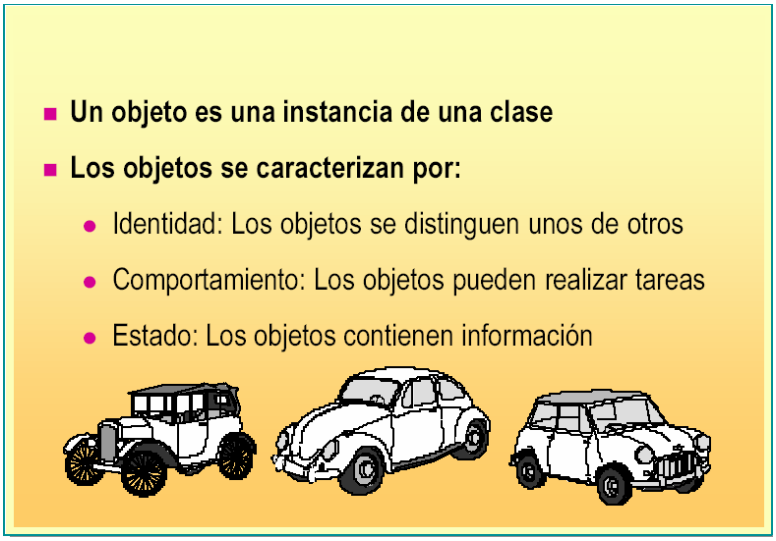
U.T. 4: DESENVOLVEMENTO DE SOFTWARE ORIENTADO A OBXECTOS

1. FUNDAMENTOS DE MODELADO ORIENTADO A OBXECTOS.

No paradigma da orientación a obxectos un sistema concíbese como un conxunto de obxectos que se comunican entre si mediante mensaxes.

1.1. DEFINICIÓN DE OBXECTO



A nivel conceptual un **obxecto** é unha entidade percibida no sistema que se está desenvolvendo.

A nivel de implementación, un obxecto é unha encapsulación do *estado* (valores de datos, atributos, campos, propiedades) e do *comportamento* (métodos, operacións) correspondentes á representación informática dun problema de maior nivel.

Un obxecto descríbese polos seus campos, tamén chamadas **atributos** e polo seu comportamento (**métodos**). O **estado** dun obxecto vén determinado polos valores que toman os seus atributos, valores que sempre han cumprir as restriccións impostas sobre eles.

Todo obxecto ten unha característica especial que o identifica univocamente denominada **identificador de obxecto**.

O obxecto é un elemento simple composto por tres elementos característicos: estado, comportamento e identidade. O obxecto encapsula o seu contido protexéndoo do exterior co que se consegue cumprir dous principios da programación robusta: maximizar a **cohesión** dentro do obxecto e minimizar o seu **acoplamento** co exterior. Os obxectos cumpren o seu papel de sacar adiante o funcionamento da aplicación mediante o intercambio de mensaxes con outros obxectos. É entón cando poñen de manifesto o seu comportamento.

1.2. CLASIFICACIÓN

Cada obxecto é un exemplar dalgunha **clase**, sóese dicir que o obxecto é unha **instancia** da clase. É dicir, a clase é unha xeralización do concepto obxecto. O comportamento dos obxectos queda deste xeito determinado pola clase á que pertencen. Tódolos obxectos da mesma clase invocan o mesmo método como resposta a unha solicitude similar.

1.3. PERSISTENCIA

Ó crear un obxecto, existe tanto tempo como sexa necesario, pero baixo ningunha circunstancia segue existindo unha vez que o programa acaba. Se ben esta circunstancia parece ter sentido a primeira vista, hai situacións nas que sería incriblemente útil manter a súa información incluso cando o programa xa non estea en execución. Desta maneira, a seguinte vez que se lance o programa, o obxecto estará aí e seguirá tendo a mesma información que tiña a última vez que se executou o programa. Por suposto, é posible lograr un efecto similar escribindo a información nun arquivo ou nunha base de datos, pero coa idea de facer que todo sexa un obxecto, sería desexable poder declarar un obxecto como persistente e facer que alguén ou algo se encargue de tódolos detalles, sen ter que facelo un mesmo.

1.4. COMUNICACIÓN ENTRE OBXECTOS: AS MENSAXES

O avance dun programa depende da colaboración entre os obxectos que o integran. Os obxectos comunícanse mediante o envío de mensaxes. Denominamos **mensaxe** á chamada recibida por un obxecto desde fóra ou dentro de si mesmo. Así pois, un obxecto activarase mediante a invocación a un método como resposta a unha mensaxe recibida.

2. REQUISITOS DO SOFTWARE.

2.1. CASOS DE USO.

Un caso de uso é unha interacción entre un usuario e un sistema informático.

Un caso de uso é un documento narrativo dunha secuencia de eventos para completar un proceso entre un axente externo ó sistema chamado actor e o sistema.

Os obxectivos dos casos de uso son os seguintes:

- Capturar os requirimentos funcionais do sistema e expresalos desde o punto de vista do usuario.
- Guiar todo o proceso de desenvolvemento do sistema de información.

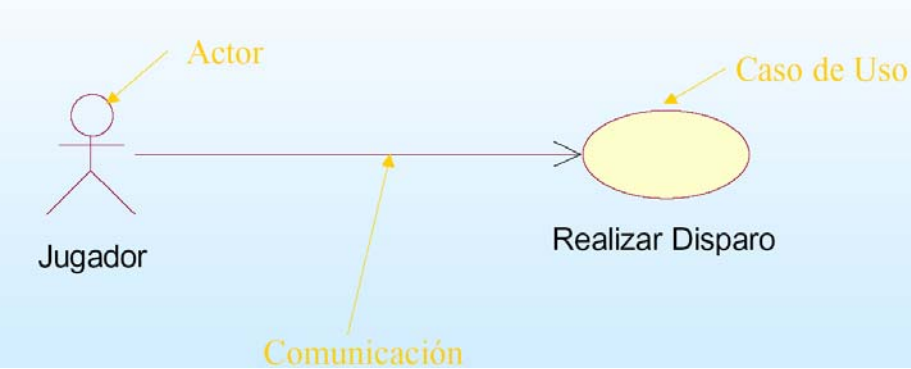
Os diagramas de casos de uso presentan dous tipos de elementos fundamentais:

- Un **actor** é algo ou alguén que se atopa fóra do sistema e que interactúa con el. En xeral, os actores serán os usuarios de sistema e aqueles sistemas alleos ó noso sistema de información. No caso de que falemos de usuarios, un actor é o papel que pode levar a cabo en canto á súa forma de interactuar co sistema. É dicir, un único actor pode representar a moitos usuarios diferentes e da masma maneira, un usuario pode actuar como actores diferentes.
- Un **caso de uso** é unha vista externa do sistema que representa algunha acción que o usuario podería realizar para completar unha tarefa. Tipicamente será un conxunto de transaccións executadas entre o sistema e os actores.

Ademais destes elementos, un diagrama de casos de uso presenta **relacións**. As relacións poden ter lugar entre actores e casos de uso ou entre casos de uso. UML define catro tipos de relación nos diagramas de casos de uso:

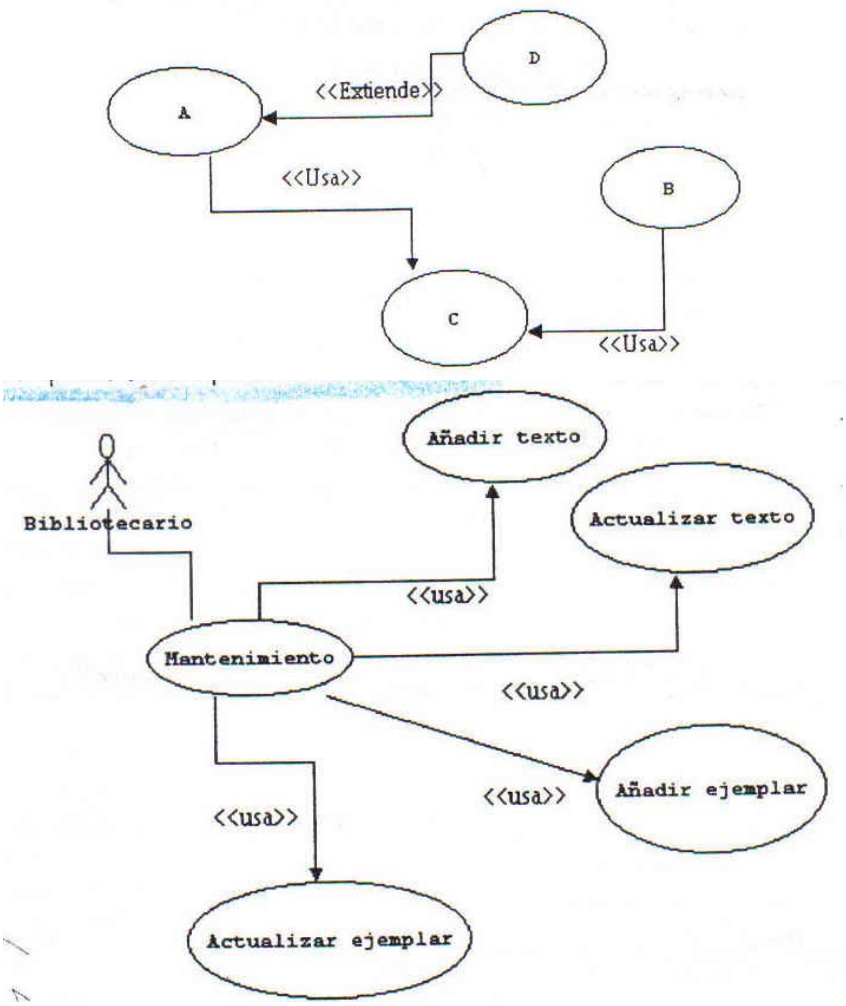
Comunicación

A relación entre un actor e un caso de uso é unha relación de comunicación, que indica que un actor intervén no caso de uso.



Inclusión

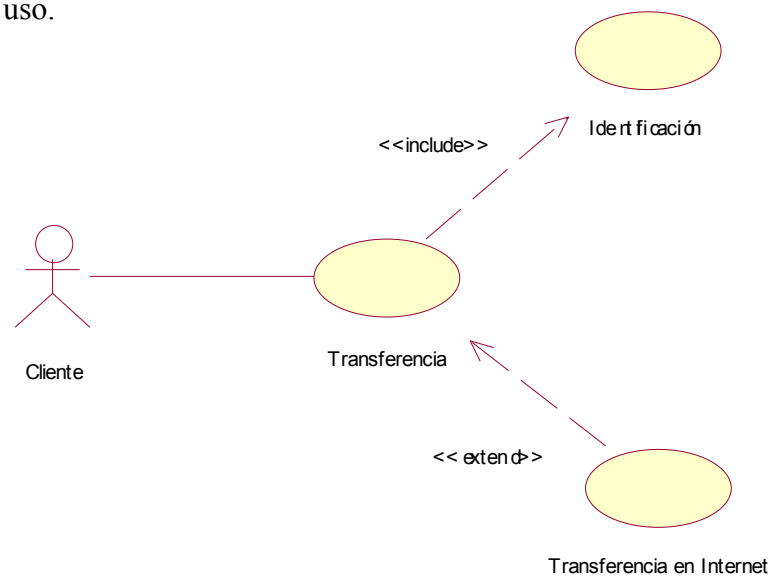
A relación "inclúe" utilízase cando se quere amosar un comportamento común en varios casos de uso. É dicir, se os casos de uso A e B presentan unha parte común, esta pódese extraer a un terceiro caso de uso C. Entón, haberá unha relación "inclúe" do caso de uso A ó C e outra do B ó C. "Inclúe" substituíu a "usa".



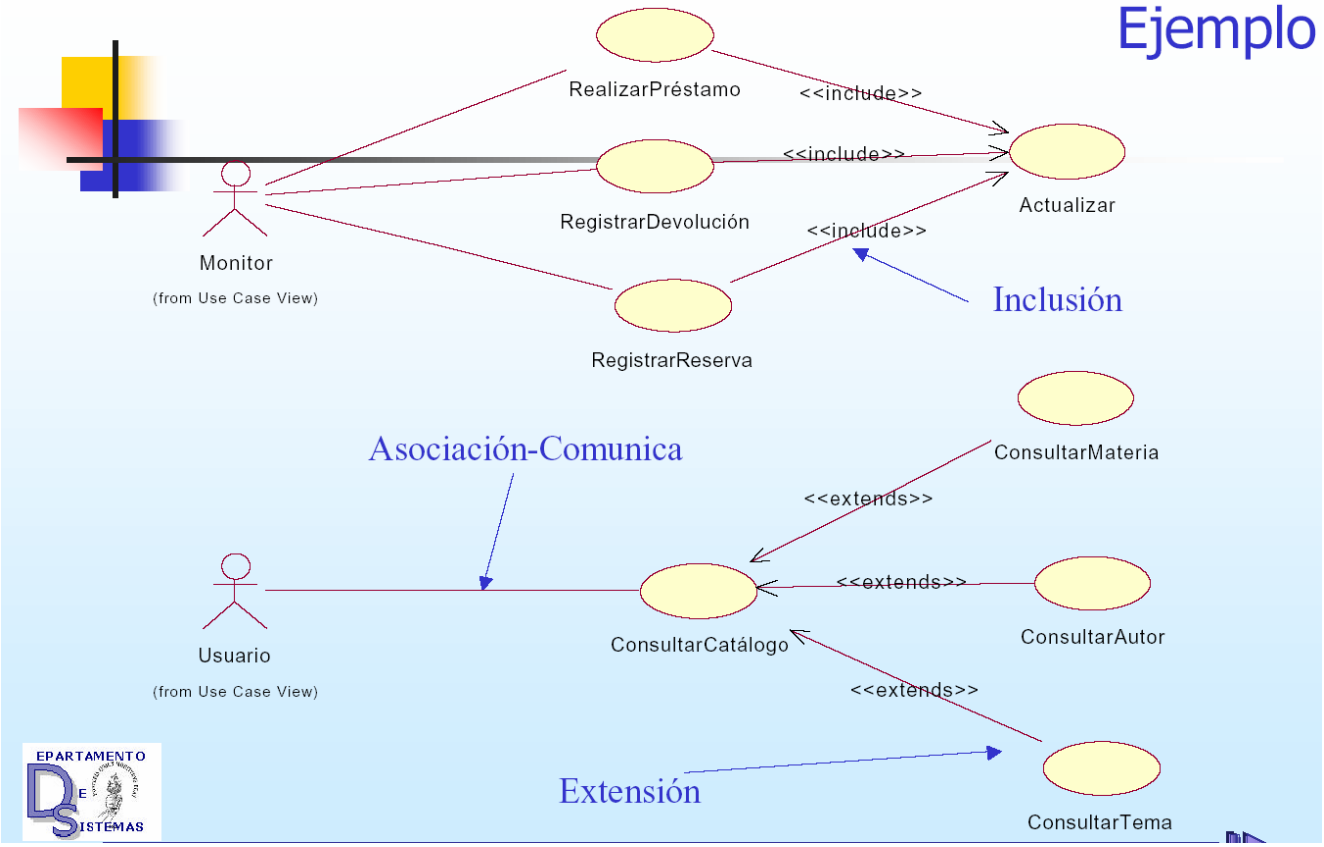
Extensión

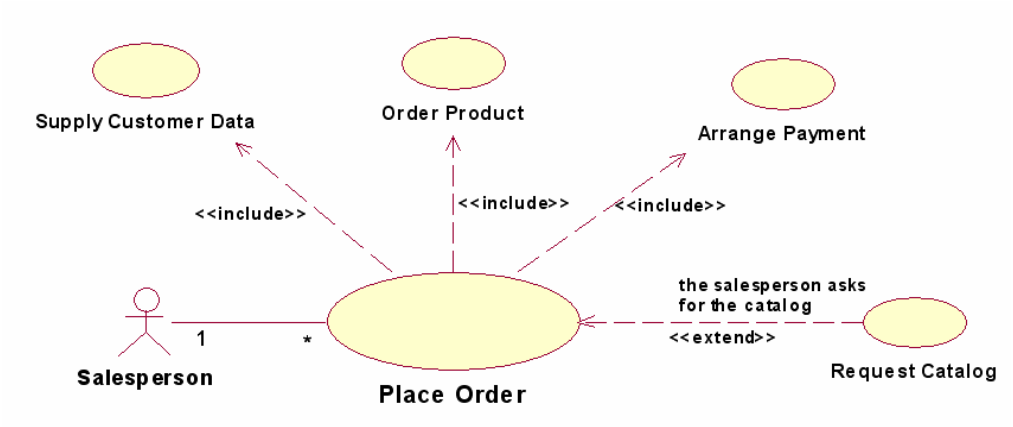
A relación "estende" utilízase cando se quere amosar un comportamento opcional dun caso de uso.

A relación "estende" pretende describir unha variación do comportamento normal dun caso de uso.



Un **escenario** é cada un dos distintos camiños polos que pode evolucionar un caso de uso, dependendo das condicións que se van dando na súa realización. É unha secuencia específica de accións.





Herdanza

O caso de uso orixe herda a especificación do caso de uso destino e posiblemente a modifica e/ou amplía.

3. INTERACCIÓN ENTRE OBXECTOS.

Os obxectos interaccionan para realizar colectivamente os servicios ofrecidos polas aplicacións. Os diagramas de interacción mostran como se comunican os obxectos nunha interacción.

Existen dous tipos de diagramas de interacción: Diagrama de Colaboración e o Diagrama de Secuencia.

3.1. DIAGRAMAS DE INTERACCIÓN

O Diagrama de Secuencia é máis adecuado para observar a perspectiva cronolóxica das interaccións.

O Diagrama de Colaboración ofrece unha mellor visión espacial mostrando os enlaces de comunicación entre obxectos.

O Diagrama de Colaboración pode obterse automaticamente a partir do correspondente Diagrama de Secuencia (ou viceversa).

3.2. DIAGRAMAS DE SECUENCIA

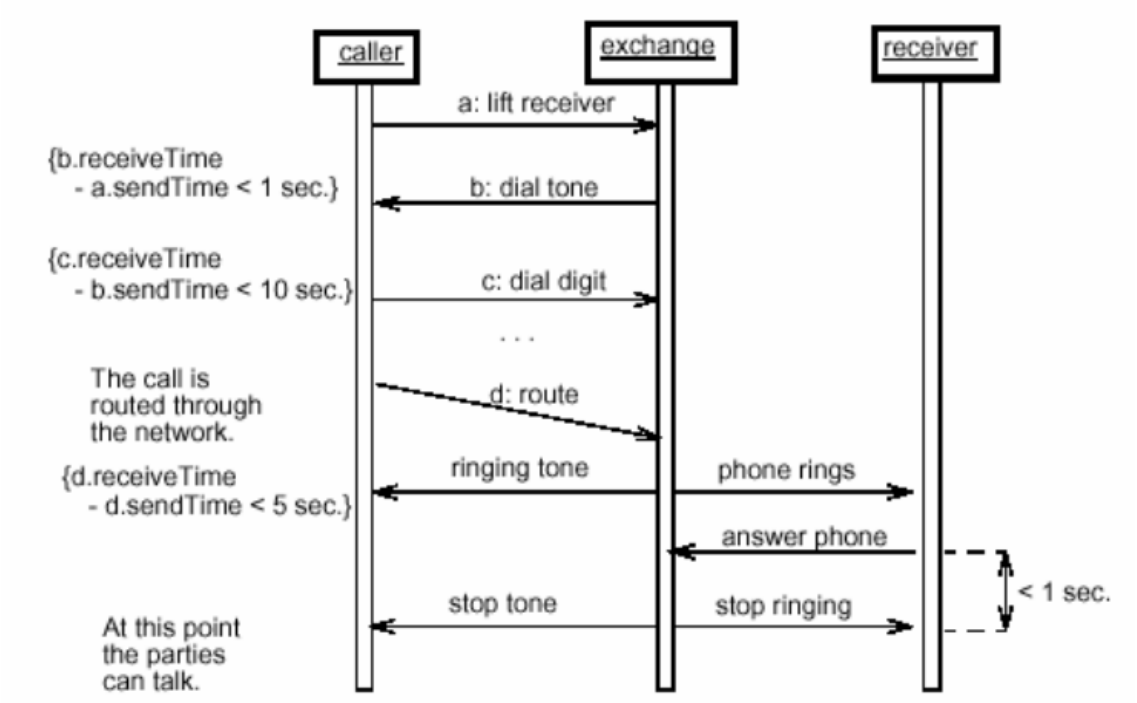
Mostra a secuencia de mensaxes entre obxectos durante un escenario concreto.

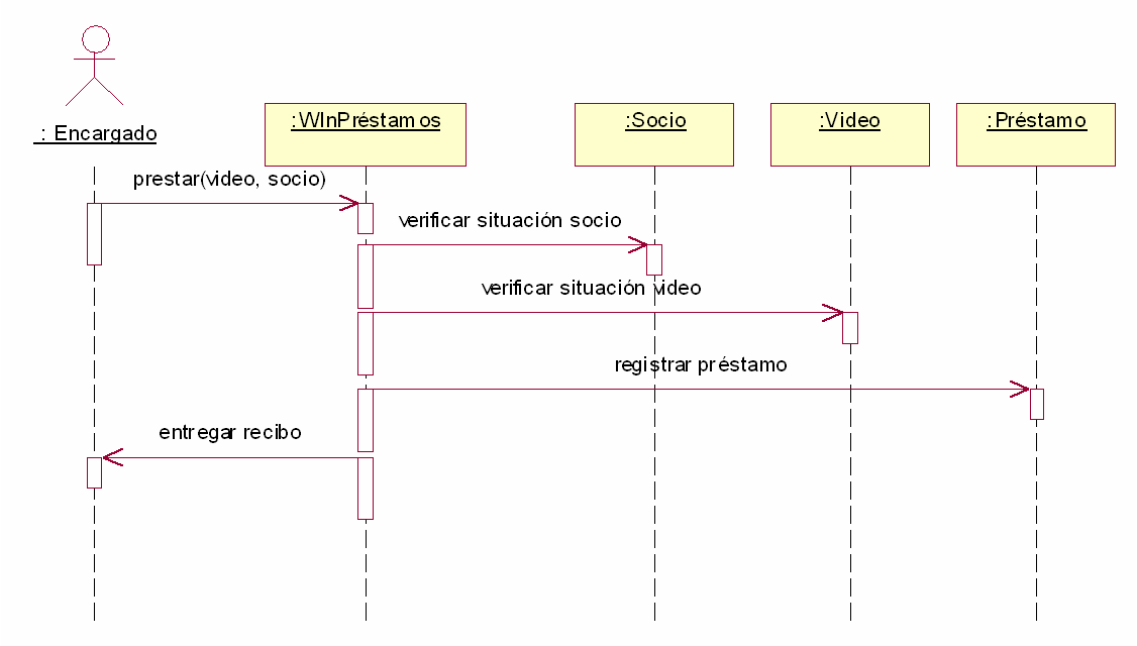
Céntranse nas secuencias de mensaxes, é dicir, como as mensaxes son enviadas e recibidas polos obxectos.

Cada obxecto ven dado por unha barra vertical.

O tempo transcorre de arriba abaixo.

Cando existe demora entre o envío e a atención pódese indicar usando unha liña oblicua.




: Encargado
:WInPréstamos
:Socio
:Video
:Préstamo
prestar(video, socio)
verificar situación socio
verificar situación video
registrar préstamo
entregar recibo

3.3. DIAGRAMAS DE COLABORACIÓN

Son útiles na fase exploratoria para identificar obxectos.

Modelan a interacción entre os obxectos dun caso de uso.

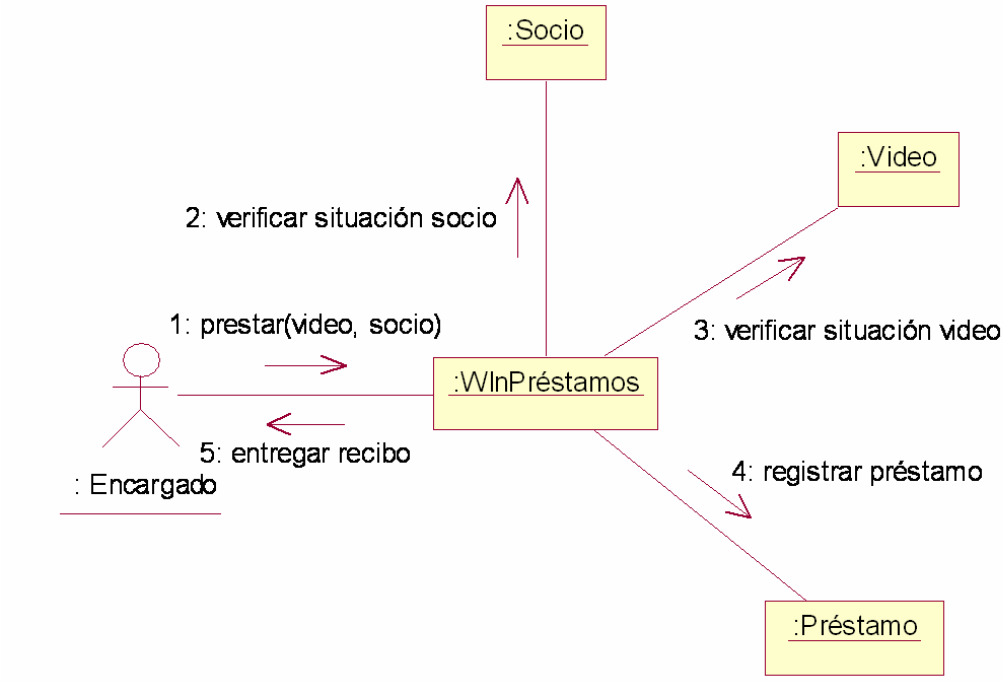
A distribución dos obxectos no diagrama permite observar adecuadamente a interacción dun obxecto con respecto dos demais.

Os obxectos están conectados por enlaces nos cales se representan as mensaxes enviadas acompañadas dunha frecha que indica a súa dirección.

O Diagrama de Colaboración ofrece unha mellor visión do escenario cando o analista está intentando comprender a participación dun obxecto no sistema.

Mentres os Diagramas de Secuencia se centran no tempo, os de Colaboración céntranse no espacio, é dicir, non mostran dimensión temporal.

Un Diagrama de Colaboración comeza cunha mensaxe que inicializa a interacción.

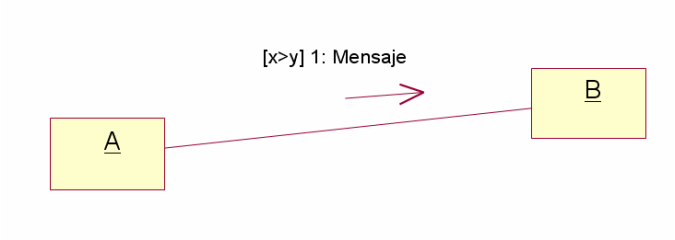


3.4. MENSAXES

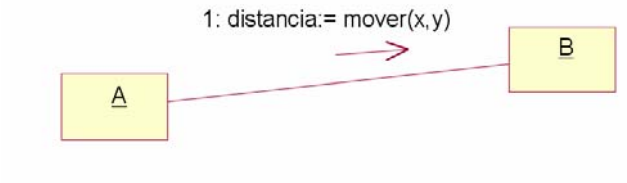
Unha mensaxe desencadea unha acción no obxecto destinatario.

Exemplos:

a. Mensaxe que se envía de maneira condicionada.



b. Mensaxe que devolve un resultado.



4. CLASES E RELACIÓNS ENTRE CLASES.

A clase define o ámbito de definición dun conxunto de obxectos.

Cada obxecto pertence a unha clase.

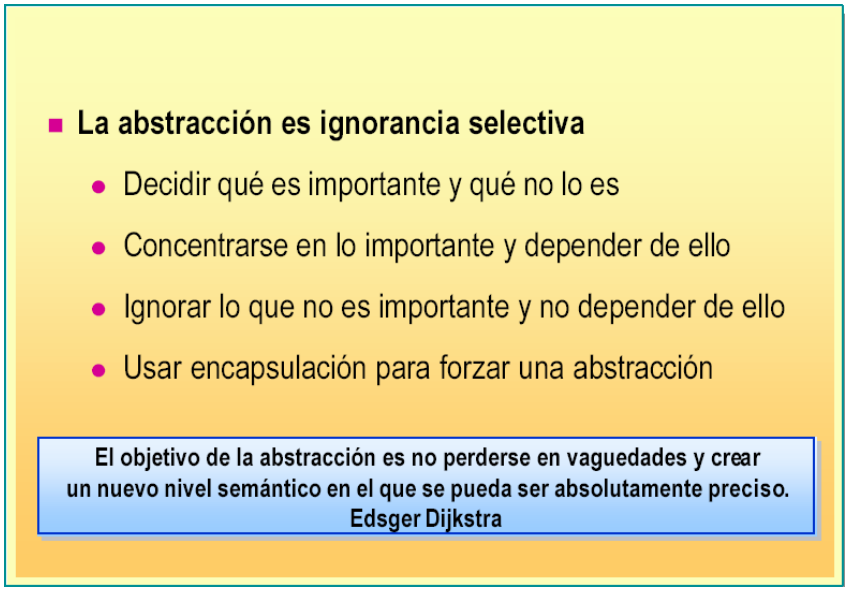
Os obxectos créanse por instanciación das clases.

4.1 NOTACIÓN GRÁFICA

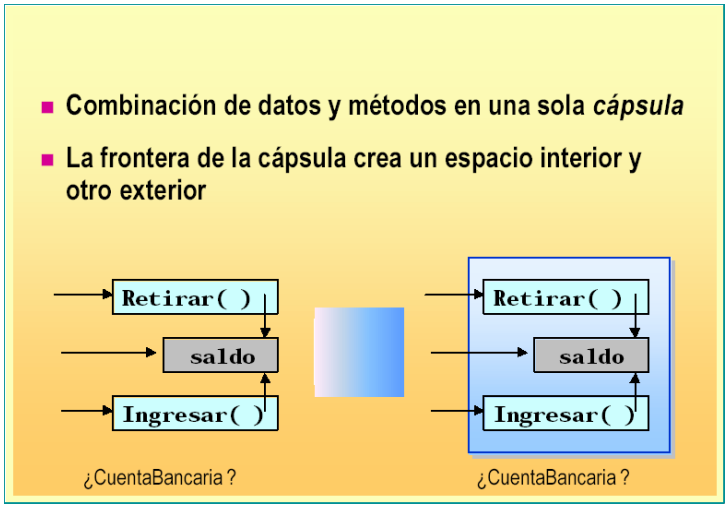
Cada clase represéntase nun rectángulo con tres compartimentos:

- Nome da clase.
- Atributos da clase.
- Operacións da clase.

4.2. ABSTRACCIÓN

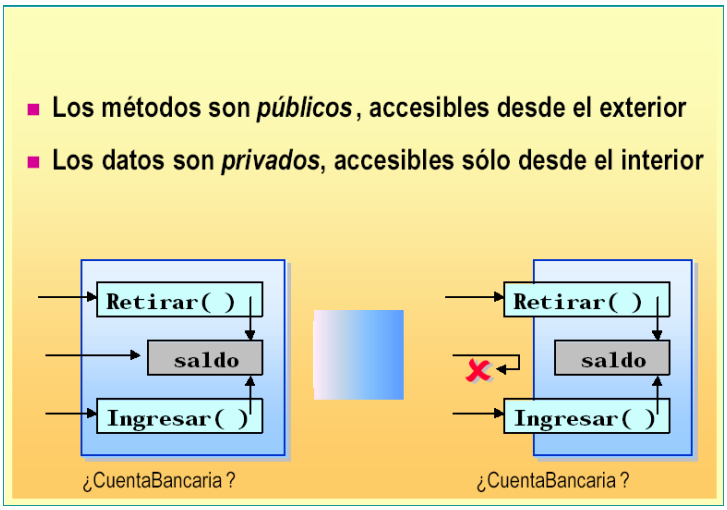


4.3. ENCAPSULACIÓN



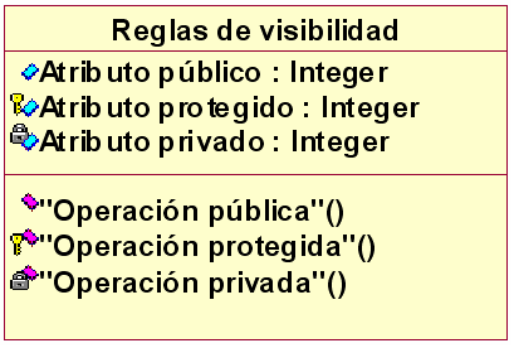
A encapsulación presenta dúas vantaxes básicas:

- Protéxense os datos de accesos indebidos.
- O **acoplamento** entre as clases disminúese.
- Favorece a **modularidade** e o mantemento.
- Os atributos dunha clase non deberían ser manipulables directamente polo resto de obxectos.

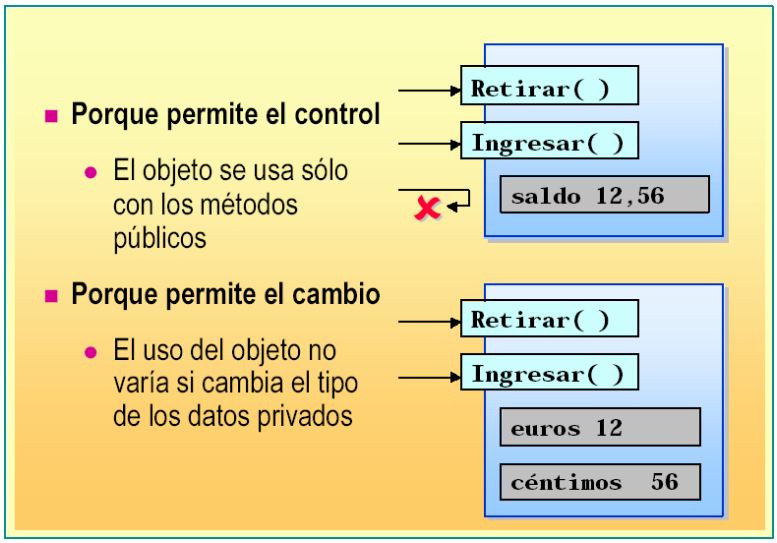


Os niveis de encapsulación están herdados dos niveis de C++:

- **(-) Privado**: é o máis forte. Esta parte é totalmente invisible (excepto para clases *friends* en terminoloxía C++)
- **(#)** Os atributos/operacións **protexidos** están visibles para as clases *friends* e para as clases derivadas da orixinal.
- **(+)** Os atributos/operacións **públicos** son visibles a outras clases (cando se trata de atributos estase transgredindo o principio de encapsulación).



¿Por que se encapsula?



4.4. RELACIÓNS ENTRE CLASES

Os enlaces entre obxectos poden representarse entre as respectivas clases.

Formas de relación entre clases:

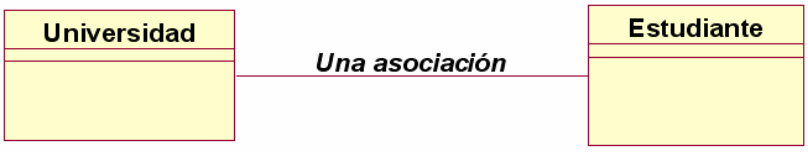
- Asociación e Agregación (vista como un caso particular de asociación)
- Xeralización/Especialización

As relacións de Agregación e Xeralización forman xerarquías de clases.

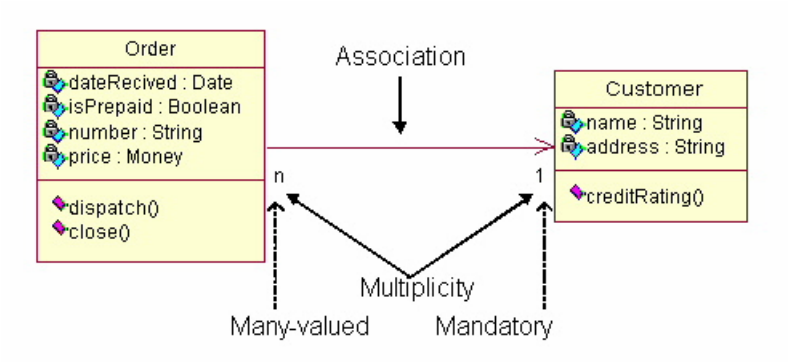
4.4.1. Asociación

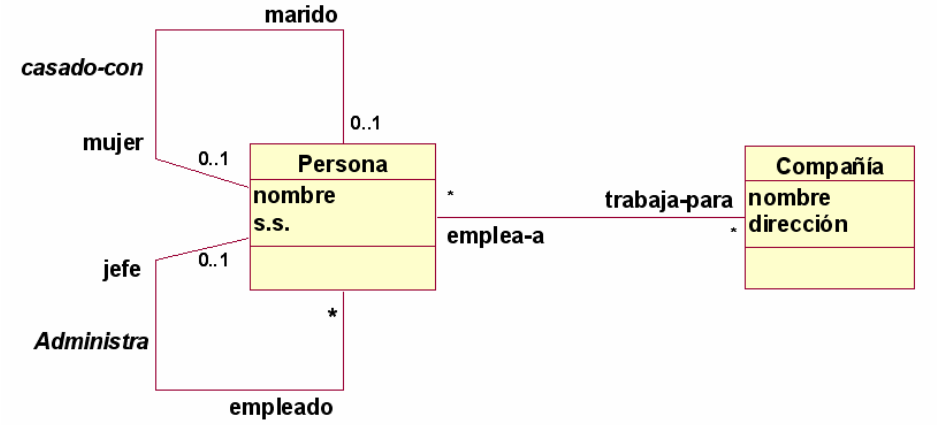
A asociación exprese unha conexión bidireccional entre obxectos.

Unha asociación é unha abstracción da relación existente nos enlaces entre os obxectos.



Exemplo:





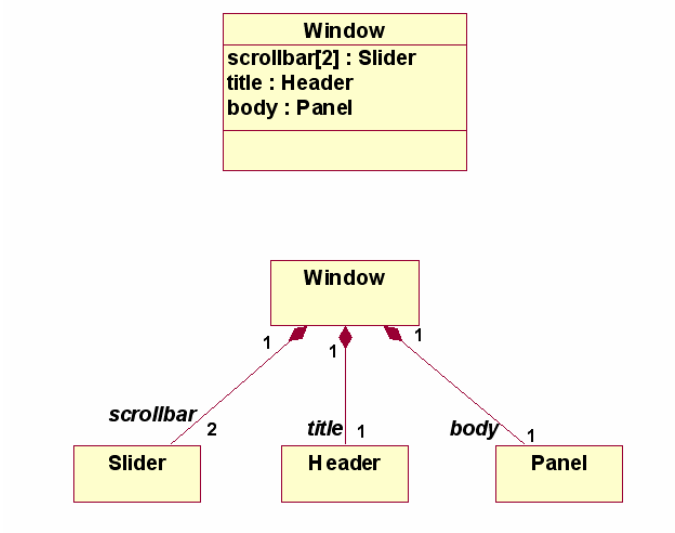
Especificación de multiplicidade (mínima ... máxima)

1	Un e só un
0..1	Cero ou un
M..N	Dende M hasta N (enteiros naturais)
*	Cero ou moitos
0..*	Cero ou moitos
1..*	Un ou moitos (polo menos un)

A multiplicidade mínima >= 1 establece unha restricción de existencia.

4.4.2. Agregación

A agregación representa unha relación *parte_de* entre obxectos.



4.4.3. Xeralización

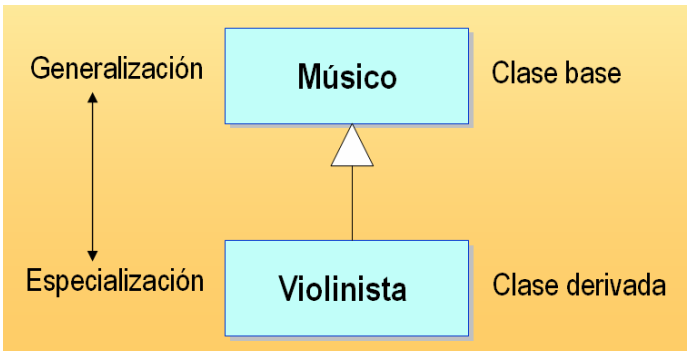
Obtense usando os mecanismos de abstracción de Xeralización e/ou Especialización.

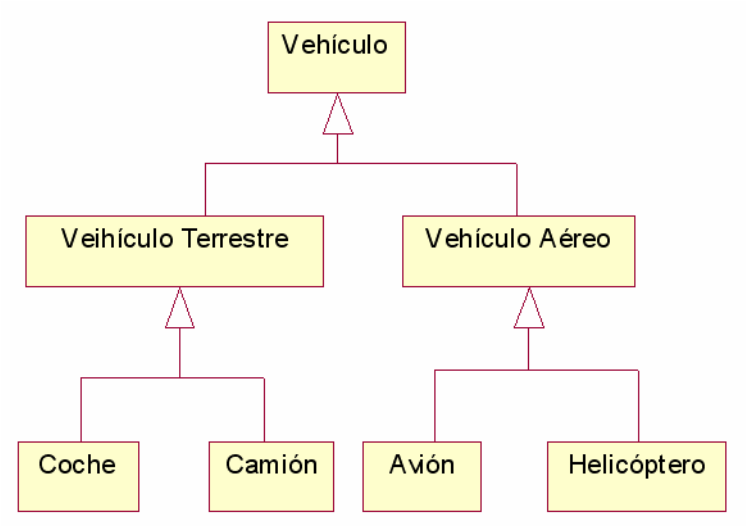
A Xeralización consiste en factorizar os campos comúns dun conxunto de clases nunha clase máis xeral.

Nomes usados: clase pai - clase filla. Outros nomes: superclase - subclase, clase base - clase derivada.

As subclases herdan campos das súas clases pai, é dicir, atributos e operacións (e asociacións) da clase pai están dispoñibles nas súas clases fillas.

Xeralización e Especialización expresan relacións de inclusión entre conxuntos.



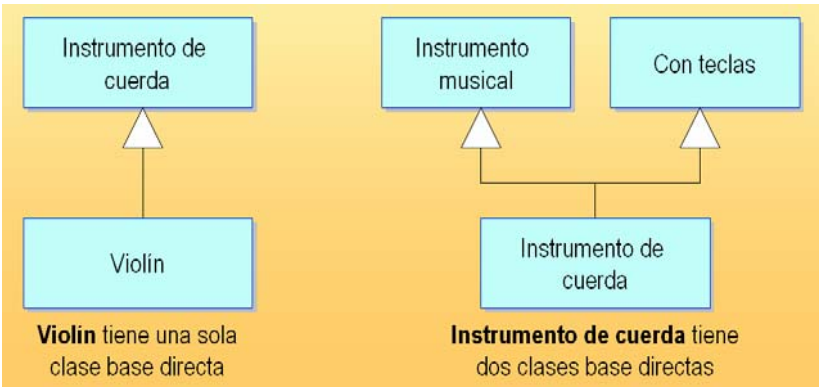


A Especialización é unha técnica moi eficaz para a reutilización.

4.4.4. Clasificación múltiple (herdanza múltiple)

Preséntase cando unha subclase ten máis dunha superclase.

A herdanza múltiple debe manexarse con precaución. Recoméndase un uso restrinxido e disciplinado da mesma. Java e Ada 95 simplemente non ofrecen herdanza múltiple.



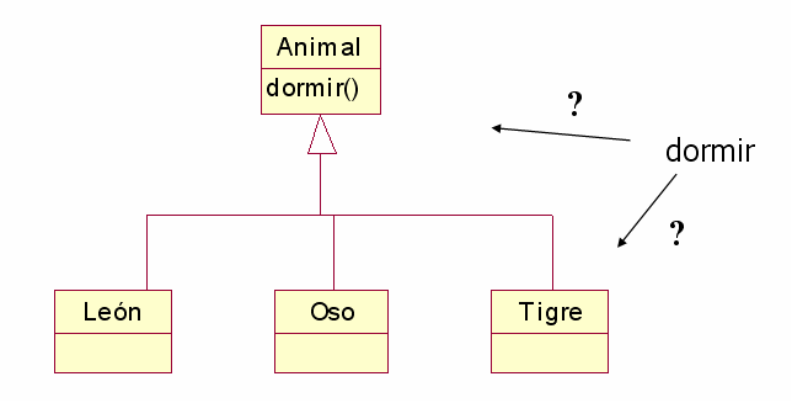
4.5. POLIMORFISMO

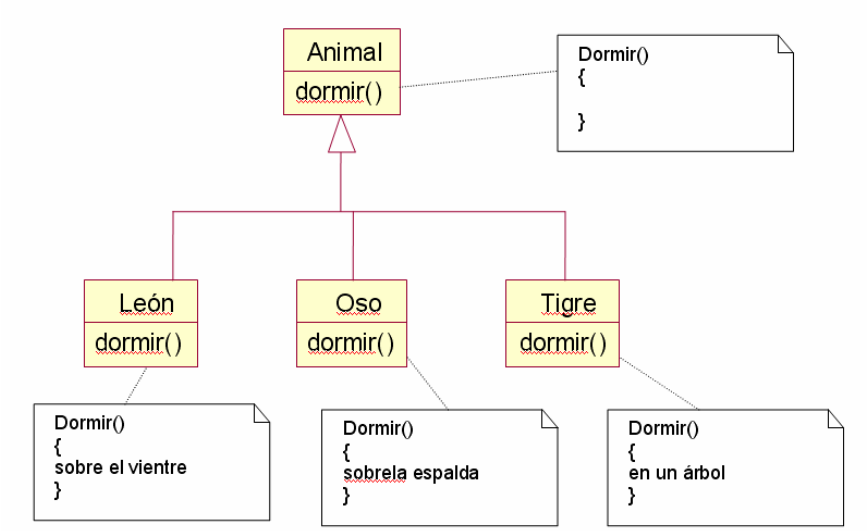
O termo polimorfismo refírese a que unha característica dunha clase pode tomar varias formas.

O polimorfismo representa no noso caso a posibilidade de desencadear operacións distintas en resposta a unha mesma mensaxe.

Cada subclase herda as operacións pero ten a posibilidade de modificar localmente o comportamento desas operacións.

Exemplo: todo animal durme, pero faino de diferente maneira





Exemplo: Podemos crear unha clase denominada CadeaDeCaracteres na que definiremos o método suma consistente en concatenar secuencialmente dous obxectos desta clase obtendo como resultado outro obxecto da mesma clase. Posteriormente podemos definir para esta clase a mensaxe suma e vinculala ó símbolo '+'. Por outro lado podemos crear unha clase denominada NumeroEnteiro na que definiremos o método suma consistente en sumar dous obxectos desta clase obtendo como resultado outro obxecto da mesma clase. Posteriormente tamén definiremos para esta clase a mensaxe suma vinculándoo ó símbolo '+'.

Así a expresión A+B podería ser unha suma de enteiros (se A e B son enteiros) ou unha suma de cadeas de caracteres (se A e B o son). Corresponde á mesma mensaxe + identificar a clase destinataria en función dos parámetros que se lle aporten.

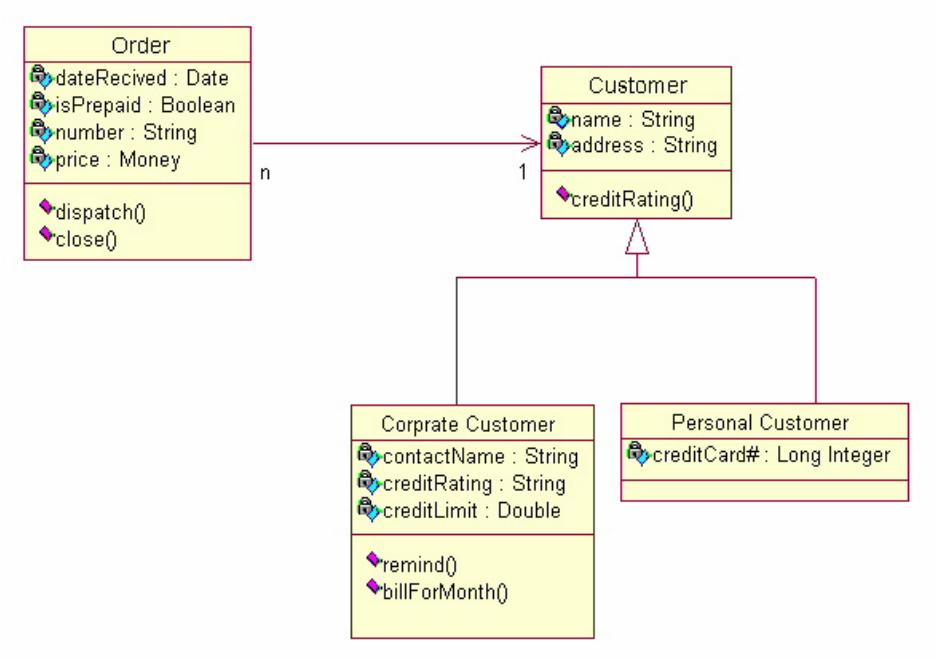
4.6. DIAGRAMA DE CLASES

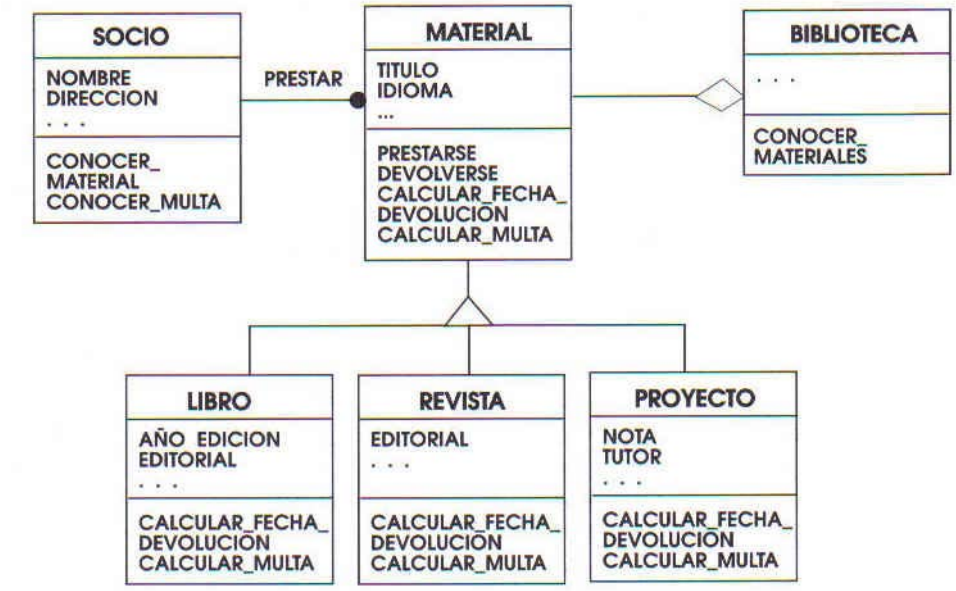
O Diagrama de Clases é o diagrama principal para a análise e deseño do sistema.

Un diagrama de clases presenta as clases do sistema coas súas relacións estructurais e de herdanza.

A definición de clase inclúe definicións para atributos e operacións.

O modelo de casos de uso debería aportar información para establecer as clases, obxectos, atributos e operacións.





5. COMPORTAMENTO DE OBXECTOS.

5.1 DIAGRAMA DE ESTADOS

Os diagramas de estados constitúen unha técnica que permite poñer de manifesto o comportamento dependente do tempo dun sistema de información. Mediante o seu uso represéntanse os estados que pode tomar un compoñente ou un sistema e móstranse os eventos que implican o cambio dun estado a outro.

Son útiles só para os obxectos cun comportamento significativo.

Cada obxecto está nun estado en certo instante.

O estado está caracterizado parcialmente polos valores dalgúns dos atributos do obxecto.

O estado no que se atopa un obxecto determina o seu comportamento.

Cada obxecto segue o comportamento descrito no diagrama de estados asociado á súa clase.

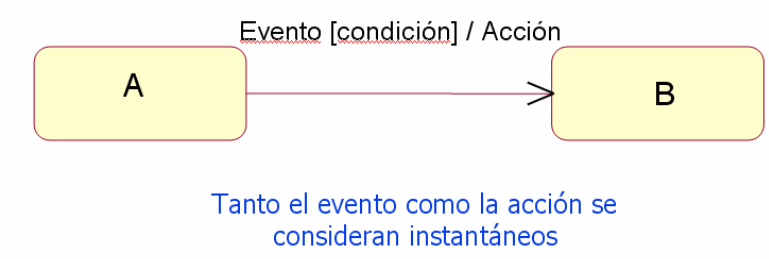
Os diagramas de estados e os escenarios son complementarios.

Os estados inicial e final están diferenciados do resto.

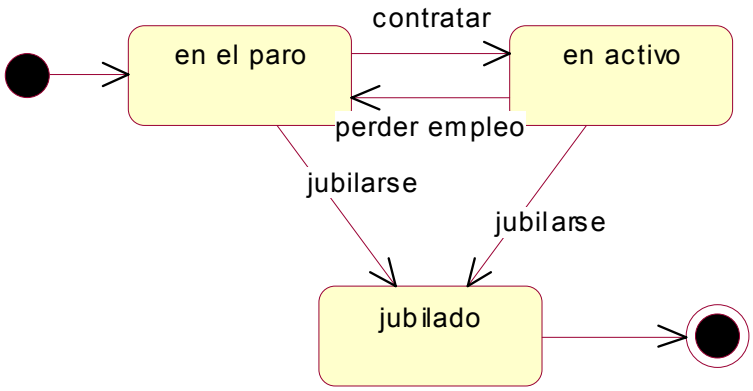
A transición entre estados é instantánea e débese á ocorrencia dun evento.

Os dous elementos principais nos diagramas de estado son:

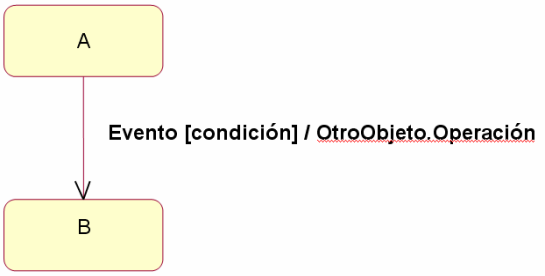
- O **estado** dun compoñente ou sistema: representa algún comportamento que é observable externamente e que perdura durante un período de tempo finito.
- Unha **transición** é un cambio de estado producido por un evento e reflexa os posibles camiños para chegar a un estado final desde un estado inicial, estados que non teñen necesariamente que ser diferentes.



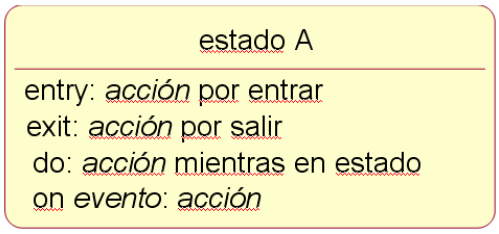
Exemplo dun diagrama de estados para a clase Persoa:



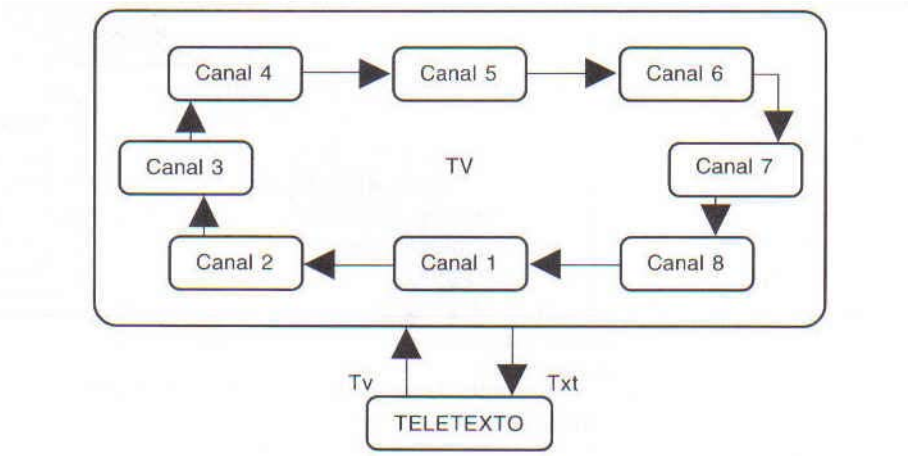
Podemos especificar a solicitude dun servicio a outro obxecto como resultado da transición:



Pódese especificar executar unha acción como consecuencia de entrar, saír, estar nun estado, ou pola ocorrencia dun evento:



Cando se trata un diagrama de transición de estados cun maior nivel de detalle e se comproba que unha actividade dispara unha secuencia de eventos para completarse, pódese crear un novo diagrama de transición de estados, anidado no anterior, para representar con maior detalle esa actividade. O diagrama de estados incluído únese a un estado do diagrama de orde superior mediante eventos de entrada e de saída.

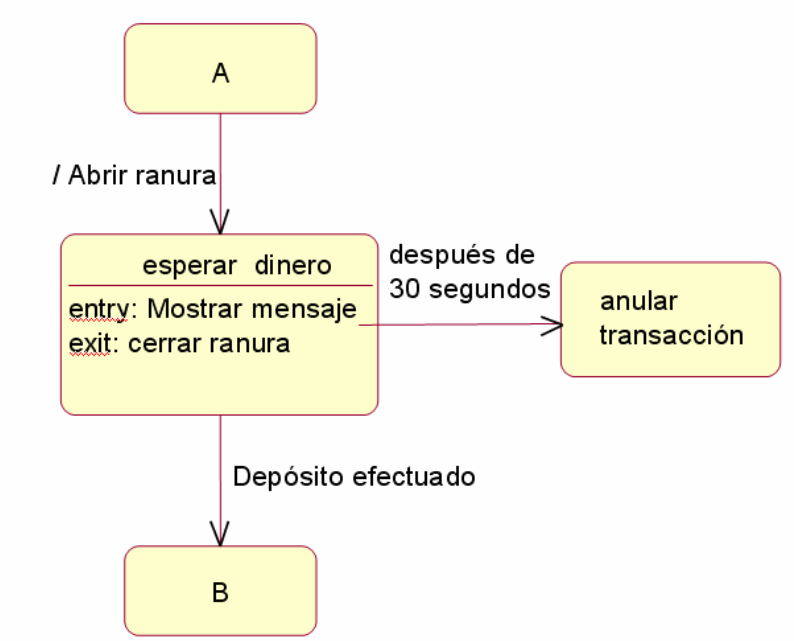


Transicións temporizadas

As esperas son actividades que teñen asociada certa duración.

A actividade de espera interrómpese cando o evento esperado ten lugar.

Este evento desencadea unha transición que permite saír do estado que alberga a actividade de espera. O fluxo de control transmítese entón a outro estado.



5.2 DIAGRAMA DE ACTIVIDADE

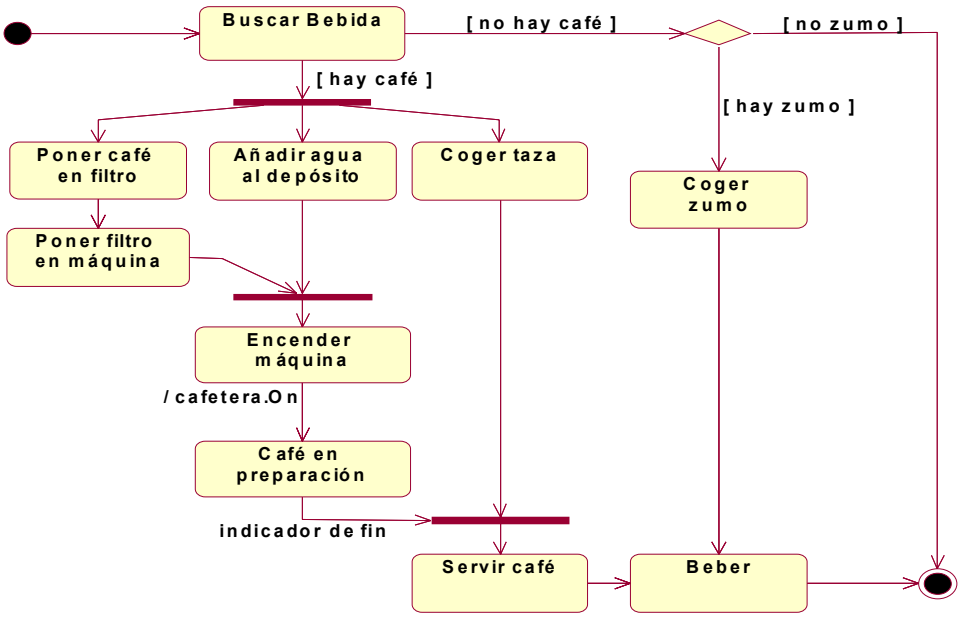
O diagrama de actividade é unha especialización do diagrama de estado, organizado respecto das accións e usado para especificar: un método, un caso de uso ou un proceso de negocio.

Mostra os pasos a realizar para levar a cabo unha operación.

As actividades enlázanse por transicións automáticas. Cando unha actividade remata desencadéase o paso á seguinte actividade.

Consta de:

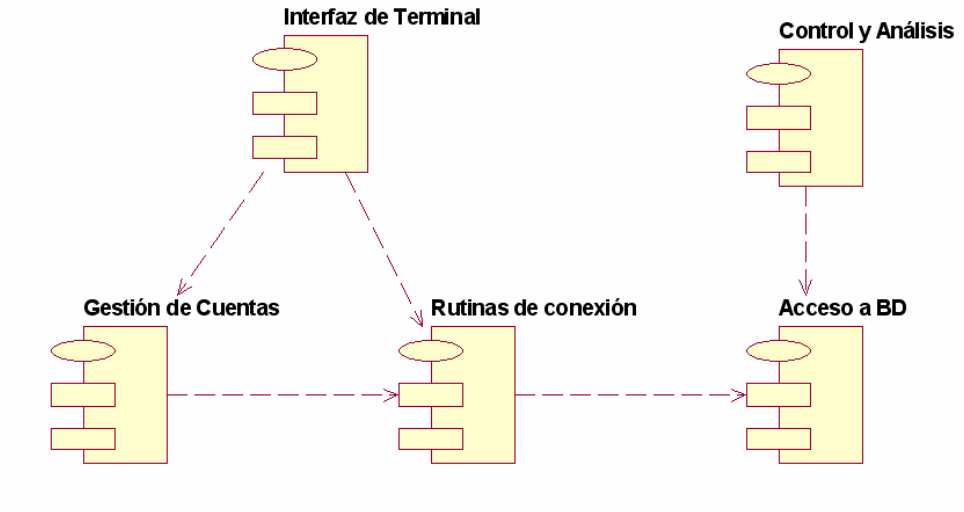
- Punto inicial e punto final para indicar onde comeza e onde remata a implementación da operación.
- Accións que mostran o que a operación vai realizando.
- Transicións entre accións.
- Puntos de decisión ou bifurcacións (con rombos).



6. COMPOÑENTES.

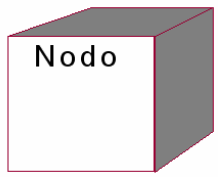
Os compoñentes representan tódolos tipos de elementos software que entran na fabricación de aplicacións informáticas. Poden ser simples arquivos, paquetes de Ada, bibliotecas cargadas dinamicamente, etc.

As relacións de dependencia utilízanse nos diagramas de compoñentes para indicar que un compoñente utiliza os servicios ofrecidos por outro compoñente.



7. DISTRIBUCIÓN E DESPREGUE DE COMPOÑENTES.

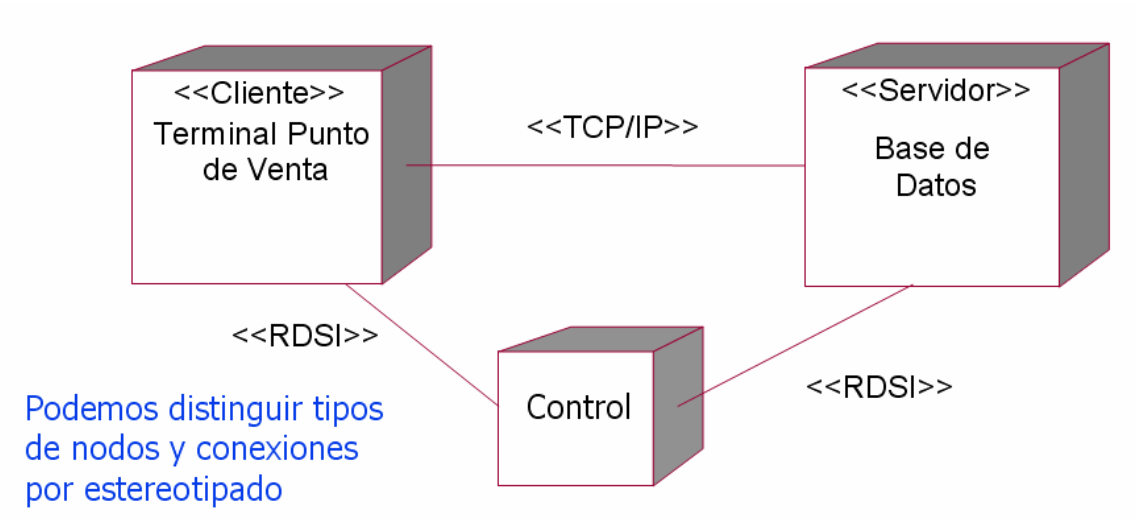
Os **Diagramas de Despregue** mostran a disposición física dos distintos nodos que compoñen un sistema e o reparto dos compoñentes sobre eses nodos.



Os estereotipos permiten precisar a natureza do equipo: dispositivos, procesadores, memoria.

Os nodos interconéctanse mediante soportes bidireccionais que poden á súa vez estereotiparse.

Exemplo de conexión entre nodos:



8. UML (Unified Modelling Language¸ Linguaxe de Modelado Unificada)

O UML é unha linguaxe estándar para especificar, visualizar, construír e documentar os sistemas software, así como para modelado de empresa e outros sistemas non relacionados co software. O UML representa unha colección das mellores prácticas enxeñería que foron aplicadas con éxito no modelado de sistemas complexos e de envergadura. O UML é unha parte moi importante no desenvolvemento de software orientado a obxectos e no proceso de desenvolvemento do software.

O UML utiliza principalmente notacións gráficas para expresar o deseño de proxectos software. Usar UML axuda a que os grupos de proxectos se comuniquen, exploren deseños potenciais e validen o deseño da arquitectura do software.

UML é unha "linguaxe" para especificar pero non é un procedemento nin un método.

Cada diagrama UML se deseña para permitir que analistas e clientes vexan un sistema software desde unha perspectiva diferente e en diferentes graos de abstracción.

8.1 DIAGRAMAS DE UML

Diagrama de Casos de Uso.

Diagrama de Clases.

Diagrama de Obxectos.

Diagramas de Comportamento.

- Diagrama de Estados.
- Diagrama de Actividade

Diagramas de Interacción.

- Diagrama de Secuencia.
- Diagrama de Colaboración.

Diagramas de Implementación.

- Diagrama de Compoñentes.
- Diagrama de Despregue.