Дисциплина: Програмни Спецификации

Упражнение 6-7-8-9. UML диаграма на класовете. Използване на StarUML за създаване на клас диаграми - един пример (Java профил): Forward и Reverse Engineering

Цел: запознаване с UML диаграми на класовете: основни понятия, използвани символи. Използване на StarUML за създаване на клас диаграма, генериране на код от модел (Forward Engineering) и възвратно проектиране (Reverse Engineering)

1. UML диаграма на класовете

Основни понятия

Класове — това са базови елементи на всяка ОО система. **Класовете** представят множество обекти с еднакви свойства: атрибути, операции, отношения и семантика. В рамките на създавания „Обектен модел" на системата, на всеки клас в UML се присвоява уникално име (съществително име в единствено число, например Student, Account...), отличаващо го от другите класове.

Ако се използва съставно име (в началото се добавя името на пакета, в който е класа), като името на класа трябва да е уникално в пакета.

Атрибут — това е свойство на класа, което може да приема множество значения. Атрибутът има име и отразява някое свойство на моделираната същност, общо за всички обекти на дадения клас. Класът може да има произволно количество атрибути.

Операция — реализация на функция, която може да се извика (поиска) за всеки обект на класа. Изпълнението на операцията често е свързано с обработка и изменение на стойността на атрибутите на обекта, а така също и на състоянието му.

Прието е в UML, атрибутите и операциите на един клас да се наричат общо свойства (**features)** на клас. Свойствата (properties) са структурните възможности на един клас, на първо време – полетата на класа, независимо че на практика се разбират доста повече неща, често и атрибутите и операциите (методите на класа) заедно.

Клас диаграми

Клас диаграмите се явяват основни при анализ на изискванията и проектиране на ОО Системи, тък като позволяват нагледно да се изобрази структурата на класовете в приложенията и статичните връзки между тях. Това са **структурни UML диаграми,** чрез които се описва **статичната структура на системата или на части от нея**.

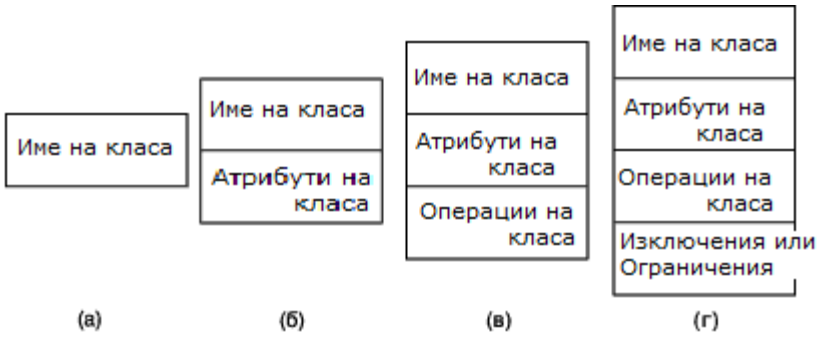
Кога се използват клас диаграми?

Използват се по време на анализиране на изискванията за моделиране на обектите от реалния свят и по време на проектиране - за моделиране на подсистемите. Такива диаграми са полезни както при предварително проектиране, така и при рефакторинг, съпровождане и поправяне на грешки, а така също и при изучаване на ПО. По тези диаграми се генерира програмния код, както и - те лесно се възстановяват по съществуващ програмен код.

Графично изобразяване на клас

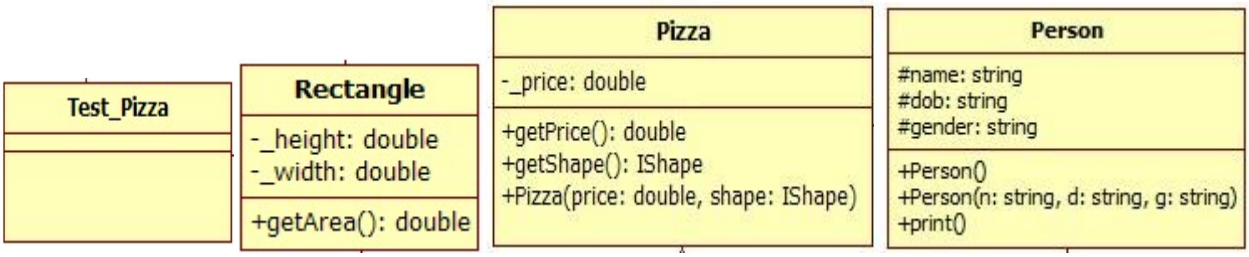
Как се изобразява графично класа?

В UML, класът се изобразява като правоъгълник, който допълнително може да бъде разделен на раздели или секции с хоризонтални линии. Последната секция в (г) е незадължителна. В тези секции се указва името на класа, атрибутите и операциите му:



фигура 1.

Още примери:



фигура 2.

Ако класа принадлежи на пакет, то името му се задава по следния начин:
<Име на пакета>::< Име на класа >

Един клас може да участва в няколко диаграми на класовете.

Това е графично изображение на клас „Поръчка” в UML: показани са името на кла

фигура 3. Изображение на клас в UML

Синтаксисът на UML за свойствата на класа

Синтаксисът на UML за свойствата на класа - атрибути и операции е следния:

- Пълната форма на **атрибута** е:

<видимост> <име на атрибута> : <тип> = <стойност по подразбиране>
или
<видимост> <име на атрибута> [кратност] : <тип> = <стойност по подразбиране>

- **Операции** - описват по следния начин:

<видимост> <име на операция> <(списък на аргументи)>
<видимост> <име на операция> <(списък на аргументи)>: return-type

Например: **+ balanceOn (date: Date) :Money**

Видимост на свойство (атрибут или операция)

Видимостта на едно свойство (атрибут или операция) указва възможността за използването му от другите класове. Един клас може да види друг, ако последния се намира в същата област на действие и между тях съществува явно или неявно отношение.

В UML има 3 нива на видимост на свойствата – public, private и protected. Ако видимостта отсъства, то в UML няма подразбираща се стойност!

- public (обща) — всеки външен клас, който вижда дадения клас може да се ползва от неговите общи свойства). Обозначава се с public или с " + " пред името на атрибута или операцията;
- protected (защитен) — класът е предоставя свойствата си (атрибути или операции) само за класовете - потомци. Обозначава се с " # ";
- private (недостъпен) — само дадения клас може да ползва свойствата си (атрибути или операции). Обозначава се с " - " .

package - видимост като на пакета – " ~ "

Област на действие на свойство (атрибут или операция)

Област на действие: Това е друга важна характеристика на атрибутите и операциите на класа. Областта на действие указва, дали даденото свойство по различен начин ще се проявява във всеки екземпляр на класа, или даденото свойство съвместно ще се ползва от всички екземпляри на класа:

- instance (екземплярно) — всеки екземпляр на класа има собствена стойност за това свойство на класа;
- classifier (класификатор) — всички екземпляри на класа съвместно ще ползват общата стойност на това свойство (в диаграмите се подчертава).

Кратност (multiplicity) на атрибут

Кратност (multiplicity) на атрибута — спецификация на допустимата мощност на областта от значения на атрибута, тоест колко стойности може да има този атрибут. Задава се най-често под формата на [долна граница .. горна граница]. Кратност 1 или 1..1, означава 1; Кратност " * " в качеството на горна граница замества 0 или произволно положително число.
Например, атрибут с име: Attribute1, Integer тип, който може да има стойности от 0 до 3 се задава като: **Attribute1 : Integer [0..3].**

Кратност (multiplicity) на клас

Принципно, вие можете да задавате свойство „кратност" както за атрибути на класове, така и за асоциациите между класовете. В последния случай, кратността показва броя на екземплярите (обектите), които могат да участват във връзката (relationship) между класовете.

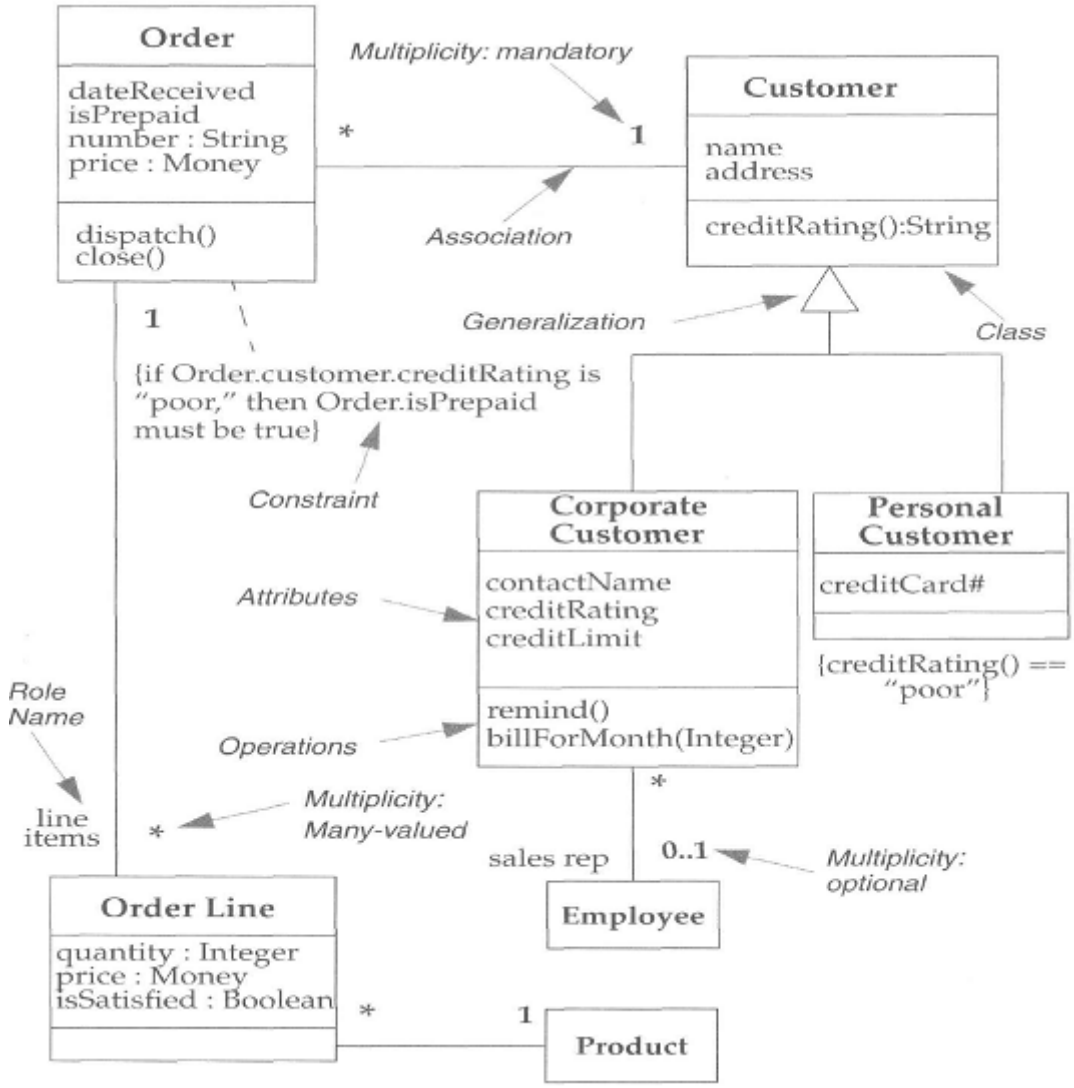
Кратността може да е конкретно положително цяло число, диапазон от цели числа и последователност от цели числа, разделени със запетая. * - означава не-лимитирана горна граница. Ето няколко промери за спецификации на кратността:

1 - Точно 1
0..1 - 0 или 1
* - Много, всяко число, включително 0
n - Много, всяко число, включително 0
1..* - Едно или много
0..* - означава 0 или много
Константа (например, 2, 10, 100).
Интервал (например, 3..5, 10..20).

В този контекст, в UML има следните разновидности на класове:

- клас, който не съдържа нито един екземпляр - тогава класът служебно става (Abstract). Ако класа е абстрактен, то името му е в курсив.
- Клас, който съдържа един екземпляр (Singleton).
- Клас, който съдържа фиксиран брой екземпляри.
- Клас, който съдържа произволно количество екземпляри.

Пример (на Фаулър): Диаграма на класове за on-line магазин:



Фигура 4

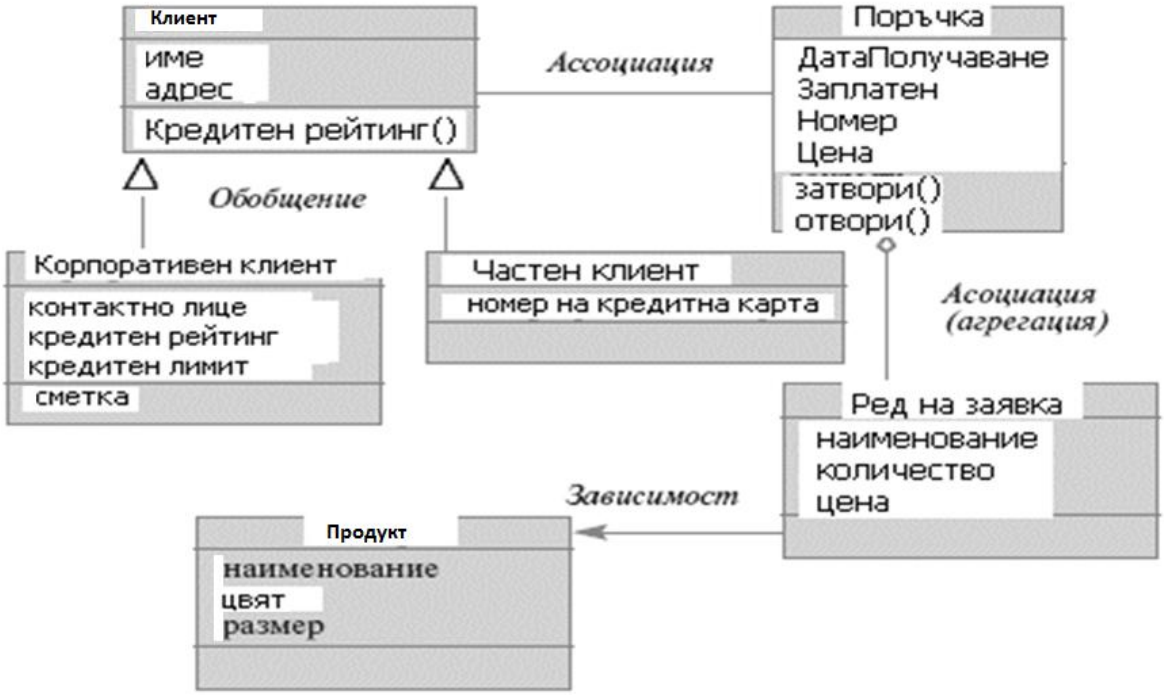
Правила за ограничения

В диаграмите на класовете се описват и ограничения. На фигура 4 е зададено ограничение - това е текстът, заключен във фигурни {Ако ...то OderisPrepaid=true}. Смисълът на ограничението в случая е, че поръчката ще е предплатена за частни клиенти (тогава OderisPrepaid=true).

Отношения между класовете

Между класовете са възможни различни отношения, представени на клас диаграмата по-долу:

- **зависимости** (dependency relationship), които описват съществуващо между класовете отношение на използване;
- **обобщения** (generalization relationship), свързващи обобщените класове със специализираните;
- **асоциации** (association relationship), отразяващи структурни отношения между обектите на класовете.



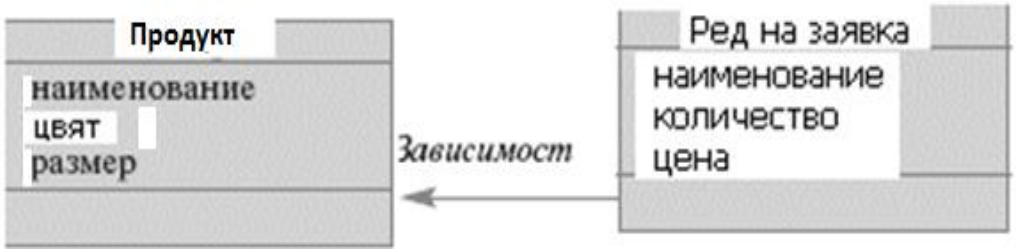
фигура 5. Между класовете са възможни различни отношения

Други, считани също за базовите отношения между класовете са:

- Отношение на **агрегация** (aggregation relationship)
- Отношение на **композиция** (composition relationship)

Забележка: Последните две отношения, от някои автори се считат за вариации на **association relationship**.

Зависимост се нарича **отношение на използване**, съгласно което, изменението на елемент в спецификацията на един клас, например, на клас "**Продукт**" (фигура 5) може да повлияе на класа, който го използва, например - клас "**Ред на заявка**" (в случая, клас „Ред на заявка" е зависим от клас "продукт"). Зависимостта се изявява по различни начини.



фигура 6

Ключовите думи за зависимост са **create**, **use**, **call**, **derive**, **use**, **permit**, **realize**…, например:

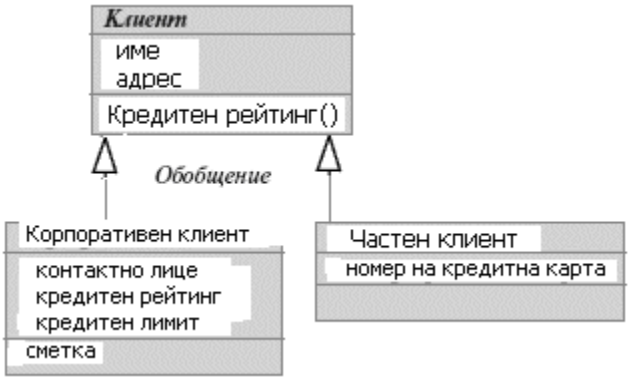
- един клас използва друг в качеството на аргумент,
- един клас указва друг като параметър на своя операция,
- един клас А изпраща съобщение на втори клас С - **асоциацията** е с посока от А към С, тоест А е зависим от С.

При последния случай: ако се окаже, че при промяна на интерфейса на клас С, съобщенията които изпраща клас А не се получават в С, то се казва, че клас А е зависим от С, но С не е зависим от А, тоест спокойно може да променяме интерфейса на А без това да се отрази на С (еднопосочна зависимост).

UML позволява да указвате различни зависимости… Препоръките са да ги минимизираме.

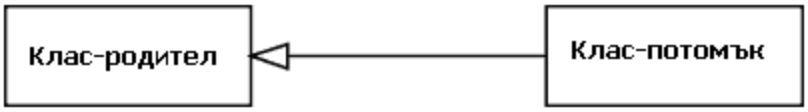
На фигура 4 асоциацията с посока от Order към Customer показва, че Order е зависим от Customer, но не и обратно (тъй като Customer create Order).

Обобщение (generalization) — това е отношение между общата същност (родител), нейните по-малко общи потомци, например "клиенти" („родител” <— клас "клиент") и нейните конкретни потомци - "корпоративен клиент " или " частен клиент ".
Класът – потомък наследява всички свойства на родителя (атрибути и операции). Операцията на потомъка със същата сигнатура замества тази на родителя: това, както знаете се нарича полиморфизъм. Клас, който няма родители, но има потомци се нарича коренов. Класове, които нямат потомци – листа.

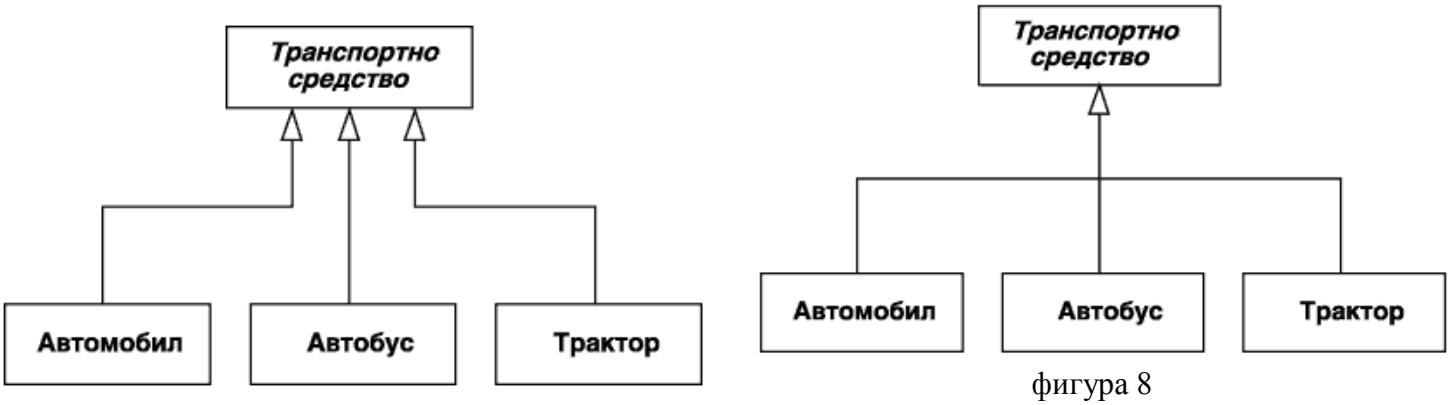


фигура 7

Ето как се представя в диаграмите това отношение: с празна стрелка, от потомъка към родителския клас.



Или

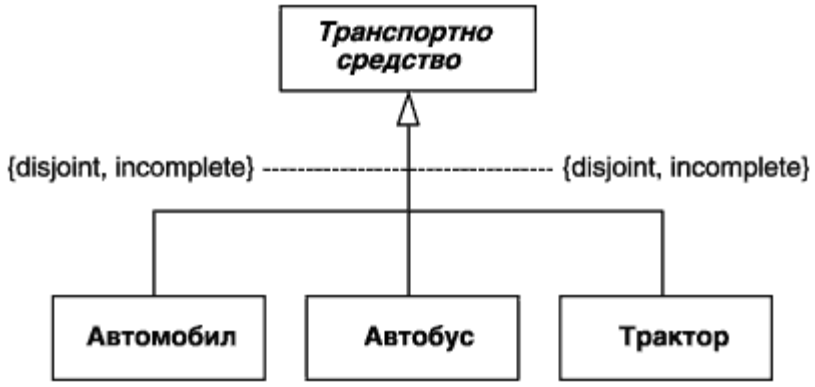


фигура 8

В допълнение към **стрелката за обобщение** може да бъде присъединен и текст, указващ специалните свойства на това отношение във вид на ограничение. Този текст се отнася към всички линии на обобщението, които отиват към класовете потомци. В качество на ограничения може да бъдат използвани следните ключови думи на езика UML:

- {complete} - означава, че в отношението обобщение са специфицирани всички класове-потомци, и класа – родител не може да има други класове-потомци.
- {incomplete} - означава, че в отношението обобщение не са специфицирани всички класове-потомци, и класа – родител може да има и други класове-потомци.
- {disjoint} - означава, че класовете потомци не могат да съдържат обекти, едновременно явяващи се екземпляри на 2 и повече класа.
- {overlapping} - случай, противоположен на предния, а именно: предполага се, че отделните екземпляри на класовете-потомци могат да принадлежат едновременно на няколко класа.

С отчитане на допълнителното използване на стандартни ограничения, диаграмата на класовете (фигура 8) може да бъде уточнена (фигура 9).



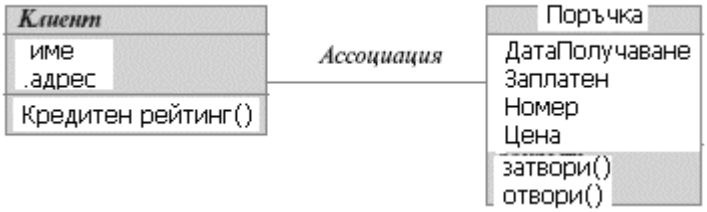
фигура 9.Един вариант на уточнено графично изображение на отношението „обобщение" на класовете, с използване на стрингове-ограничения

Асоциация (association) — това е отношение показващо, че обектите на един тип по някакъв начин са свързани с обектите на друг тип, например:

- Извикват методите си, взимно,
- Работят с обща памет,
- Обектите на единия са параметри на другия,
- Един клас има атрибут от тип – другия клас (или указател към него).

Ако между двата класа е определена асоциация, то може да се прехвърляме от обектите на единия клас към обектите на другия.

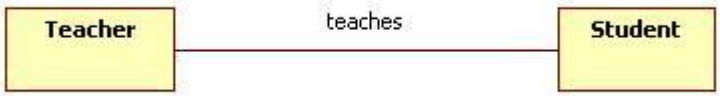
Например: "клиент" **X** може да направи „поръчка" **Y**:



фигура 10

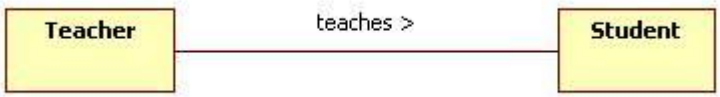
Или: **Teacher X teaches Student Y**

- Може да се представи като асоциация между клас Teacher и клас Student:



фигура 11

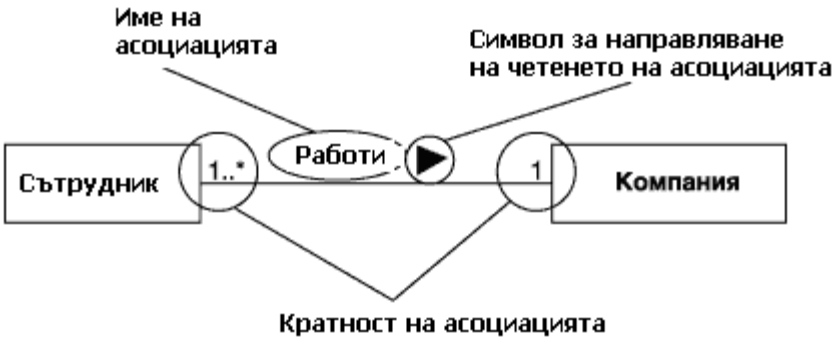
- Вижда се, че на асоциацията може да се присвои **име**, указващо семантиката на отношението.
- Някои средства за моделиране допускат и **направление** на асоциацията „teaches >":



фигура 12

- Направлението на асоциацията помага да се ориентираме в това, кой клас трябва да се указва **първи** при разглеждане на връзката и кой – **втори,** с което се разграничаваме от обратната връзка: **Student Y is taught by Teacher X**

Още един пример:



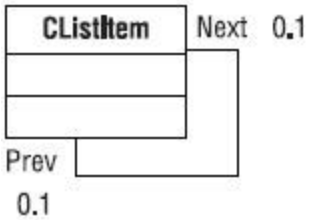
фигура 13

- **Асоциацията като връзка между класовете обезателно преминава във връзка между обектите им**. По това, тя принципно се различава от наследяването.
- Ассоциациите се съединяват с класовете чрез специални конструкции – **краища на асоциацията (association ends или endpoints)**: **end1** и **end2**. Именно в тези краища се определят различните свойства на асоциацията, такива като кратност (multiplicity), агрегиране и др. Един край може да има много свойства, но 6 са основни:
  - end1.name, end1.visibility
  - end1.isNavigable, end1.aggregation
  - end1.multiplicity, end1.participant
- Ако един преподавател трябва да знае своите студенти, например за да се генерира списък на студентите, тогава краят на асоциацията teaches към student трябва да е **плаващ, т.е: end2.isNaviagable = истина**. Графично, това се изразява чрез бодлив край на линията, указваща асоциация:

фигура 14

Внимание: Ако и двата края трябва да са плаваща, то някои инструменти за моделиране не могат да го реализират!

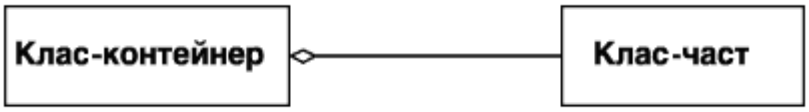
- Горният пример (както и пример 13) демонстрират още **кратност** (end1.multiplicity) в краищата на асоциацията: Всеки студент се обучава от точно 5 преподавателя, а всеки преподавател обучава един или много студенти. В една компанния работят един или много сътрудници. На всеки сътрудник съответства точно една компания.
- Краищата на асоциацията често се именоват на диаграмите (роли на асоциацията): end1.name и end2.name), за да бъде по-нагледна диаграмата, но това не е задължително. Именоването е полезно най-вече когато асоциацията е рефлексна:



фигура 15

Рефлексна асоциация: Задаване на асоциация в един клас, тоест двата края на асоциацията се отнасят за един и същи клас (рефлексна асоциация). Това означава, че обекти от дадения клас могат да се свързват с обекти от същия клас.

Агрегация (aggregation): Ако трябва да се моделира тип "част-от цяло", то се използва специален тип асоциация — **агрегиране** или **агрегация**. В такава асоциация един от класовете има по-висок ранг – това е **целият клас (агрегата)** , например такива са клас „Клас-контейнер” или клас "Системен блок". Те се състоят от няколко по-малки по ранг класа (класове-**части** — съответно клас „Клас-Част” и класове „Процесор”...”Флопи дисково у-во”):



фигура 15

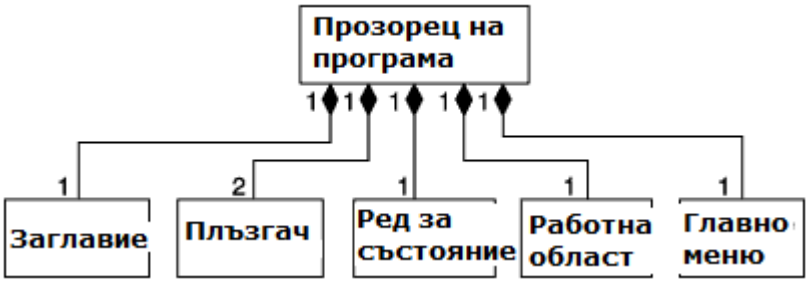
Ромбът указва класа, който представя „цялото” или Класа-агрегат. Осталите класове са неговите части.

фигура 16. Диаграма на класовете за илюстрация на отношение „агрегация” – пример „Системен Блок на ПК”

Композиция (composition): В UML се използва и по-силна разновидност на агрегация — композиция, в която класът-част може да принадлежи само на един „цял клас”. В композицията жизненият цикъл на цял клас и частите му съвпадат и всяко изтриване в цялото обезателно се отразява на частите.

Ромбът указва класа, който представя „цялото” или Клас-композиция. Осталите класове са неговите части.

Още един пример за композиция: клас-композит „Прозорец на програма”.



Диаграма на класове, илюстрираща отношение на композиция на пример за клас-композит „Прозорец на програма”.

Агрегирането се отличава принципно от наследяването, въпреки че при моделиране на предметната област с помощта на диаграми на класовете може да се забележи някакво сходство:

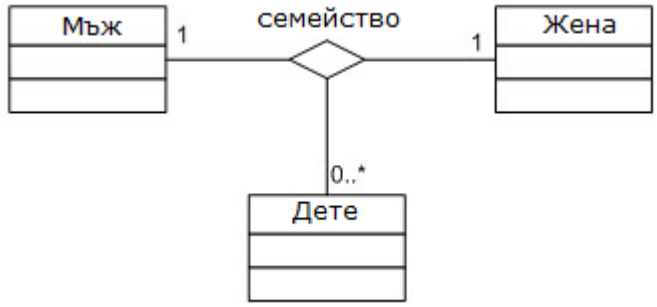
(i) Двете (агрегиране и наследяване) позволяват да се строи дървовидна йерархия от класове;
(ii) Предците, така също както и агрегируемия клас добавят функционалност родителя/агрегата;
(iii) Изображенията визуално си приличат.

Ето защо са различни:

- Наследяването е само отношение между класове и не преминава в отношение между екземплярите на класовете, за разлика от агрегирането. Обекта на породения клас, съдържа в себе си обекта на класа - родител, но никакви отношения няма между тях докато при агрегирането – втория е несамостоятелна част от първия.
- Наследяването модифицира класа-родител, непосредствено добавяйки , "вливайки" в него нови свойства (атрибути, методи, реализация на методив). Агрегирането (сумиране) не засяга агрегата (единицата, цялото), и в последния може да има собствени методи и атрибути. В случая на агрегиране частите не се разтварят в цялото, оставайки отделни части в неговия състав.

N-арни асоциации и клас-ассоциации

Досега говорихме за бинарните асоциации (binary association) – тоест такива които свързват 2 класа. По-горе бяха разгледани и рефлексивни. Заедно с тях има и n-арни ассоциации (n-ary associations), които свързват няколко класа. Например, асоциация под название "Семейство" може да свързва следните класове: "Мъж", "Жена", "Дете", показани на фигура 17:



фигура 17. Пример за N-арна асоциация

В този пример мъж и жена трябва да присъстват задължително в семейството (стойността на кратността в края на съответстващата асоциация "семейство" е 1), а децата могат да са произволно количество 0..* . Ассоциацията принципно няма атрибути, но в някои случаи те са крайно желателни. Например, ако клас „Студент” е свързан с асоциация "много към много" с клас „Дисциплина”, то тази асоциация е целесъобразно да има атрибут под название "Оценка". Това се постига, свързвайки двата класа чрез клас-асоциация (association class), в който се указват всички нужни атрибути, както е показано на фигура 18.

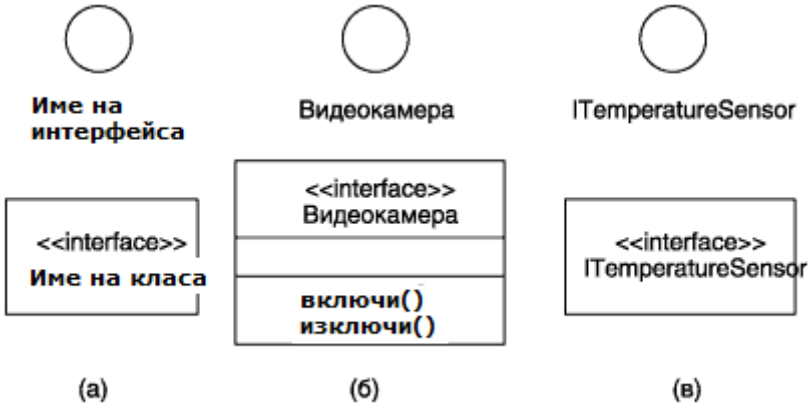
фигура 18. Пример на клас-ассоциация

Интерфейс (interface)

Интерфейс — именовано множество от операции, които характеризират поведението на отделен елемент на модела. Интерфейс, в UML контекст е специален случай на клас, който има операции, но няма атрибути. За обозначение на интерфейс се използва окръжност или стандартния начин за обозначаване на клас – правоъгълник, със стереотип <<interface>> (фигура 19).

Стереотип на клас: описва предназначението на класа. Механизмът на стереотипите се явява средство за разширение на речника на UML, с цел - решаване на специфични проблеми.

На диаграмата на случите на използване, интерфейсът се изобразява като малък кръг, под който се записва името му (фигура 19, а): съществително, което характеризира съответстващата информация или сървис, предоставен от интерфейса, например, "Датчик за температурата", "Форма за вход", "Сирена", " Видеокамера" (Фигура 19, б). С оглед на по-натъшната реализация на модела, е препоръчително името на интерфейса да се напише на английски, започвайки с буквата I, например, ITemperatureSensor, IsecureInformation (фигура 19, в).



Фигура 19. Примери за графично изобразяване на интерфейс в диаграмите на класовете

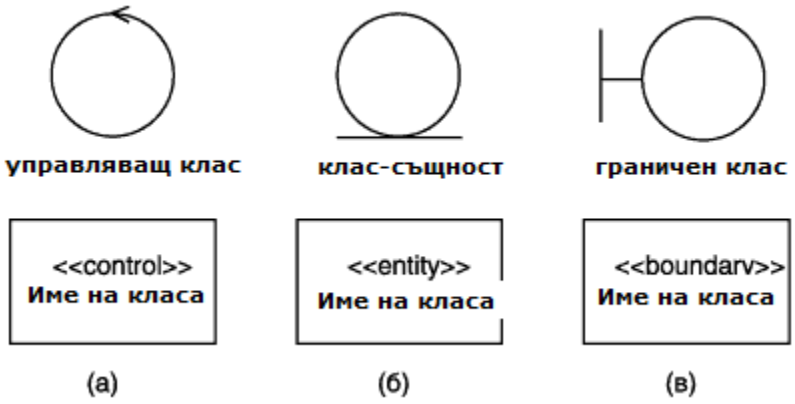
Графичното изображение на интерфейси във вид на кръг се използва и във други диаграми, например, диаграми на компоненти и на разгръщане.

Разширение на UML за построяване на модела на ПО

Едно от достойнствата на езика UML е наличието на механизъм за разширение, който позволява да се внасят допълнителни графични обозначения, ориентирани към конкретната задача на предметната област. UML съдържа 2 специални разширения: профил за процеса на разработка на ПО (The UML Profile for Software Development Processes) и профил за бизнес-моделиране (The UML Profile for Business Modeling).

В рамките на първия профил са предложени 3 специални графични примитива, които могат да бъдат използвани за уточнение на семантиката на отделните класове при построяване на различните диаграми:

- Управляващ клас (control class) — клас, отговарящ за координацията на действията на другите класове; на една диаграма може да има само един такъв клас; изобразява се се като правоъгълник на клас със стереотип <<control>> (фигура 20, а).
- Клас -същност (entity class) — пасивен клас, информацията за който се съхранява постоянно и не се унищожава при изключване на системата; като правило - този клас съответства на таблица от БД с атрибути полетата на таблицата, а операции - присъединените или съхранени процедури. Той е пасивен, може само да приема съобщения от други класове; изобразява се като правоъгълник на клас със стереотип <<entity>> (фигура 20, б).
- Граничен клас (boundary class) — клас, който се разполага на границата на системата със външната среда и непосредствено взаимодейства с актьорите но не се явява съставна част от системата; стреотип <<boundary>> (фигура 20, в).



Фигура 20. Графично изобразяване на класове за моделиране на ПО

2 Използване на StarUML за създаване на клас диаграми - един пример (Java профил). Forward и Reverse Engineering

Виж. „Програмни спецификации . Ръководство за лабораторни упражнения” тема VI.

Използвана литература:

- **Фаулър, „UML основи”**
- http://cnx.org/content/m15092/latest/
- http://www.clear.rice.edu/comp212/07-spring/labs/01.5/