagramas de Estado, son diagramas utilizados para mostrar los cambios de estado de los objetos.



Diagramas de Actividad, muestra la secuencia de actividades. El flujo de trabajo desde un punto de inicio al punto final detallando las decisiones que surgen en la progresión de los eventos contenidos en la actividad.



* 1. **Diagramas de Clases**

Como ya hemos inferido, un diagrama de clases es un tipo de diagrama de estructuras que describe la estructura de un sistema orientado a objetos mostrando sus clases así como las asociaciones entre ellas.

* + 1. **Clases**

Las clases se constituyen como la unidad básica que encapsula toda la información de un objeto o instancia de la clase. A través de la clase, podemos modelar el entorno en estudio.

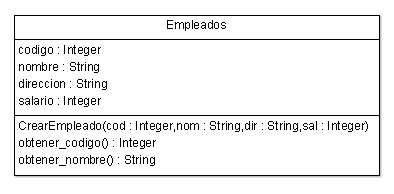
Las clases pueden ser de distinto tipo según ciertos modificadores de acceso:

* Package (o por defecto, sin modificador), se trata de una clase que sólo es accesible desde las clases definidas en el mismo paquete.
* Public, se trata de una clase que puede ser accedida desde cualquier otra clase, (en Java, en el caso de no pertenecer al mismo paquete, para poder utilizar una clase publica, es necesario utilizar la sentencia import)
* Final, este modificador indica que no puede haber subclases de esta clase, prohíbe la herencia de esta clase.
* Abstract, indica que la clase es abstracta, una clase abstracta no puede tener instancias, no se pueden crear objetos de una clase abstracta, pero una clase abstracta sirve como guión para la creación de nuevas clases que si pueden crear objetos o instancias.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **VISIBILIDAD DEL CAMPO O MÉTODO DE LA CLASE** | **Public** | **Protected** | **Package** | **Private** |
| Desde la misma clase. | Sí | Sí | Sí | Sí |
| Desde cualquier clase en el mismo paquete. | Sí | Sí | Sí | No |
| Desde una clase heredada y desde el mismo paquete. | Sí | Sí | Sí | No |
| Desde una clase heredada, pero de otro paquete. | Sí | Sí | No | No |
| Desde cualquier clase no heredada y de otro paquete. | Sí | No | No | No |

En el estándar UML, una clase será representada por un rectángulo con tres divisiones verticales.

* En la parte superior se sitúa el nombre de la clase.
* En la parte intermedia se sitúan los atributos o variables de instancia que caracterizan la clase.
* En la parte inferior se sitúan los métodos u operaciones, que representan la forma en la que interacciona la clase con su entorno.



**Atributos:**

Un atributo representa una propiedad de la clase que se encuentra en todas las instancias de la misma. Un atributo en UML se puede representar mostrando su nombre, su nombre y tipo o incluso mostrando también su valor por defecto. En la imagen de ejemplo, los atributos de la clase Empleados, son código, nombre, dirección y salario.

Un atributo se define por:

* Su nombre.
* Su tipo.
* Su valor por defecto (esto no es necesario si el atributo se debe definir a través de un método).

Al crear un atributo, es necesario indicar su visibilidad, la visibilidad está relacionada con el encapsulamiento.

**Métodos:**

Un método u operación representa la implementación de un servicio que muestra un comportamiento común a todas las instancias de la clase.

En toda clase existe un **método constructor** que se invoca para crear un objeto de la misma (en Java si no se indica explícitamente el método constructor, el compilador genera un constructor por defecto que no tienen ningún parámetro de entrada). El método constructor se llama igual que la clase. Una clase puede tener varios métodos constructores, se distingue de uno a otro según el número y tipo de los parámetros que espera el método.

Cuando en la definición de una clase hay más de un método con el mismo nombre, se dice que hay **sobrecarga de métodos** (que por tanto no es una característica exclusiva del método constructor), en una clase puede haber varios métodos con el mismo nombre, pero diferente firma.

La **firma de un método** son las características que lo definen (en Java la firma de un método está formada por su nombre, los argumentos que recibe y el tipo de dichos argumentos, sin embargo, el tipo de información que devuelve no forma parte de su firma).

Un método se define mediante:

* Su nombre, único en la clase, pero que puede ser compartido entre diferentes clases, esto da lugar al polimorfismo.
* Los parámetros de entrada, si los tuviese, que indican los parámetros de entrada que son precisos proporcionar al método cuando se invoca.
* El tipo de valor u objeto que devuelve, en el caso de devolver algún tipo de información (en Java esto significa que el método debe incluir la sentencia return en el cuerpo del método).
* El cuerpo del método, constituido por las diferentes sentencias que codifican la operación accesible mediante el método.

Al igual que con los atributos, al crear un método se hace preciso indicar su visibilidad.

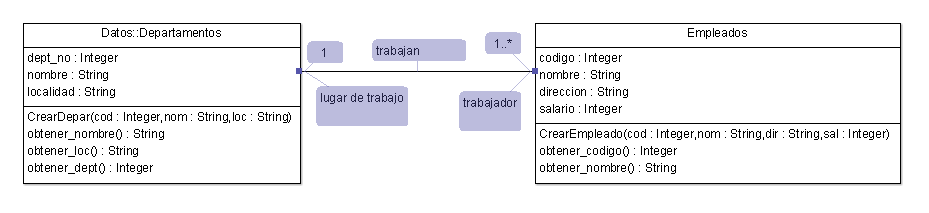
**Visibilidad de atributos y métodos:**

* **Public**: un atributo o método público que es visible tanto dentro de la clase como desde fuera de la misma (por ejemplo, desde otra clase). Se puede representar añadiendo el signo **+**
* **Private**: un atributo o método privado que sólo es visible dentro de la clase. Se puede representar añadiendo el signo **–**
* **Protected**: un atributo o método protegido, respecto a que sólo será visible desde dentro de la clase y además desde cualquier subclase que se derive. Se puede representar añadiendo el signo **#**
* **Pakage**: un atributo o método empaquetado, respecto a que sólo será visible desde dentro de la clase y desde cualquier otra clase que pertenezca al mismo paquete que la clase contenedora. Se puede representar añadiendo el signo **~**

Además de lo anterior, un método o atributo puede ser declarado como static. Cuando un atributo o método se declara como static, dicho atributo o método existe antes de crear ninguna instancia de la clase y puede ser invocado sin la necesidad de crear una instancia de dicha clase.

Hay otros modificadores definidos y que de momento no vamos a ver. Una muestra de algunos de ellos: *transient*, *volatile*, *native* y *synchronized*.

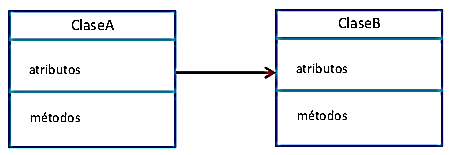
* + 1. **Relaciones**

Los vínculos que pueden existir entre las distintas clases, pueden ser representados mediante UML, en una relación se puede indicar la multiplicidad de la relación que da cuenta del número de instancias de una clase que se pueden, mediante dicha relación, vincular con un determinado número de instancias de otra clase.

En el diagrama UML anterior, la relación se expresa con una línea que une las dos clases, y la multiplicidad se muestra con números a los extremos de la línea, en concreto se indica en un extremo 1 y en el otro 1..\* con lo que se quiere reflejar que una instancia de la clase Departamentos se vincula con un número entre 1 y n de instancias de la clase Empleados (un Departamento puede estar constituido por uno o más empleados).

Se distinguen distintos tipos de relaciones:

* Asociación, cuando uno de los atributos de una clase es un objeto de otra clase se establece una asociación, las asociaciones pueden ser unidireccionales o bidireccionales. La direccionalidad de una asociación puede representarse mediante una punta de flecha en la relación.



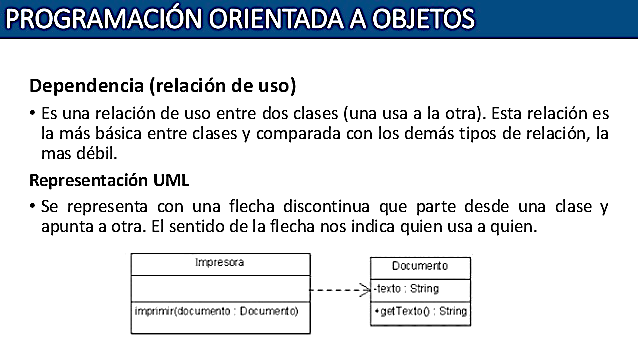
En particular, en una asociación unidireccional, la clase destino no sabe de la existencia de la clase origen.

Hay un tipo de asociación particular que es la reflexiva en la cual se unen entre si instancias de una misma clase.

Ejemplo, en el diagrama anterior, se podría crear una relación entre instancias de empleados, de forma que un empleado sea jefe de otros.

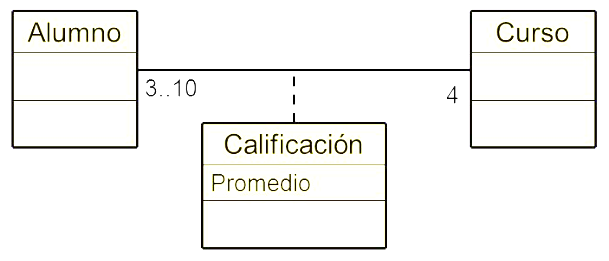
* Dependencia, cuando una clase necesita usar a otra para definir alguno de sus métodos, bien como parámetro de entrada del méroto o bien en el desarrollo del método.

La dependencia se representa con una línea discontinua.



* Clase Asociación, una asociación entre dos clases puede llevar una información necesaria para dicha asociación.

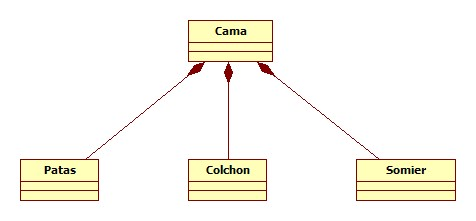
Este tipo de clase que surge de una asociación se representa mediante una línea discontinua con la relación de asociación correspondiente.



Ejemplo, supongamos un programa de asignación de notas, con dos clases para alumnos y para curso, además una relación de calificaciones entre ambas clases que añade la información del promedio de calificaciones de las asignaturas del curso.

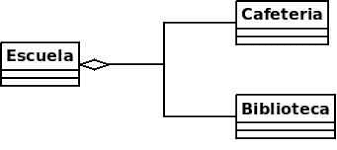
* **Composición**, puede darse el caso de un objeto compuesto de otros objetos, si los objetos componentes no tienen sentido fuera del objeto compuesto y la supresión de dicho objeto compuesto supone la supresión de sus componentes se habla de asociación de composición.

La relación de composición se representa mediante puntas de flecha con rombos rellenos.



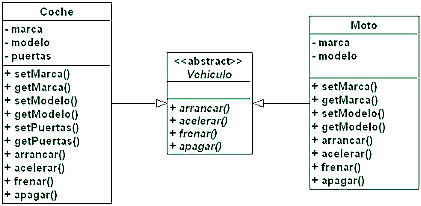
* **Agregación** es una relación muy similar a la de composición, salvo por el hecho que los objetos componentes tienen sentido fuera del objeto compuesto y la supresión de dicho objeto compuesto no conlleva la supresión de sus componentes.

La relación de agregación se representa mediante puntas de flecha con rombos huecos.



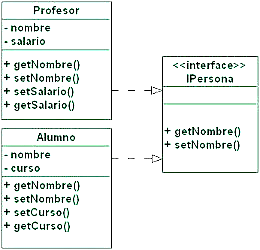
* Herencia (Generalización y Especialización, Superclase y Subclases), una de las características más importantes de la POO es la herencia, la posibilidad de compartir similitudes entre clases. Una herencia establece una relación entre una clase generalizada y una o más clases refinadas o especializadas de ella.

La clase más generalizada se denomina superclase y las especializaciones subclases, esta relación se representa con puntas de flecha en forma de triángulos huecos.



* Realización, se trata de una relación de herencia en la cual la superclase es una clase abstracta (o interfaz) y las subclases son las clases que implementan dicha clase abstracta.

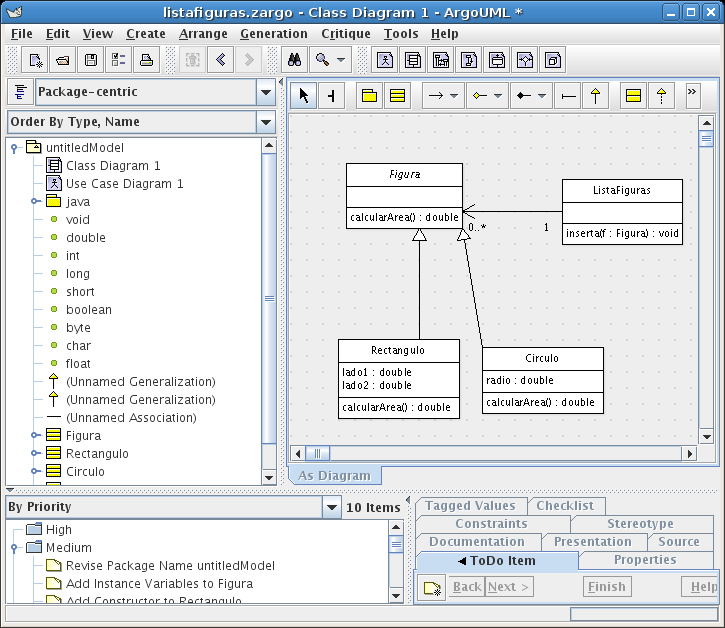
La relación de realización se representa mediante una línea discontinua y flechas en forma de triángulos huecos.

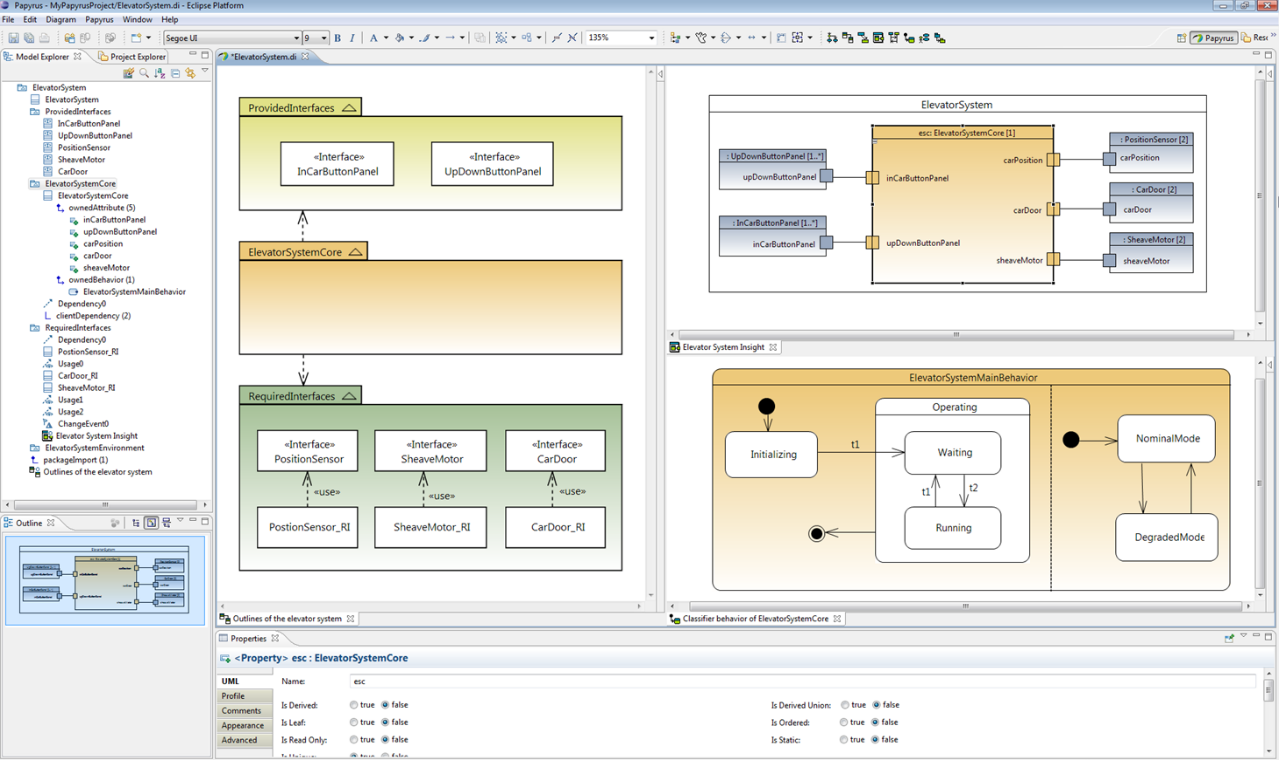


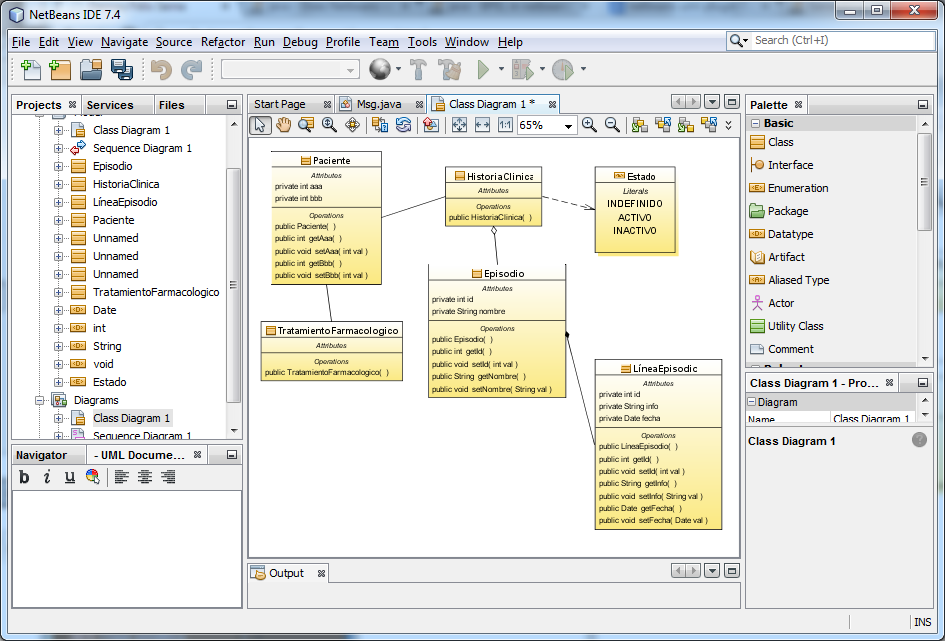
* + 1. **Estereotipos**

No los he comentado, no entran en este examen, a excepción de interface.

* 1. **Herramientas para diseño de Diagramas:**
* ArgoUML, se trata de un software con licencia Open Source, lamentablemente el proyecto no muestra progresos desde el 2011 y muestra soporte completo hasta UML 1.4 (soport los 9 tipos de diagramas descritos en esa versión: clases, despliegue, objetos, componentes, estados, actividad, casos de uso, colaboración y secuencia).

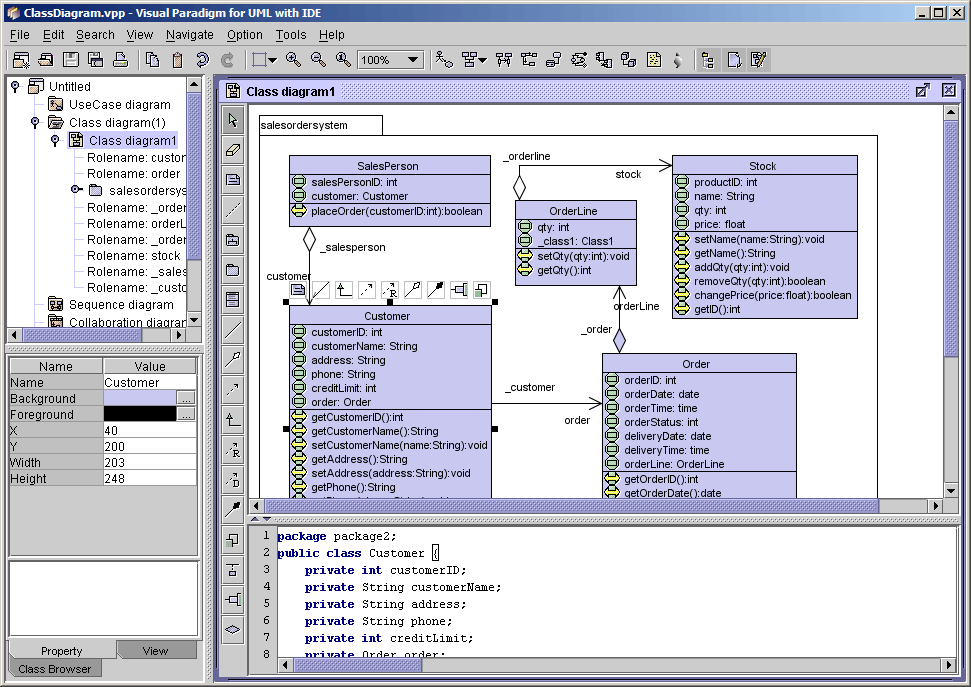


* Eclipse (UML Designer plugin), software que ya hemos comentado en el tema de los entornos de desarrollo. Eclipse a través del plugin UML Designer permite realizar diagramas UML, en concreto soporta los 10 diagramas más usados de la especificación UML 2.5.
* NetBeans (PlantUML plugin), este IDE del cual también hemos hablado, permite, como en el caso de Eclipse, realizar diagramas UML mediante el uso de plugins.



* Visual Paradigm, se trata de un software comercial que asiste en los procesos de desarrollo (empresariales, de software, etc) y que, entre otros, soporta UML con soporte para todos los diagramas especificados en la última versión del estándar.

Al tratarse de un software comercial, las características son mas completas y su uso más intuitivo, por el contrario, como suele suceder con este tipo de programas empresariales tan especializados, tiene un precio elevado.



Los anteriores no son los únicos, la lista de programas que permiten realizar diagramas UML, es realmente extensa: WhiteStarUML, JDeveloper, Microsoft Visual Studio, etc.

1. Diagramas originales del blog <http://www.notodocodigo.com> y de <http://clasesyobjetos.shoutwiki.com> [↑](#endnote-ref-1)