Обектно-ориентирано програмиране (ECESIS)

(Rev: 1.3)

Любомир Чорбаджиев[1]

lchorbadjiev@elsys-bg.org

[1]Технологическо училище "Електронни системи"
Технически университет, София

31 октомври 2007 г.

Съдържание

Принципи на обектно-ориентираното програмиране

- Обекти
- Класове, екземпляри (инстанции), полета и методи
- Капсулация
- Полиморфизъм
- Наследяване
- Динамично свързване

Какво е обектно-ориентирана технология?

- Обектно-ориентираната технология позволява софтуера да бъде моделиран по подобие на реалността.
  - Това се постига посредством моделиране на обекти и взаимодействието между тях.
- Обектно-ориентираната технология е начин да се мисли за софтуера в термините на компоненти, които могат да се използват многократно
  - Всеки обект сам по себе се е компонент, който може да си използва в различни контексти

Предимства на обектно-ориентираните системи

- По-голяма гъвкавост
- Повторна използваемост на обектите
- По-високо качество на продукта
- По-ниска цена за поддръжка и съпровождане на софтуера

Разработване на обектно ориентиран софтуера

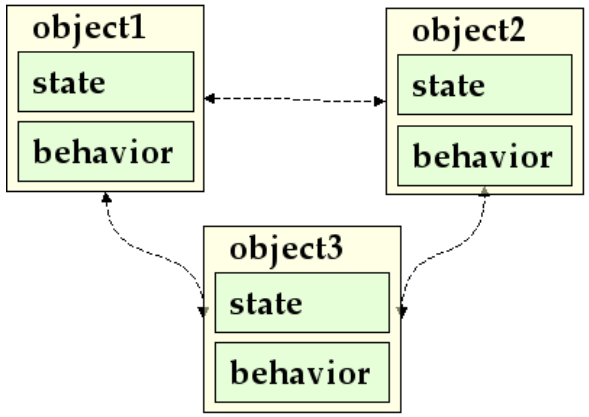
- Софтуерът се моделира чрез обекти
  - Обектите в програмата моделират реалните обекти
- Всеки софтуерен обект обединява в себе си:
  - Състояние, т.е. данни
  - Поведение, т.е. методи, функции

Обектно ориентиран софтуер

- При разработването на обектно-ориентиран софтуер трябва да се идентифицират:
  - Обектите
  - Характеристиките на отделните обекти
  - Връзките и взаимоотношенията между обектите
- Обектите си взаимодействат като си изпращат съобщения
  - Взаимодействащите помежду си обекти изграждат една обектно-ориентирана система

Обекти и слабо обвързване между частите на програмата

- Ако се промени вътрешното представяне на данните в обекта, то това няма да доведе до промяна на външното поведение на обекта
  - Външният интерфейс на обекта се запазва
  - Това спомага за слабо обвързване между обектите

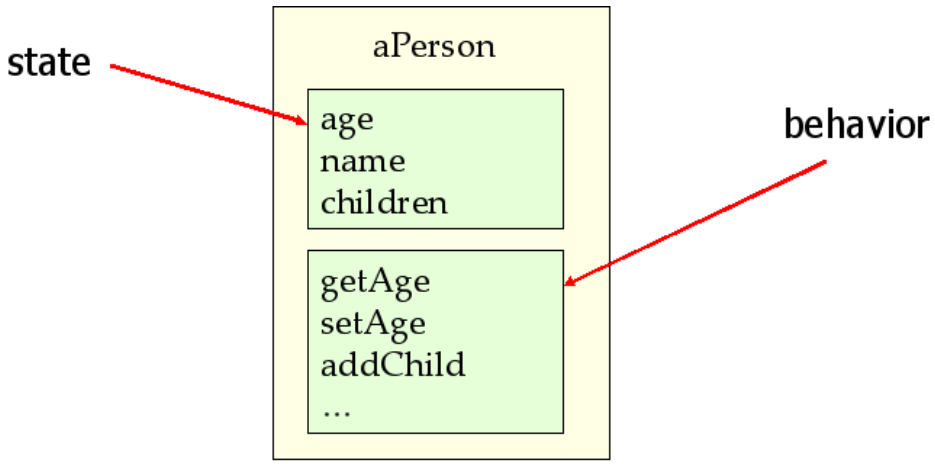


Обекти

- Всеки обект има:
  - Състояние
  - Поведение
- Състоянието е описание на данните на обекта — какво обекта знае, какво съдържа
- Поведението е какво обекта може да прави

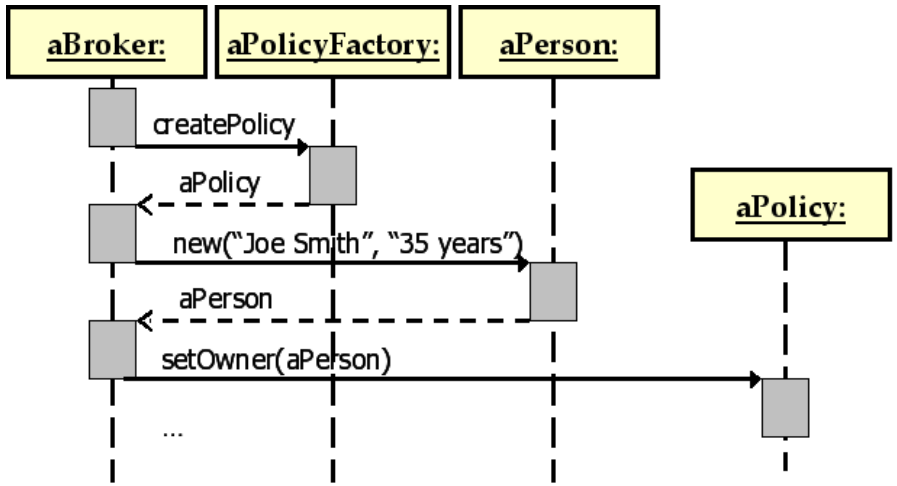
Състояние и поведение на обектите

- Обектът `Person`:
  - Състояние: `age`, `name`, `children`
  - Поведение: `addChild`, `getAge`, `setAge`



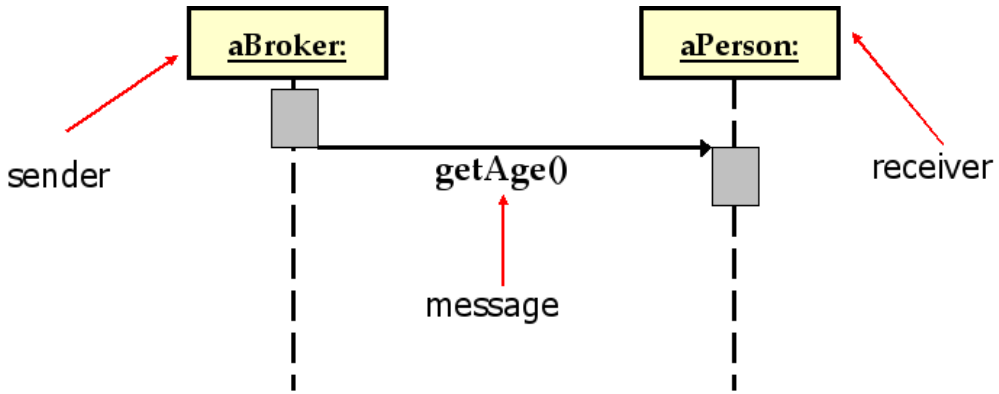
Взаимодействие между обекти

- Обектите си взаимодействат като си изпращат съобщения
- Обектите и взаимодействията между тях изграждат обектно-ориентираната система



Съобщения

- Има два основни участника при изпращането на съобщения:
  - Обект, който изпраща съобщението
  - Обект, който получава съобщението — получател
- Съобщението може да има аргументи



Методи

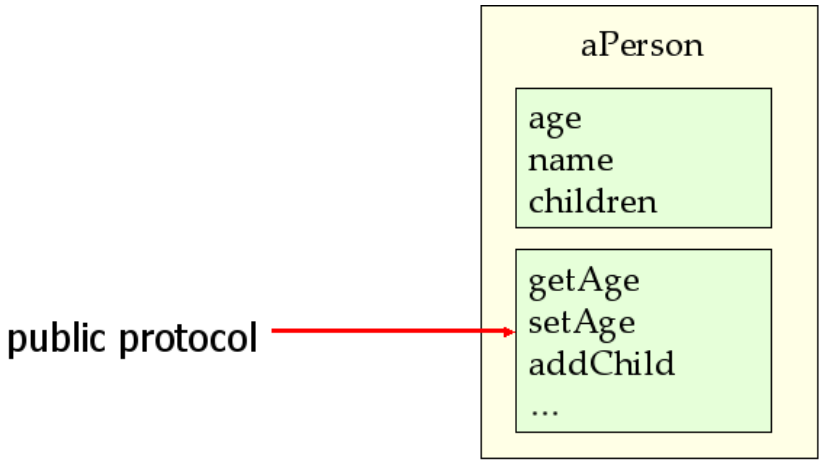
- Методът е конкретна имлементация (реализация) на съобщение.
- Когато едно съобщение е изпратено до получателя:
  - Намира се метод съобразно типа на обекта получател и сигнатурата на метода.
  - Изпълнява се намерения метод.
- Методът представлява отговора на обекта на полученото съобщение

Сигнатура на метода

- Сигнатуратата на метода представлява уникален идентификатор на метода.
- Сигнатурата се използва за да се различат методи, които имат едно и също име.
- Сигнатурата се състои от:
  - Името на метода
  - Броя на параметрите
  - Типа на параметрите

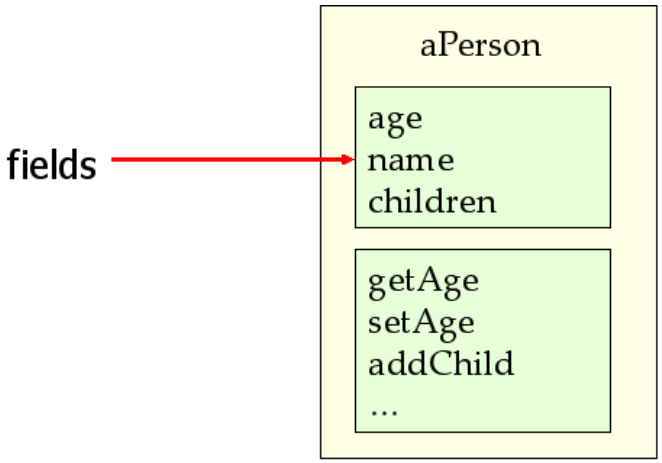
Публичен протокол на обекта

- Публичният протокол е набора от съобщения, които могат да се изпратят на даден обект.
- Той не включва съобщенията, които обекта може да изпрати до себе си.
  - Такива съобщения се наричат скрити (private).



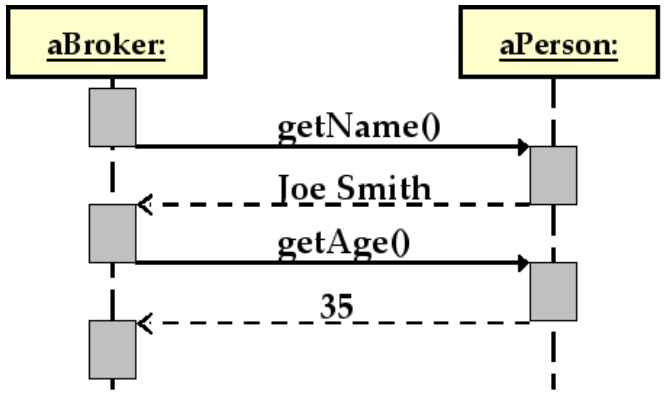
Полета

- Полетата описват на характеристиките на обекта.
- Полетата са също така известни като атрибути или променливи на екземпляра.



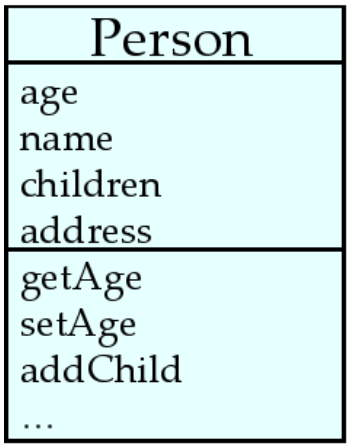
Принципи на обектно-ориентираното програмиране: Капсулация

- Обектите крият вътрешната реализация (имплементацията) на методите чрез техният публичен протокол.
  - Вътрешната реализация на обекта е достъпна само за този обект.
- Капсулацията е известна още като скриване на реализацията.



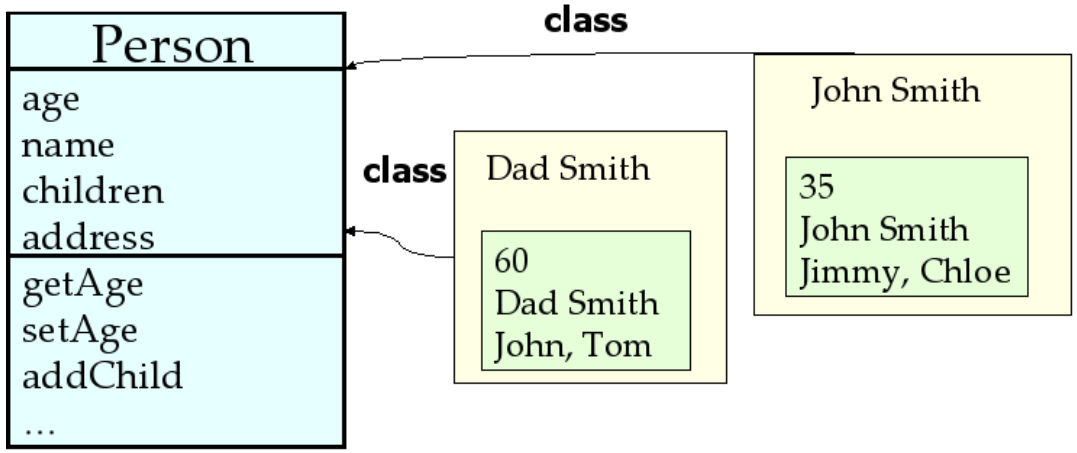
Класове

- Класовете са:
  - Фабрики за създаване на обекти.
  - Шаблони за създаване на обекти от един и същ вид, които описват тяхното състояние и поведение.
  - Хранилище за кода на обектите.
- Класовете дефинират обектите (дефинирайки тяхното състояние и поведение) и техния тип.



Екземпляри

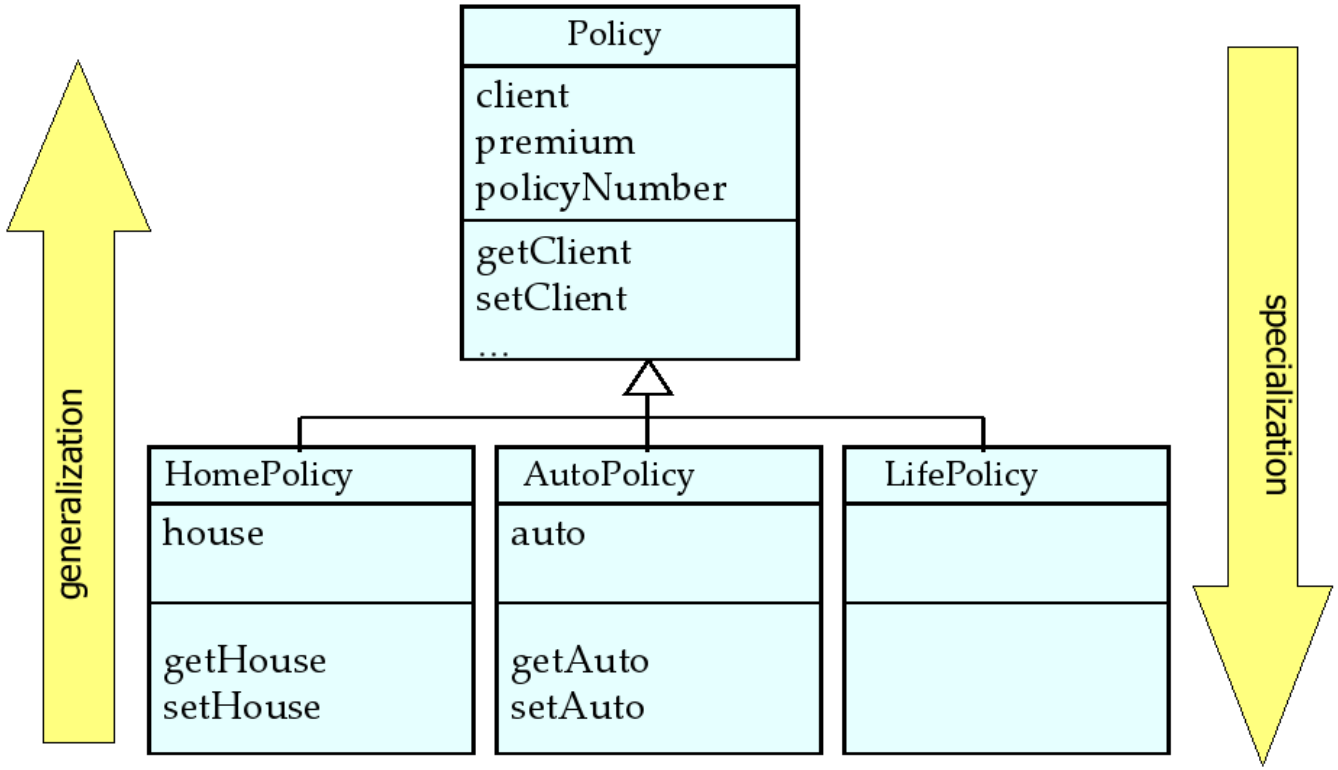
- Всеки обект е екзепляр (инстанция) на даден клас.
- Всички екземпляри (инстанции) на даден клас имат един и същ протокол.
  - Те имат същите полета и методи, които са дефинирани от класа.



Принципи на обектно-ориентираното програмиране: наследяване

- Някои класове могат да имат общи черти.
  - Например: класовете `HomePolicy`, `AutoPolicy`, `LifePolicy` могат да имат едно и също състояние и поведение.
- Вместо общите характеристики да се повтарят във всеки клас, може тези общи черти да се извлекат на едно място.
  - Тези общи черти могат да се извлекат в общ родителски клас.
  - Всеки наследник на този суперклас ще наследи състоянието и поведението на родителския клас.

Специализация и обобщаване



Наследяване

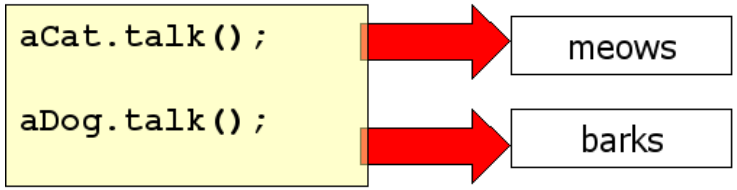
- Наследяването помага на моделирането на реалния свят.
  - Някои обекти могат да се разглеждат като специализация на други обекти.
- Наследяването подпомага повторното използване на кода.
  - Едни и същи данни и поведение се споделят между обекти от различен тип.
  - Новите данни и новото поведение, което е общо за тези обекти може да бъде добавено по-лесно.

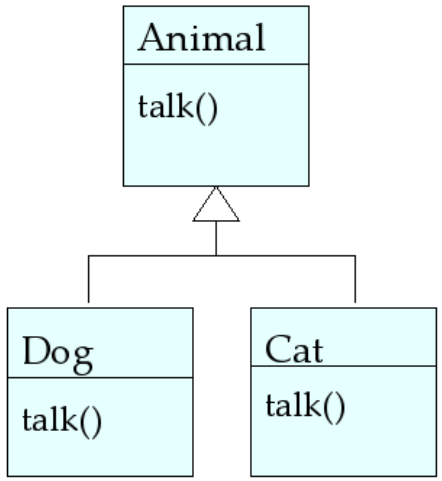
Принципи на обектно-ориентираното програмиране: полиморфизъм

- Полиморфизъм:
  - Различните обекти отговарят по различен начин на едно и също съобщение.
  - Например, ако накарате едно куче да говори, то лае, а котка мяука.
- Често полиморфизмът е реализиран чрез предефиниране на метод.
  - Предефиниране означава, че класът наследник може да реализира същия метод, като на родителския клас, но с различно поведение.

Пример за предефиниране на метод

- Да резгледаме класът `Animal`:
  - Класовете `Dob` и `Cat` са наследници на класа `Animal`
  - Всички обекти `Animal` трябва да знаят как да реагират на съобщението `talk`





Динамично свързване

- Динамичното свързване означава, че изборът на метод се извършва по време на изпълнение на програмата.
  - Това е свързване по време на изпълнение на метода, който е извикан в съобщението с метода, който имплементира това съобщение.
  - Например: