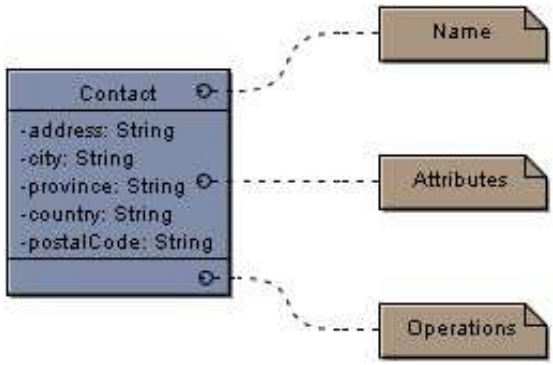
## Class Diagram

Диаграмите на класове са в основата на обектно-ориентираното проектиране. Те описват типовете на обектите в системата и статичните връзки между тях.

Показва множеството класове, интерфейси и взаимодействията между тях. Основният елемент на диаграмата на класовете е класът. В обектноориентираното програмиране класовете се използват за представяне на обектите в системата. Те описват обектите от реалния свят.

Класът *Contact* е пример на прост клас, който съхранява информация за място.




Класовете се представят като множество обекти с общи атрибути, операции, връзки. Интерфейсът е колекция от операции, задаващи поведение.

Класът е разделен на три секции:

### Име на класа

**Атрибути на класа** – характеристиките на класа. В примера класа *Contact* съдържа атрибутите address, city, province, country и postal code.

Формат на атрибутите:

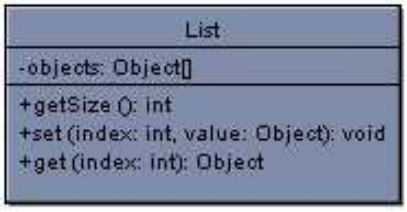
*visibility name: type = defaultValue*

Видимост на атрибутите:

- **-** Private. Атрибутите са видими само за операциите на класа
- **+** Public. Атрибутите са видими извйн класа и са част от интерфейса
- **#** Protected. Атрибутите са видими само за класовете, наследяващи текущия
- **~** Package

В обектно-ориентираното програмиране е препоръчително да се съхраняват атрибутите като private и да се използват методите за контролиране на достъпа до данните.

**Операции на класа –** списъка на операциите на класа представя функции или задачи, които могат да бъдат изпълнени с данните на класа.



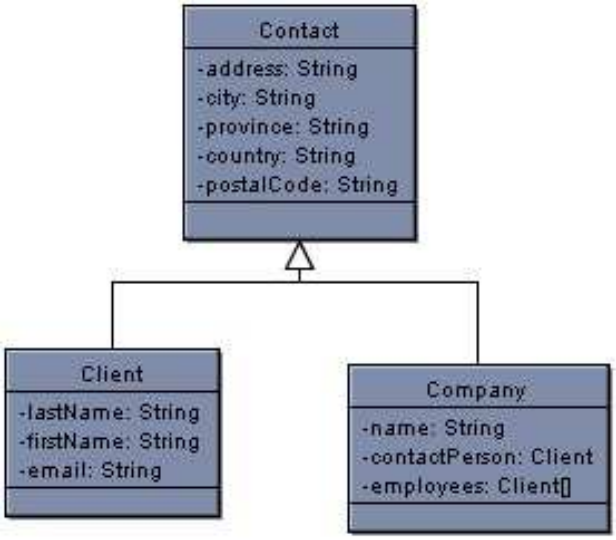

В класа *List* има един атрибут (private array of Objects) и три операции

Формата за операциите на клас е:

*visibility name (parameters): type*

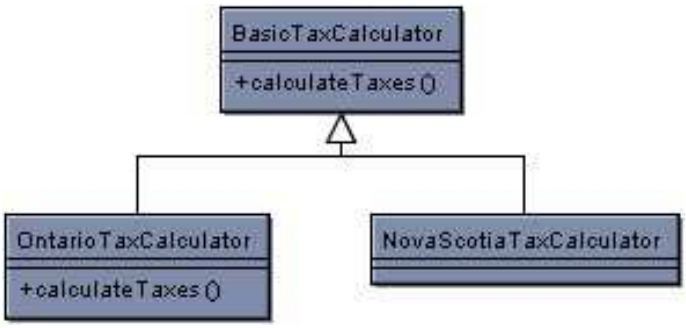
## Generalization

**Generalization** връзката между два класа се използва, за да покаже, че един клас включва всички атрибути и операции на друг клас, но добавя към тях допълнителни атрибути и операции.



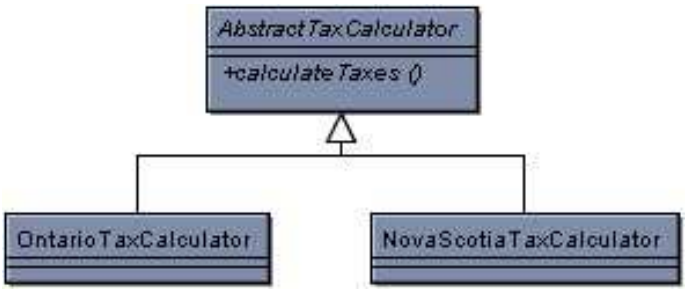

На диаграмата е показан класа *Contact* с два подкласа. Може да се каже, че класовете *Client* и *Company* **inherit**, **generalize** или **extend** *Contact*. За *Client* и *Company* всички атрибути на *Contact* (address, city, и т.н.) съществуват, но са добавени и допълнителни. Класа *Contact* може да се каже, че е **superclass** на *Client* и *Company*.

Когато се използва връзка generalization, класовете наследници могат да предефинират операциите на базовия клас – операции, които са дефинирани в базовия клас, но дефинират нова реализация.




Например *OntarioTaxCalculator* предефинира метода на BasicTaxCalculator. Всъщност кода е различен, но метода се извиква по същия начин.

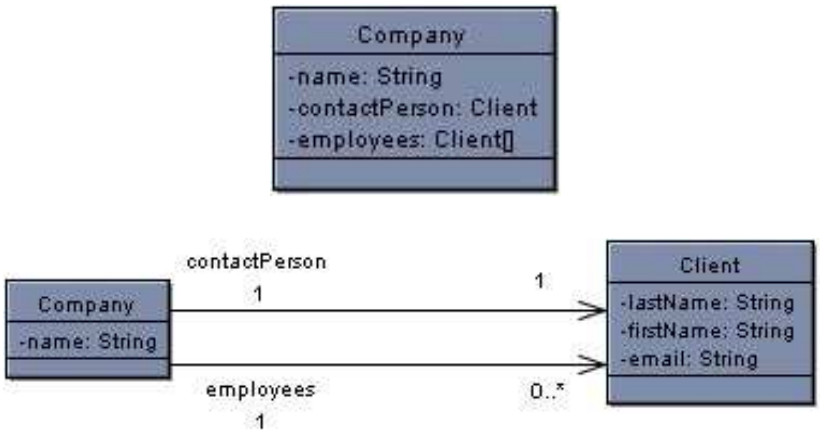
Възможно е да се дефинира метод в суперкласа като абстрактен (**abstract**). Ако класа има абстрактни операции, то и класа е абстрактен. Абстрактните методи и класове се означават като курсивни (italic). Не всички операции на абстрактния клас са абстрактни.




Абстрактната операция *calculateTaxes* в *AbstractTaxCalculator* трябва да бъде имплементирана в класовете наследници OntarioTaxCalculator и NovaScotiaTaxCalculator.

### Associations

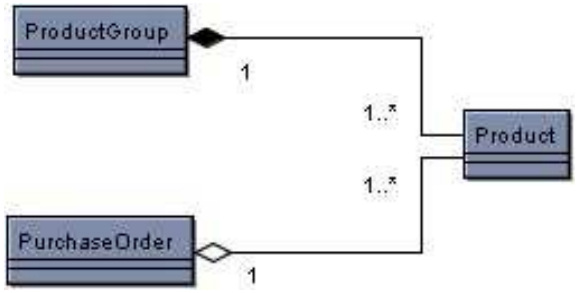
Класовете могат също да съдържат връзки към друг клас. Класът *Company* class има два атрибута, които са свързани с класа Client.




Първата асоциация представя атрибута *contactPerson*e. Съществува един *contactPerson*e в един *Company*. Това означава, че всяка компания има само едно лице за контакти и за всяко лице за контакти има само една компания. Втората асоциация описва съществуването на нито един или много служители в компанията.

| | |
|---|---|
| **0** | 0 |
| **1** | 1 |
| **1..*** | 1 или много |
| **1..2, 10..*** | 1, 2 или 10 и нагоре, но не от 3 до 9 |

### Aggregation and Composition




Примера показва асоциациите **aggregation** и **composition**.

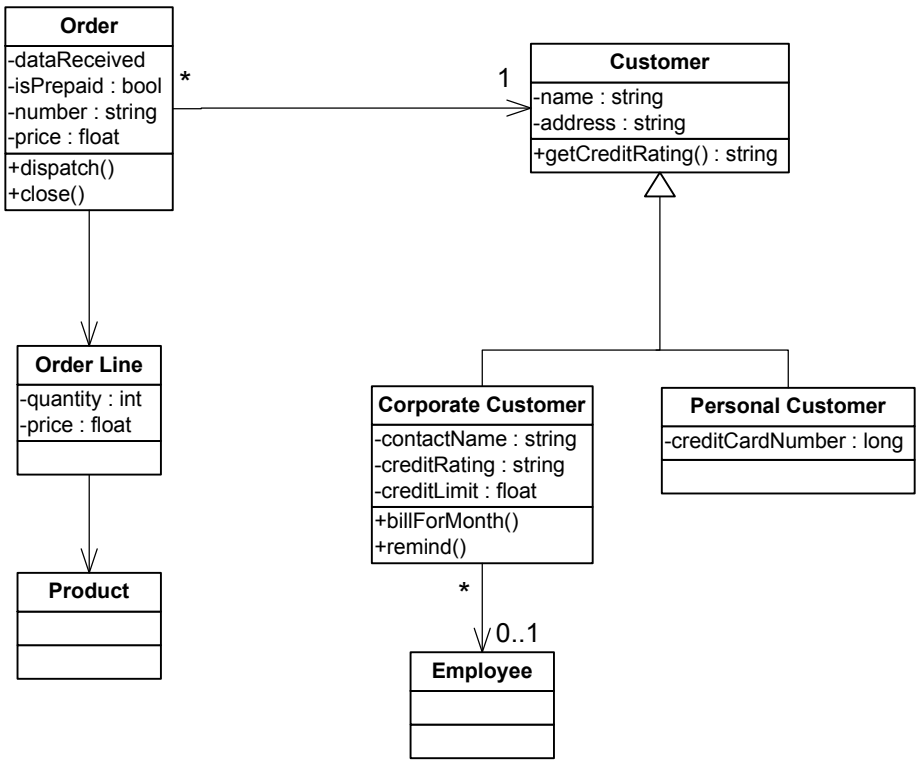
*ProductGroup* е **съставен от** *Products*. Ако *ProductGroup* се унищожи, се унищожава *и Products*

*PurchaseOrder* е съвкупност от *Products*.  Ако *PurchaseOrder* се унищожи, *Products* съществува.

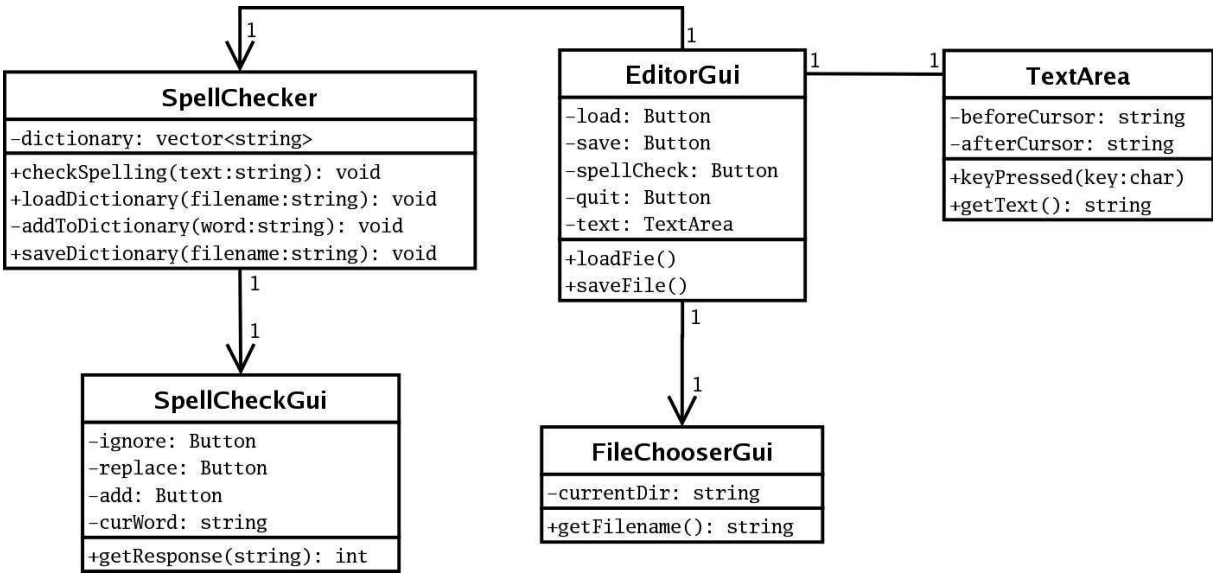
### Зависимост (Dependencies)

Съществува зависимост между два елемента, ако промяната в единия има ефект върху другия. Например, ако един клас извиква операция на друг клас, тогава съществува зависимост между двата класа. Цел при проектирането на системата е да се сведат до минимум зависимостите

## **Пример** за диаграма на класове



**Пример** за диаграма на класовете на текстов редактор:




### Пакети (Packages)

Пакетите позволяват групиране на обектите. В много обектноориентирани езици (като Java), пакетите се използват за представяне на класовете и интерфейсите.


Показва как системата се разделя на логически групи, чрез изобразяване на зависимостите между тях. Името на пакета може да е просто или уточнено, ако той е вложен в друг пакет. Пакетът може да съдържа classes, interfaces, components, nodes, collaborations, use cases, diagrams и други packages. Дефинира собствено пространство на имената.

Видимостта на елементите в пакета се контролира както при класовете:

- protected се означава с #. Елементите protected са видими само за пакети, наследяващи текущия.
- Private се означава с –. Елементите private са невидими извън пакета.
- Public се означава с +. Елементите public се включват в интерфейсната част на пакета.

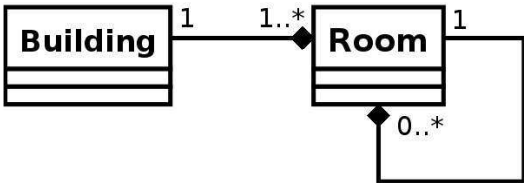
Пакетните диаграми съдържат множество пакети и връзките между тях.

Различаваме 3 вида диаграми на пакети:

- Class Package Diagrams
- Data Package Diagrams
- Use Case Package Diagrams

### Object Diagrams

- Object Diagrams показват инстанциите на класа и техните връзки
- Използват се за изясняване на комплексни връзки




**Пример** за диаграма на обекти, как са свързани инстанциите на класовете.