Обектно-ориентиран дизайн и UML

Обектно-ориентиран дизайн и UML

Съдържание:

- Основни етапи от разработката на софтуер
- Въведение в обектно-ориентирания дизайн
- Въведение в UML

Основни етапи от разработката на софтуер

Съдържание:

- Процесът на разработка на софтуер
- Анализиране на изискванията
- Дизайн на системата
- Имплементация
- Тестване

Разработка на софтуер

Разработката на софтуер включва 4 основни дейности:

- Анализ на изискванията
- Дизайн (проектиране)
- Имплементация
- Тестване

Тези дейности не следват стриктно едно след друго

- Те се засичат и си взаимодействат

Изисквания

Изискванията (software requirements) описват задачите, които системата трябва да изпълнява

- Отговарят на въпроса "какво?", не "как?"
- Функционални и нефункционални изисквания

Започва се от начална визия и изисквания и те постепенно се разширяват и уточняват

Изисквания

Много е трудно изискванията да се опишат и документират изчерпателно и еднозначно

Добрите изисквания спестяват време и пари през целия проект

Изискванията винаги се променят по време на работа по проекта

Добрата спецификация минимизира бъдещите промени

Дизайн

Софтуерният дизайн описва как системата ще изпълни изискванията

Архитектурният дизайн описва:

- Как задачата ще бъде разбита на подзадачи (модули) за да се управлява сложността
- Отговорностите на отделните модули
- Взаимодействията между модулите
- Интерфейсите за връзка

Дизайн

Детайлен дизайн

- Описва структурата и организацията на всеки от модулите в детайли

Обектно-ориентиран дизайн

- Описва кои класове и обекти са необходими, техните отговорности и как те си взаимодействат

Вътрешният дизайн на класовете описва как работи всеки клас

- Методите, отговорностите на всеки метод, използваните алгоритми

Имплементация

Имплементацията е процесът на писане на програмния код

- Кодът стриктно следва дизайна

Неопитните разработчици смятат, че писането на кода е същината при създаването на софтуера

Всички по-важни решения се взимат по време на анализиране на изискванията и по време на дизайна

Тестване и дебъгване

Тестването проверява дали даденото решение изпълнява всички изисквания

Целта на тестването е да намери дефекти (грешки)

- Black-box и white-box тестове
- Unit тестове, системни тестове

Дебъгването намира причината за даден дефект и поправя дефекта

Въведение в обектно-ориентирания дизайн

Съдържание:

- Какво е обектно-ориентиран дизайн?
- Идентификация на класовете
- Характеристики на класовете
- Дефиниране на отговорностите
- Cohesion и coupling

Обектно-ориентиран дизайн

Методология за моделиране на системата с класове от ООП

- Декомпозиция на изискванията
- Моделиране на изискванията
- Идентификация на класовете
  - Атрибути
  - Операции
- Дефиниране на връзки между класовете

Обектно-ориентиран дизайн

Инструментът на обектно-ориентирания дизайн: Unified Modeling Language (UML)

- Use Case диаграми
- Клас диаграми
- Диаграми на взаимодействията
  - Sequence диаграми
  - Activity диаграми

Обектно-ориентиран дизайн

Основна цел на ОО дизайн е да идентифицира класовете и обектите в системата

Потенциалните класове са предметите и явленията от реалния свят, описани в изискванията

- Съществителните от спецификацията са класове и техни характеристики
- Глаголите от спецификацията са операциите в класовете

Класове и обекти

Класовете представят множество обекти с еднакви характеристики и поведение

Имената на класовете трябва да са съществителни в единствено число

- Примери: `Student`, `Message`, `Account`

Можем да създаваме много конкретни инстанции от всеки клас

Идентифициране на класовете

Понякога е трудно да се прецени дали някоя същност от реалния свят трябва да бъде клас

- Например адресът може да е клас `Address` или символен низ

Колкото по-добре проучим проблема, толкова по-лесно ще решим кое трябва да е клас

Когато даден клас стане прекалено голям и сложен, той трябва да се декомпозира на няколко по-малки класове

Характеристики на класовете

Класовете, които съответстват на същност от реалния свят имат атрибути (характеристики)

- Например: класът `Student` има име, учебно заведение и списък от курсове

Не всички характеристики са важни за софтуерната система

- Например: за класа `Student` цвета на очите е несъществена характеристика
- Само съществените характеристики трябва да бъдат моделирани

Отговорности на класовете

Всеки клас трябва да има ясно дефинирани отговорности

- Какви обекти или процеси от реалния свят представя?
- Какви задачи изпълнява?

Всяко действие в програмата се извършва от един или няколко метода от един или няколко класа

- Действията се моделират с операции (методи)

Отговорности на класовете

За имената на методите се използват глагол + съществително

- Примери: `PrintReport()`, `ConnectToDatabase()`

Не може веднага да се дефинират всички методи на даден клас

- Дефинираме първо най-важните методи – тези, които реализират основните отговорности на класа
- С времето се появяват още допълнителни методи

Cohesion и coupling

Фундаментални принципи при софтуерния дизайн (и при обектно-ориентирания дизайн)

- Strong cohesion
  - Силна свързаност на отговорностите
  - Класът решава една ясно дефинирана задача (само една!)
- Loose coupling
  - Слаба свързаност с другите класове
  - Независимост от средата

Въведение в UML

Съдържание:

- Какво е UML?
- Use case диаграми
- Class диаграми
- Sequence диаграми
- Activity диаграми

Какво е UML?

UML (Unified Modeling Language)

- Утвърден стандарт за моделиране на софтуерни системи
- Универсална нотация за графично изобразяване на модели и изгледи

Поддържа се от много инструменти

- Together
- Rational Rose
- Microsoft Visio
- Poseidon

UML: принципът 80/20

Можете да моделирате 80% от задачите с 20% от средствата на UML

Нека разгледаме тези 20%

UML: видове диаграми

Use case диаграми (случаи на употреба)

- Описват поведението на системата от гледна точка на потребителя
- Какво може да правят различните видове потребители

Class диаграми

- Описват класовете (атрибути и методи) и връзките между тях (асоциации, наследяване, ...)

UML: видове диаграми

Sequence диаграми

- Описват схематично на взаимодействието между потребителите и системата
- Отделните действия са подредени във времето

Statechart диаграми

- Описват възможните състояния на даден процес и възможните преходи между тях
- Представляват краен автомат

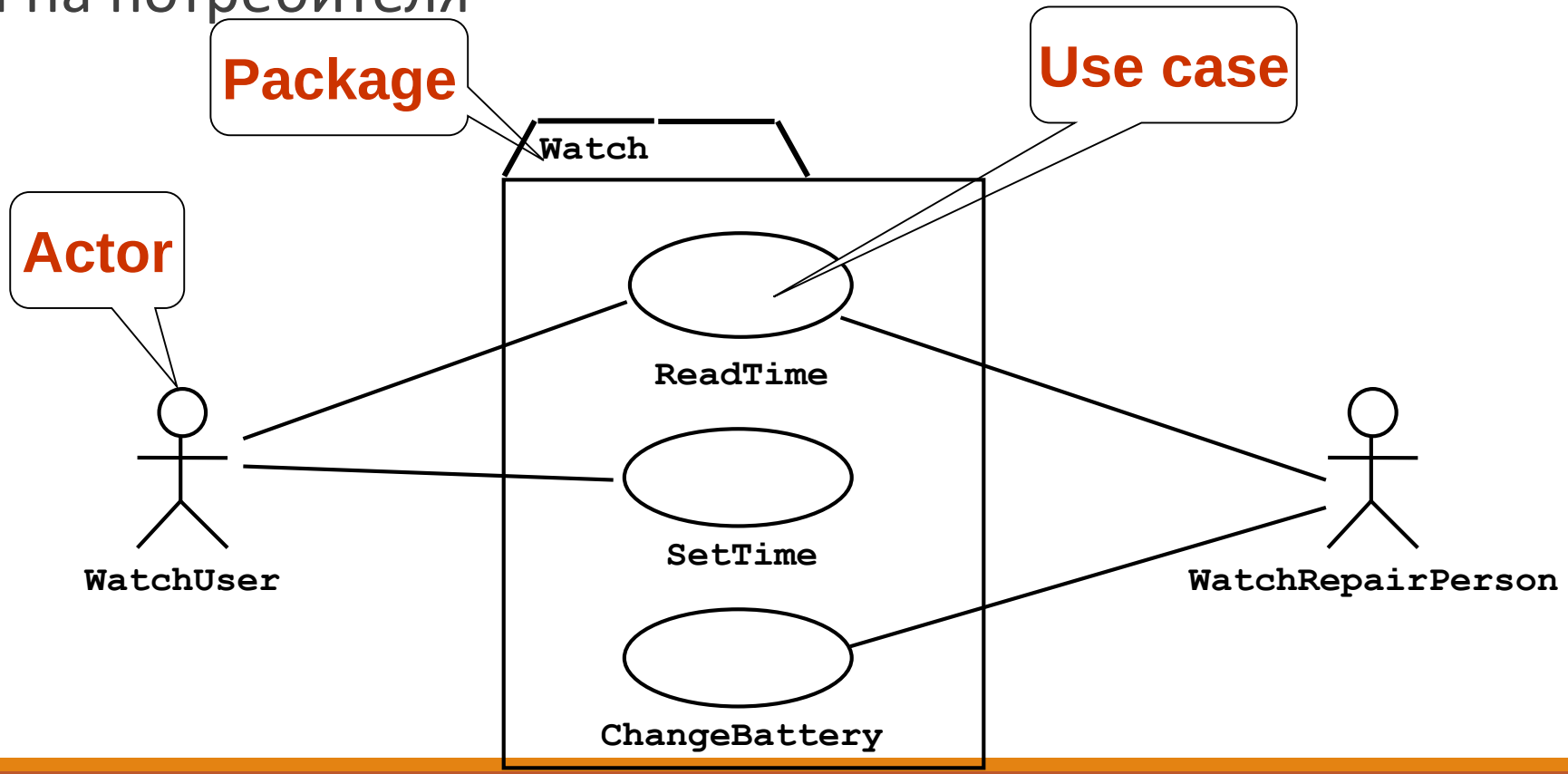
UML: видове диаграми

Activity диаграми

- Описват потока на работните процеси (workflow)
- Представляват нещо като блок-схеми

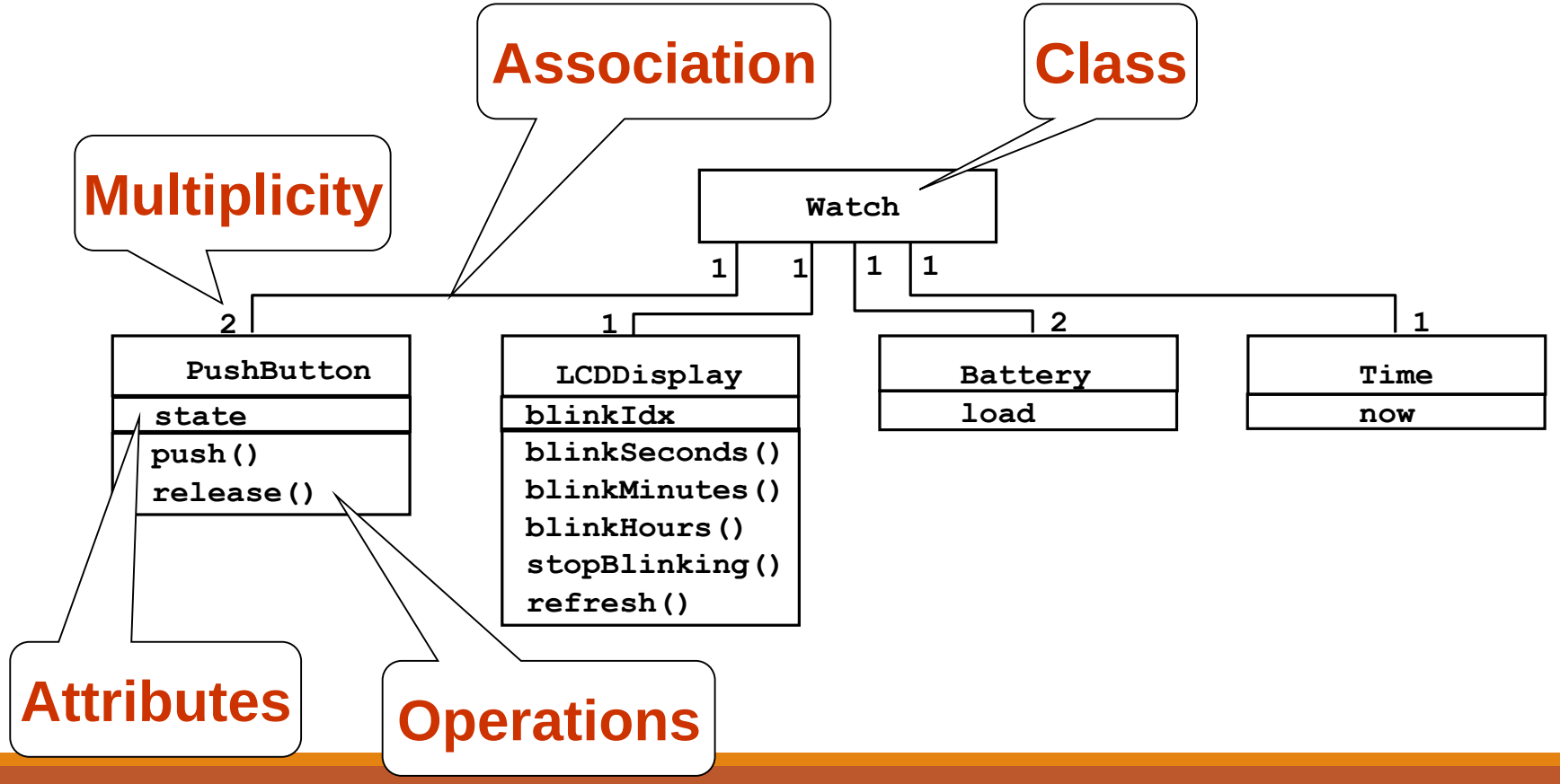
UML: Use Case диаграми

Описват функционалността на системата от гледна точка на потребителя

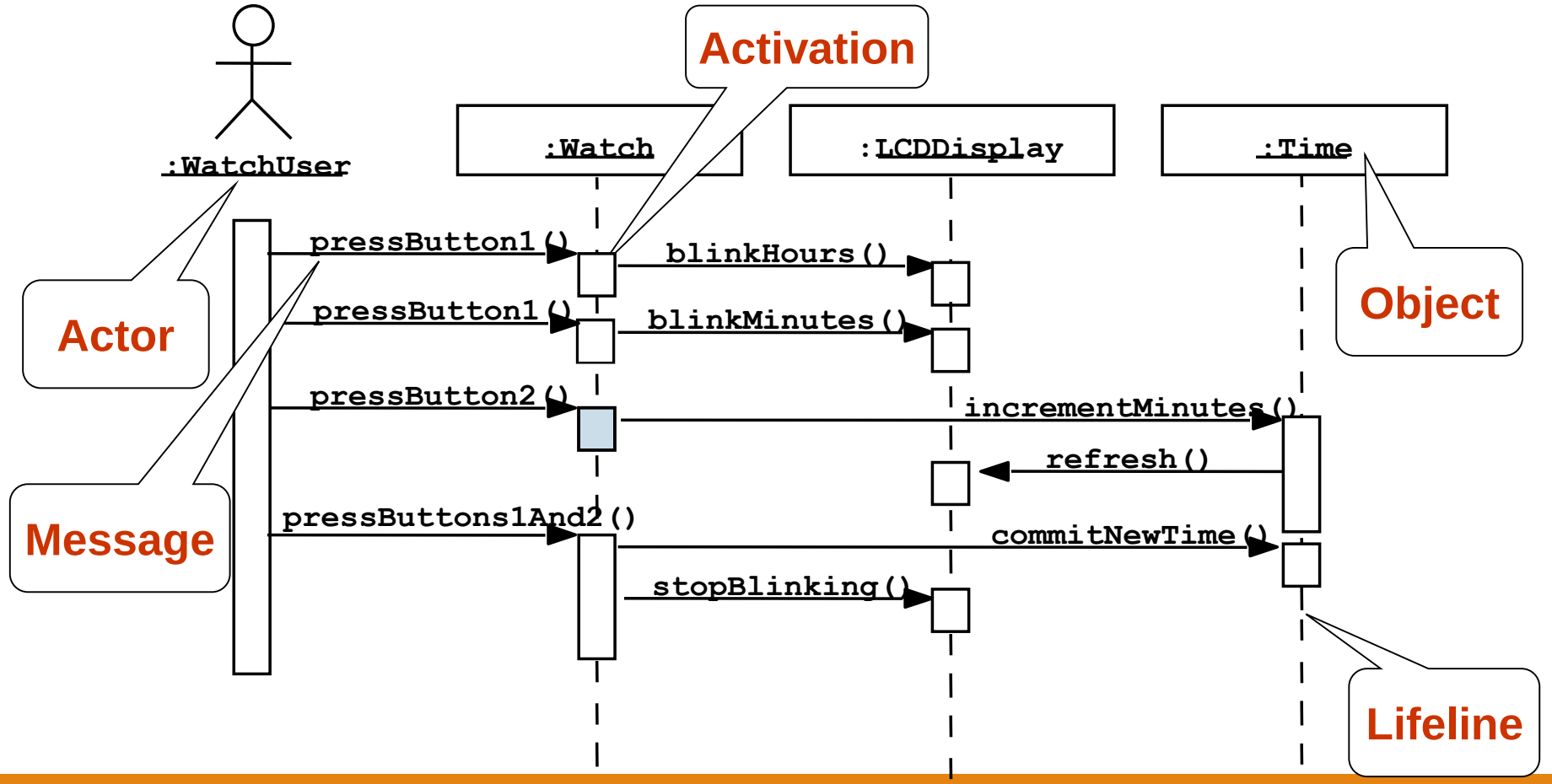


UML: Class диаграми

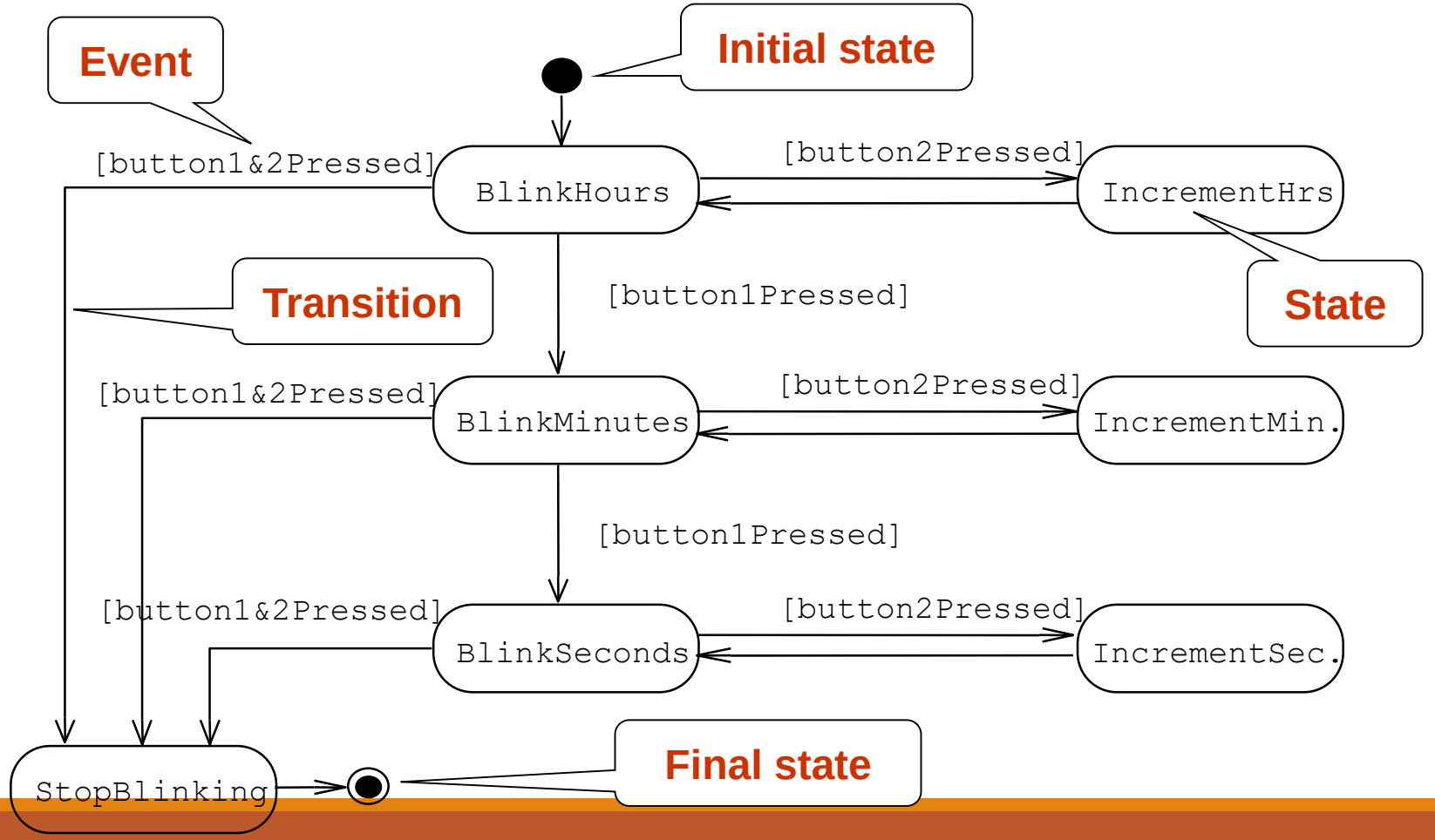
Пример: часовник



UML: Sequence диаграми



UML: Statechart диаграми



Други UML нотации

- UML предоставя и други видове диаграми (нотации)
- Имплементационни диаграми
  - Component диаграми
    - Моделират отделен компонент
  - Deployment диаграми
    - Моделират средата, в която ще се инсталира системата
- Език за ограничения на обектите
  - Дефинира входни и изходни условия

Клас диаграми

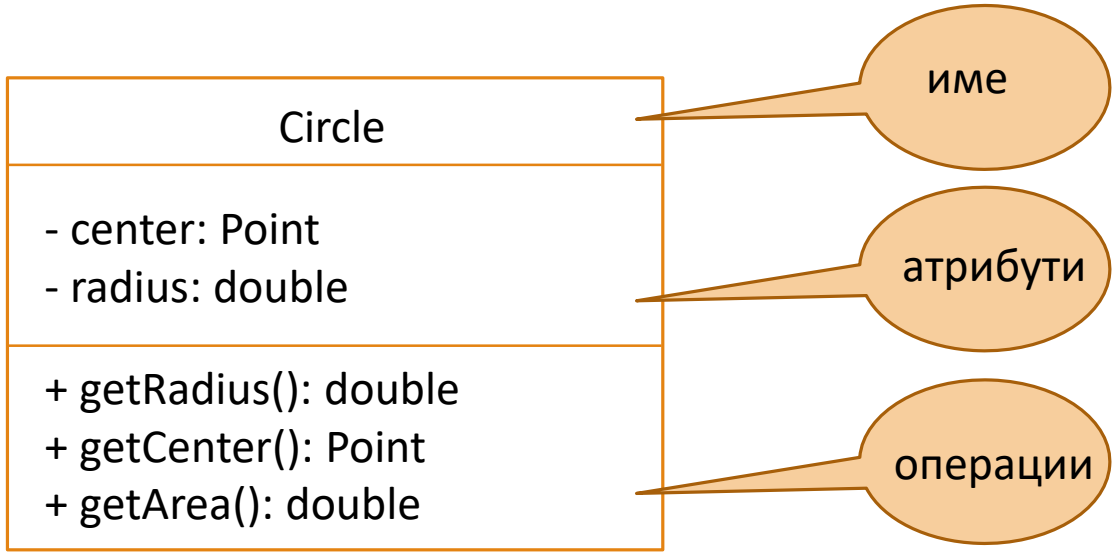
Описват структурата на системата

Използват се:

- По време на анализиране на изискванията за моделиране на обектите от реалния свят
- По време на дизайна за моделиране на подсистемите

Дефинират класове и връзки между тях

Класове



Класът е същност от реалния свят

Представя се като кутия с 3 секции

Има *име*, състояние *(атрибути, член-данни)* и поведение *(операции, методи)*

Видимост на класовете

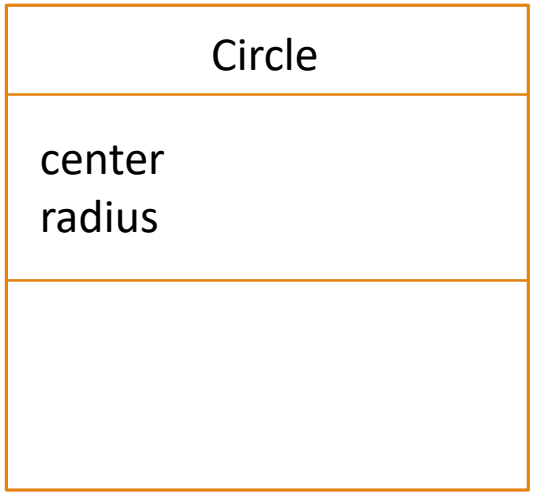
Знаците **+**, **-** и **#** пред имената на атрибутите и операциите обозначават тяхната видимост

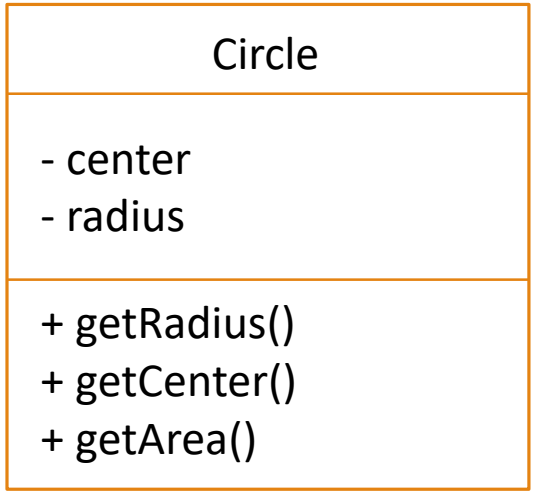
- + обозначава **public** атрибут или операция
- - обозначава **private** атрибут или операция
- # обозначава **protected** атрибут или операция

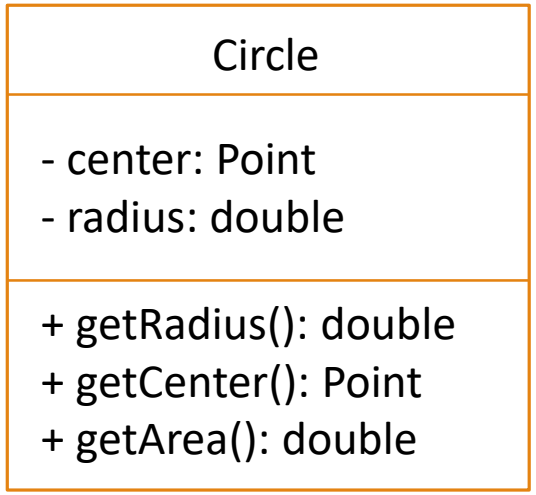
Перспективи

Една диаграма може да се разглежда от различни перспективи:

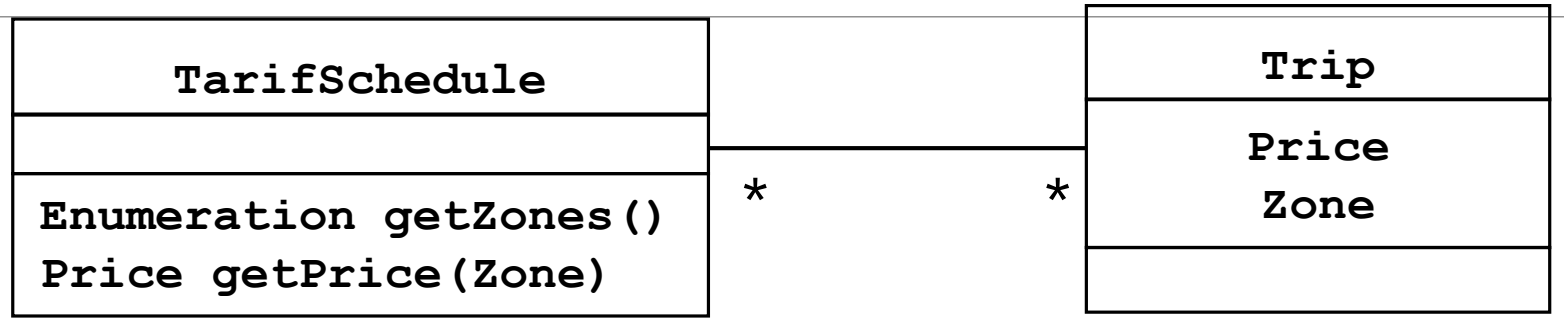
- **Conceptual**: концептуална – представя понятията в областта
- **Specification**: спецификация - фокусира върху интерфейсите на абстрактните типове данни
- **Implementation**: имплементация – описва как класовете ще реализират интерфейсите







Асоциации



Асоциациите представляват връзки между класовете

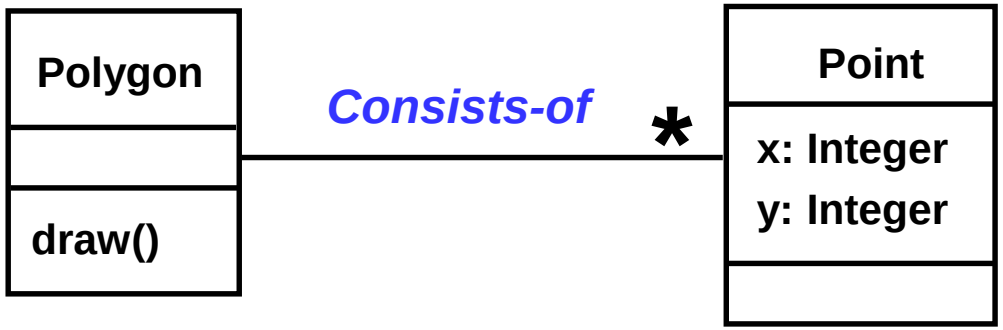
- Моделират взаимоотношения

Могат да дефинират множественост (1 към 1, 1 към много, много към 1, 1 към 2, ...)

Асоциации 1-към-1 и 1-към-много

Асоциация 1-към-1:

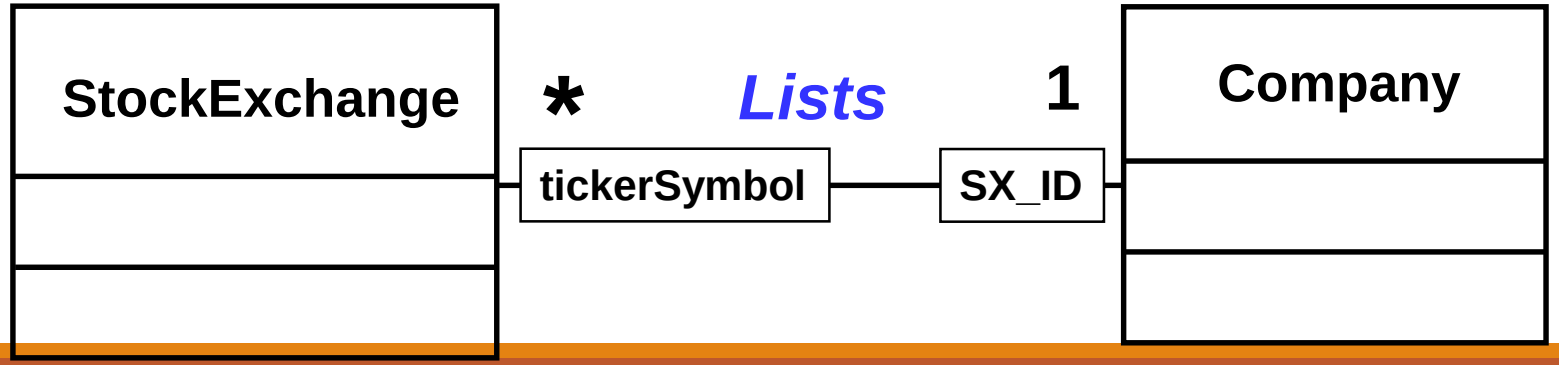
Асоциация 1-към-много:



Асоциации много-към-много

Асоциация много-към-много:

Асоциация много-към-много по атрибут:



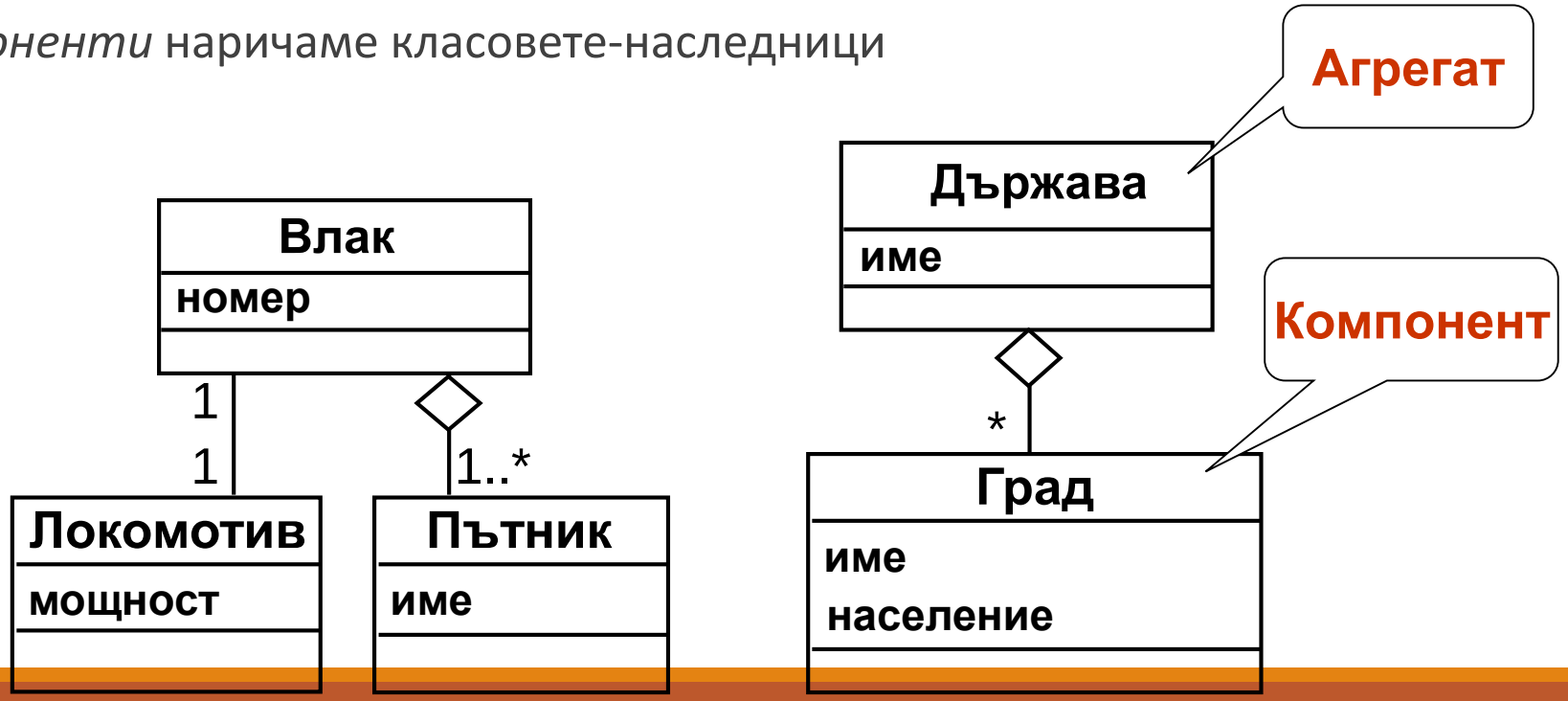
Агрегация

Агрегацията е специален вид асоциация

Моделира връзката "цяло / част"

Агрегат наричаме родителския клас

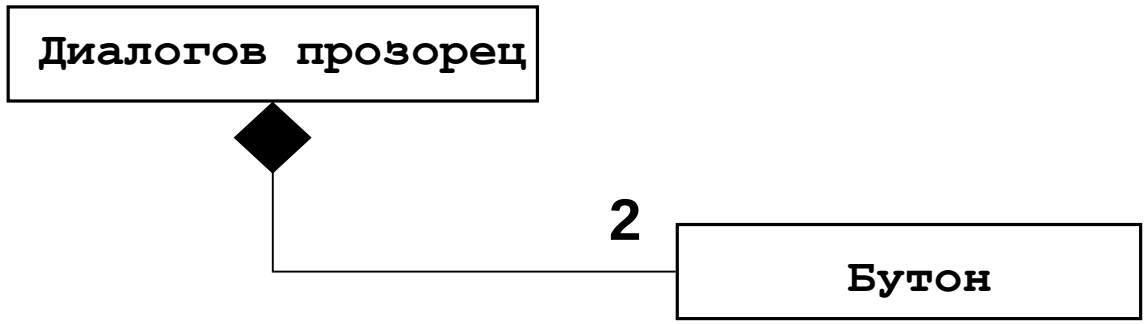
Компоненти наричаме класовете-наследници



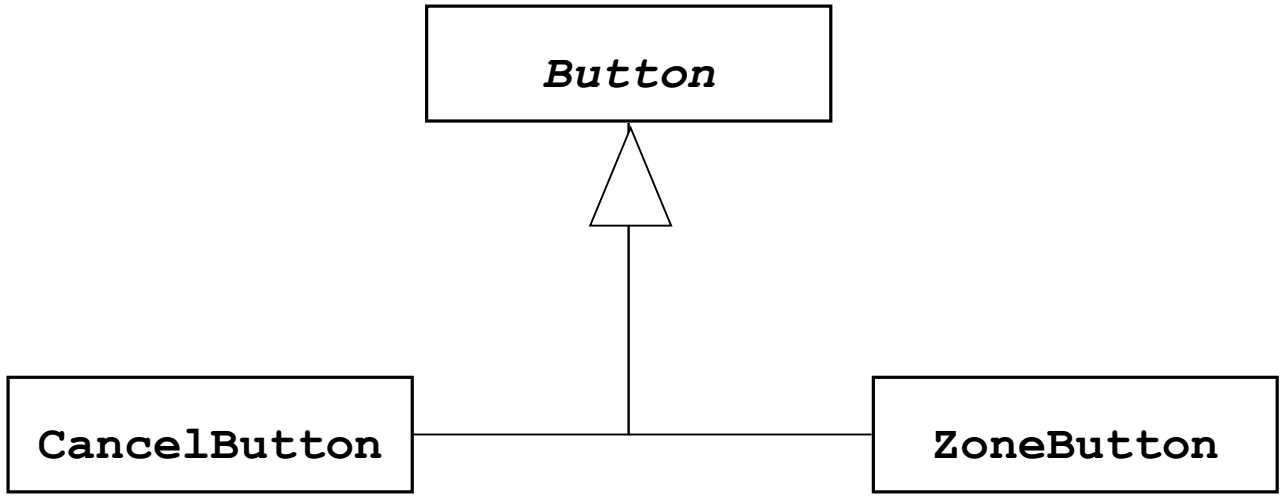
Композиция

Запълнен ромб означава *композиция*

Композицията е агрегация, при която компонентите не могат да съществуват без агрегата (родителя)



Наследяване



Класовете-наследници наследяват атрибутите и операциите от родителския (базовия) клас

Наследяването опростява модела като премахва повторенията