Лекция 13
Обектно-ориентирано проектиране (OOD)

DAAD Project
"Joint Course on Software Engineering"

**Humboldt University Berlin, University of Novi Sad, University of Plovdiv,
University of Skopje, University of Belgrade, University of Niš, University of Kragujevac**

**Parts of this topic use material from the textbook
H. Balzert, "Software-Technik", Vol. 1, 2nd ed., Spektrum Akademischer Verlag, 2001**

Version: Feb. 19, 2003 (D Jan. 30, 2003)

17. Обектно-ориентирано проектиране

a) Фази и дейности

b) Връзка с потребителския интерфейс

c) Повторно използване

d) Увеличаване на производителността

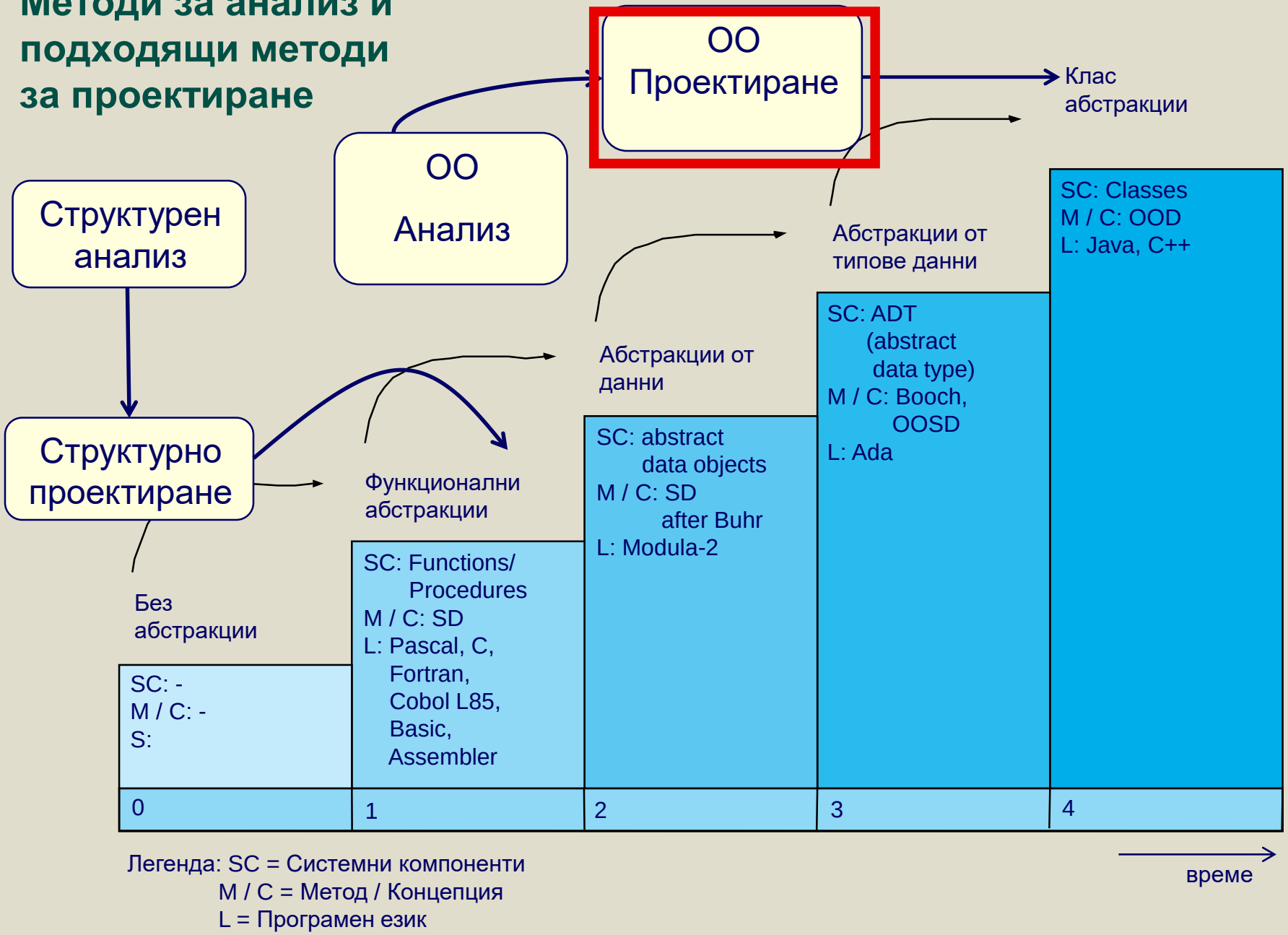
e) Проект на разработката

f) Рамки (Frameworks)

История

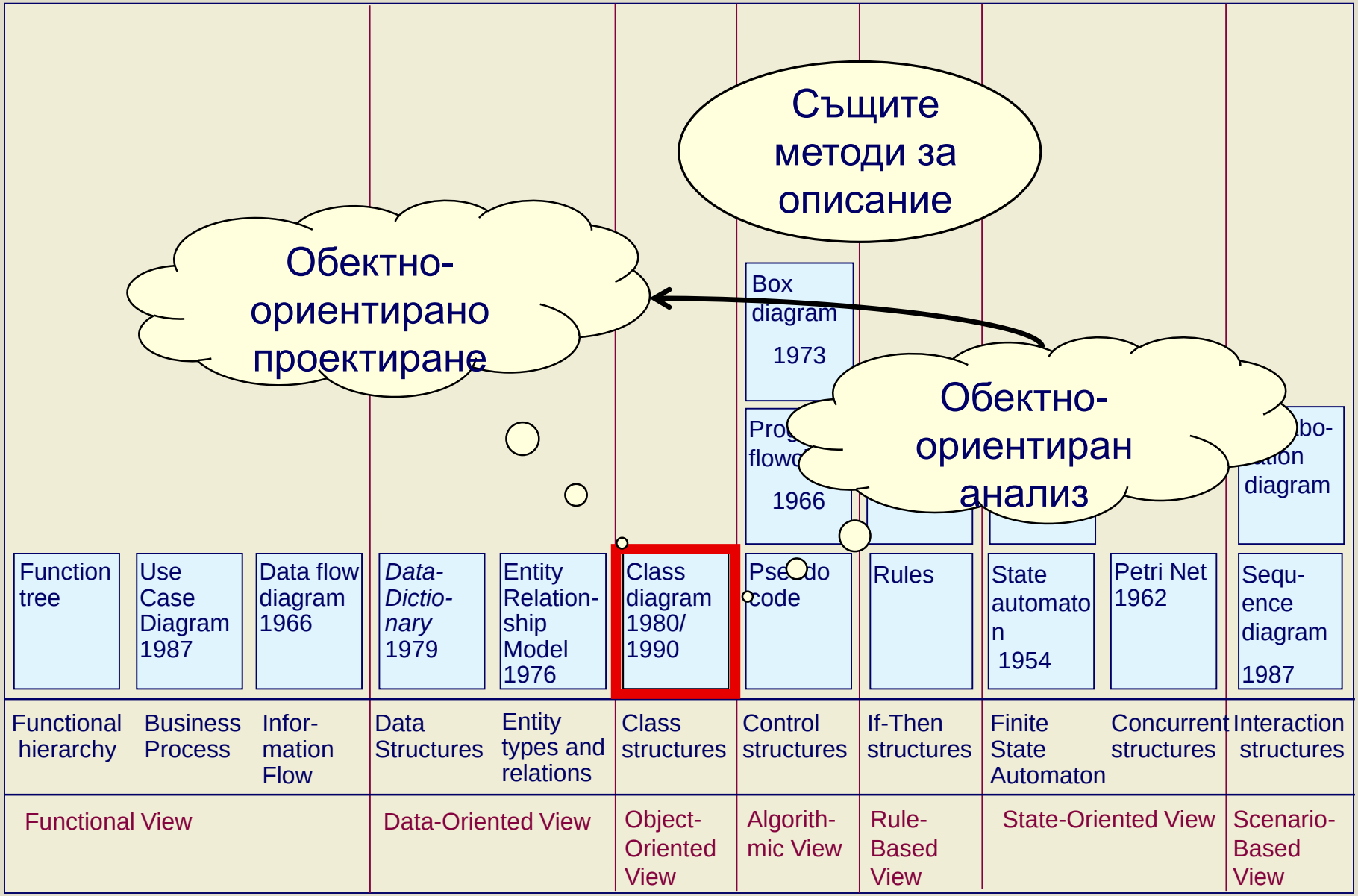
- *Dr. Bertrand Meyer*
  *1950 в Париж
  Президент на ISE Inc.
  (Interactive Software Engineering)
  в Санта Барбара, USA
- Основоположник на ООП
- Създател на програмния език Eiffel (1985/86)
- Книга *Object-Oriented Software Construction*
  1988.

Методи за анализ и подходящи методи за проектиране



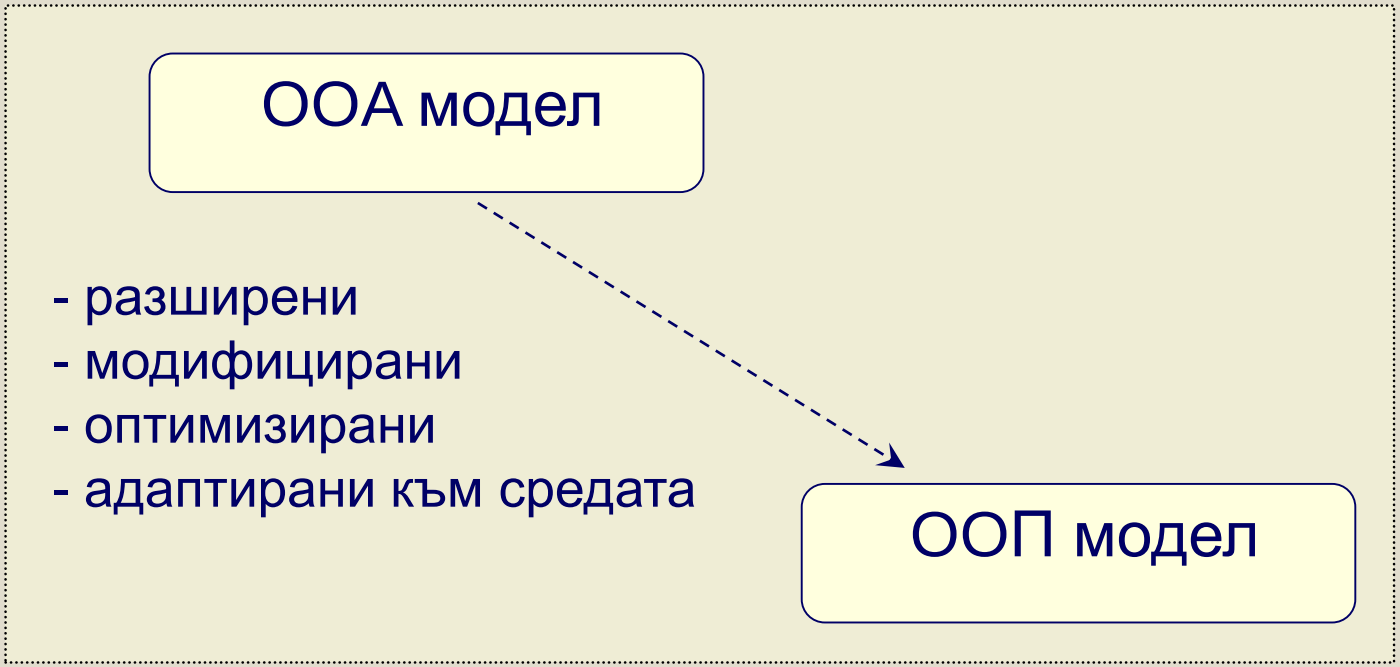
Основни концепции в софтуерната разработка

Balzert vol. 1, 2nd edition 2001



Фази и дейности за ООП

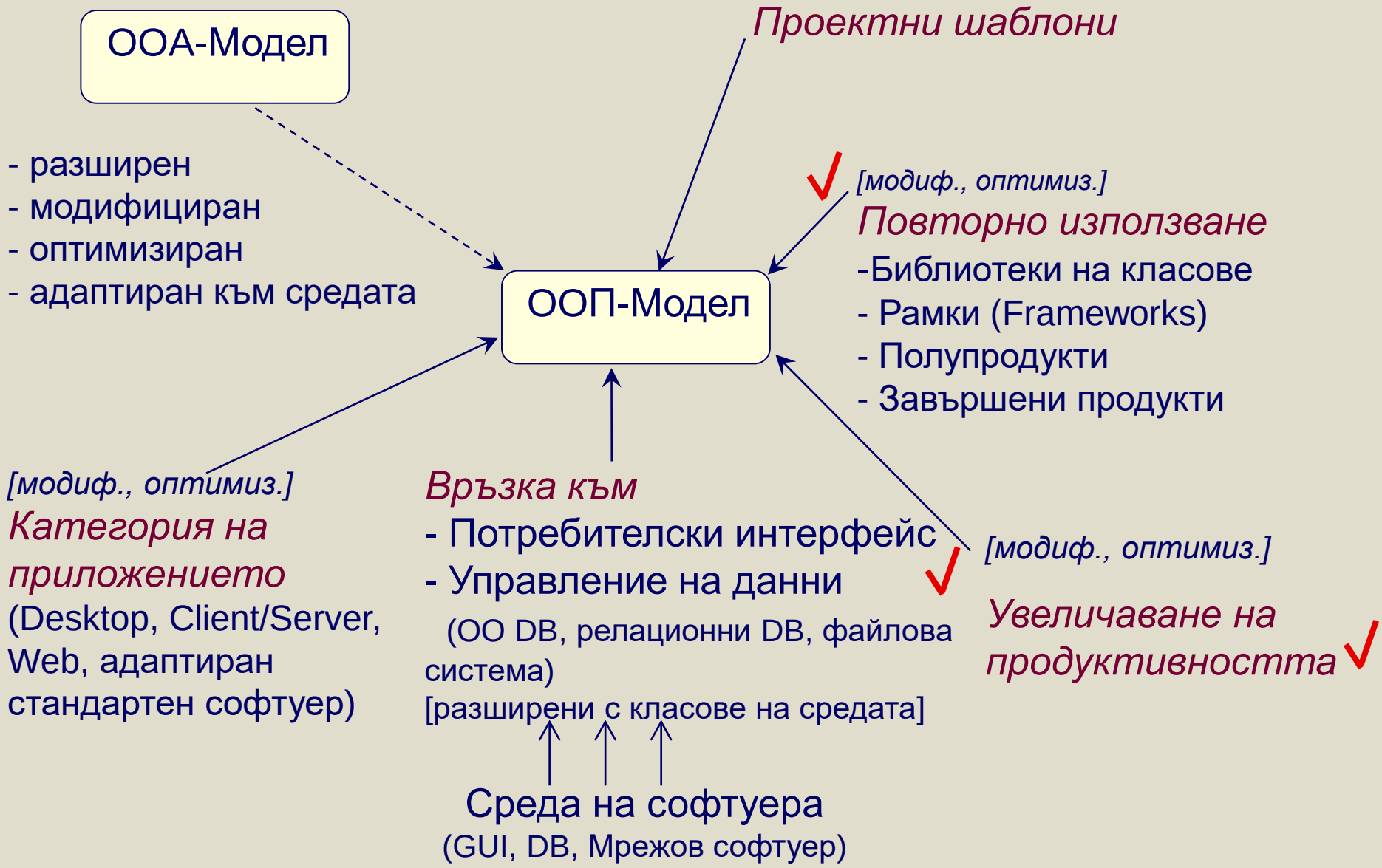
1. Фаза: проект на архитектурата



2. Фаза: проект на разработката

Настройка към езика за програмиране

Проект на архитектурата: влияещи фактори



17. Обектно-ориентирано проектиране

a) Фази и дейности

b) Връзка към потребителския интерфейс

c) Повторно използване

d) Увеличаване на производителността

e) Проект на разработката

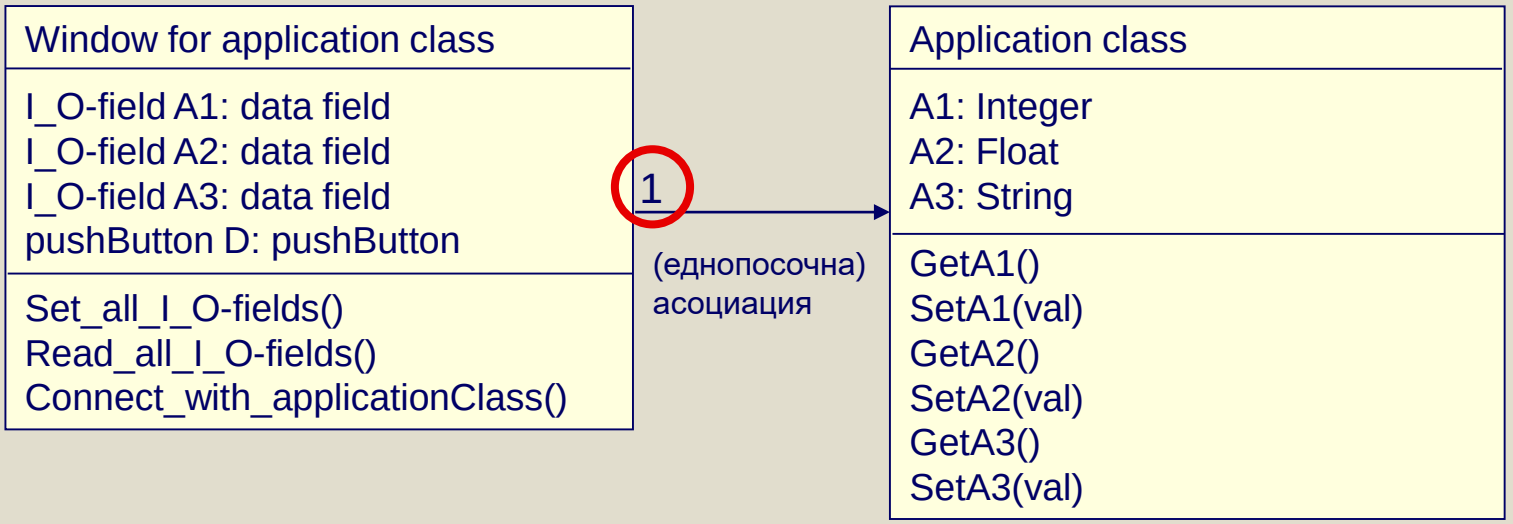
f) Рамки (Frameworks)

Връзка към потребителския интерфейс:
Разширение на ООА модел

- ООП модел: *разширение* на ООА модел с класове на потребителския интерфейс

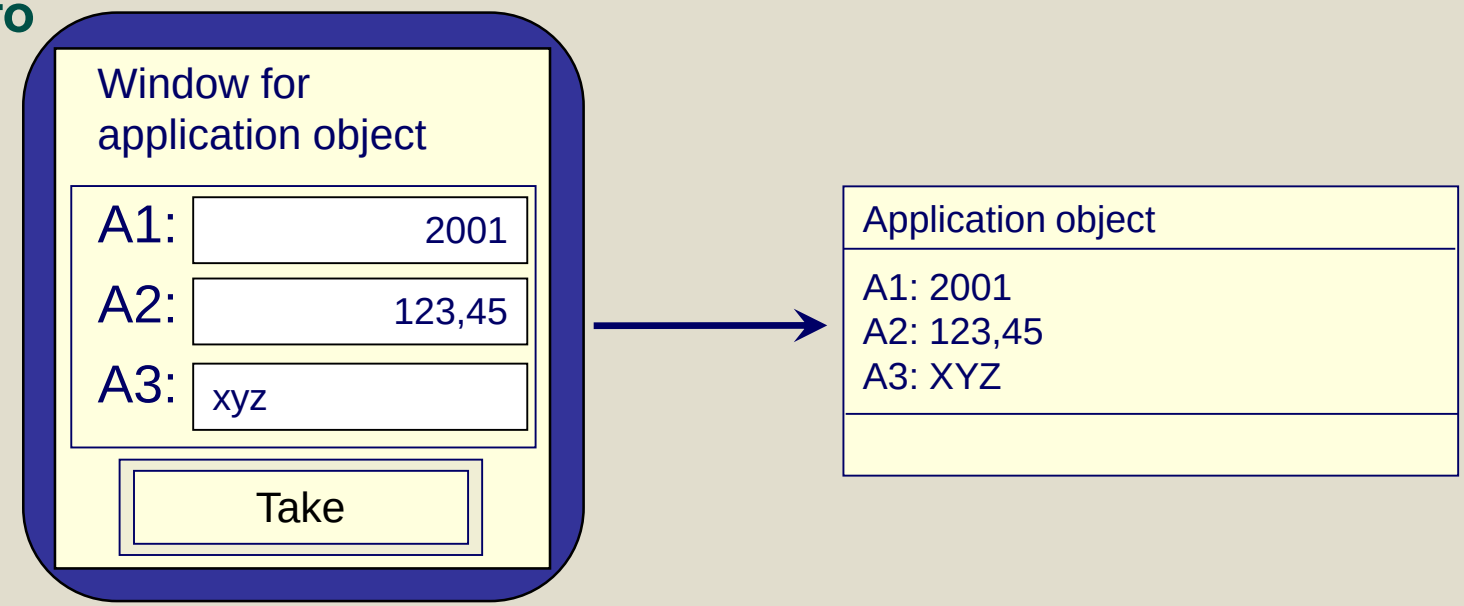
- *Принцип*: Класовете на потребителския интерфейс принадлежат към класовете на приложението.

Ниво на клас

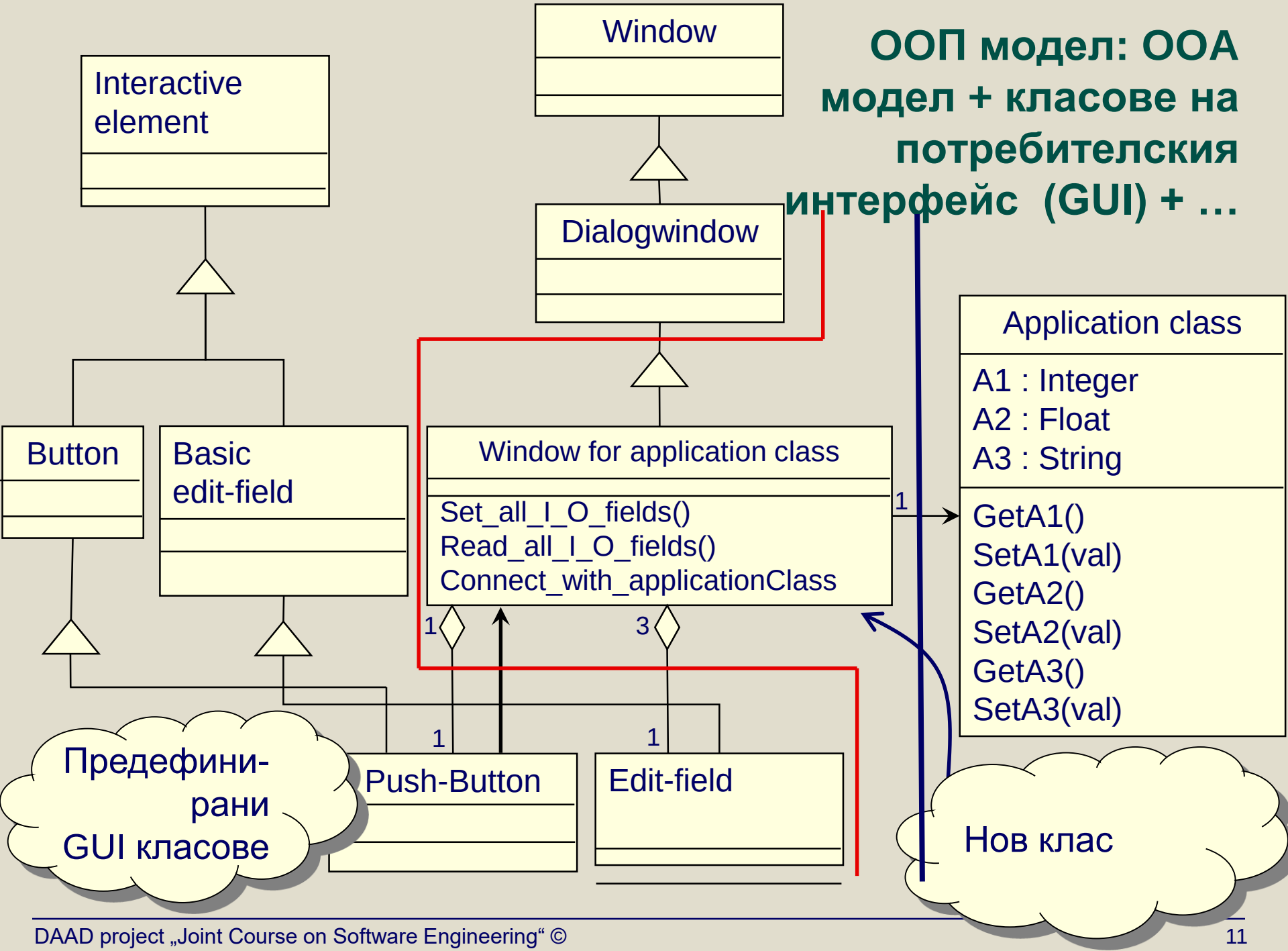


Свързване на потребителския интерфейс към приложението

Ниво на обект (пример)



ООП модел: ООА модел + класове на потребителския интерфейс (GUI) + …



17. Обектно-ориентирано проектиране

a) Фази и дейности

b) Връзка към потребителския интерфейс

c) Повторно използване

d) Увеличаване на производителността

e) Проект на разработката

f) Рамки (Frameworks)

Повторно използване

- *Разширение* на OOA модел:

  добавят се класовете, които са повторно използвани (reusable).

- *Модификация* на OOA модел:

  приспособяване към класове за повторно използване

Повторно използване: библиотеки от класове

- ООП модел:
  - модифициран ООА модел + класовете на потребителския интерфейс (GUI) + повторно използвани класове + …

17. Обектно-ориентирано проектиране

a) Фази и дейности

b) Връзка към потребителския интерфейс

c) Повторно използване

d) Увеличаване на производителността

e) Проект на разработката

f) Рамки (Frameworks)

Подобряване на продуктивността:
Модификация на OOA модел

- Съставен клас състоящ се от няколко класа:
  - В случаите на интензивна комуникация (размяна на съобщения)

  → възможна размяна на данни без операции

- Директен достъп до атрибути вместо с операции за достъп(за често достъпване)

  → по-бърз достъп

- Допълнителни атрибути вместо изчисления:

  Например: резултатен атрибут ↔ операция за изчисление

17. Обектно-ориентирано проектиране

a) Фази и дейности

b) Връзка към потребителския интерфейс

c) Повторно използване

d) Увеличаване на производителността

e) Проект на разработката

f) Рамки (Frameworks)

Проект на разработката: Настройка към програмния език

- *Модификация* на ООА модел
- Вземат се под внимание *липсващите концепции* от езика за разработка

  Примери:

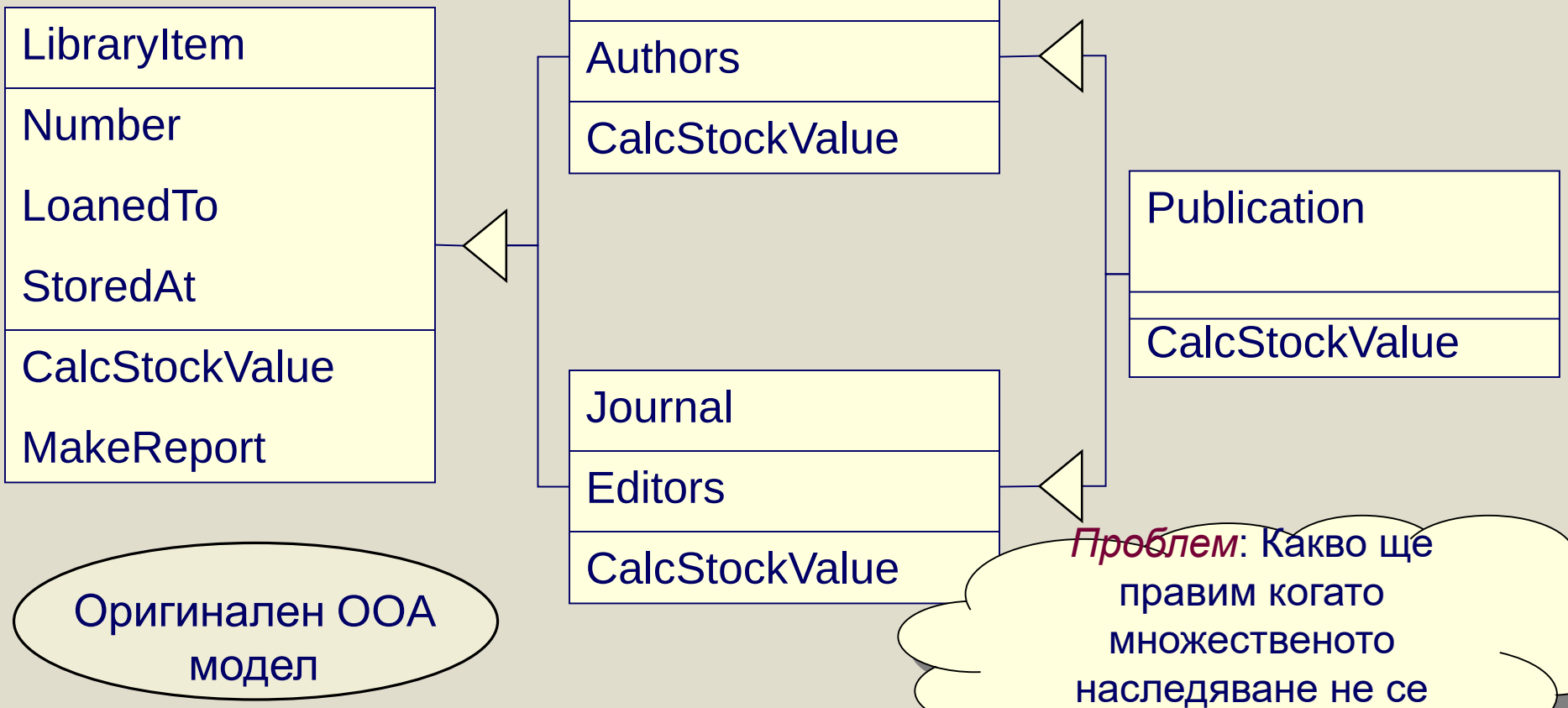
  - няма множествено наследяване
  - няма наследяване

  → ООА модел за в бъдеще трябва да бъде модифициран
- Внимава се за *специалните конструкции* на езика за разработка

  → Правила за настройка към C, Java ...

Решение за множествено наследяване и наследяване в ООП модел (1)

Множествено наследяване в ООА модел



Приложимо в C++ (например)

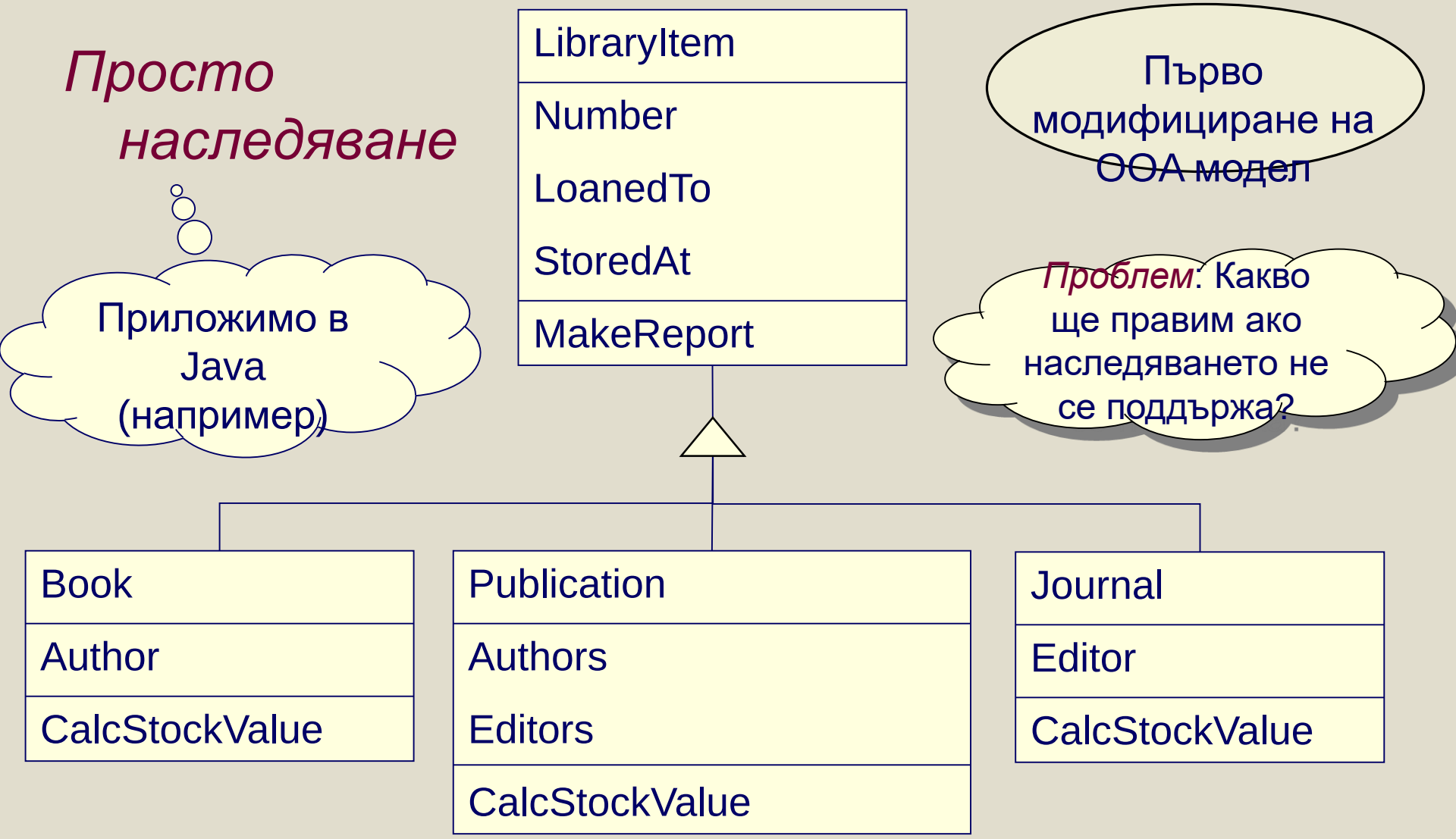
Оригинален ООА модел

Проблем: Какво ще правим когато множественото наследяване не се поддържа?

Решение за множествено наследяване и наследяване в ООП модел(2)

Просто наследяване

Приложимо в Java (например)



Първо модифициране на ~~ООА модел~~

Проблем: Какво ще правим ако наследяването не се поддържа?

Решение за множествено наследяване и наследяване в ООП модел(3)

Без наследяване

Приложимо в Modula-2 (например)

Book
Number LoanedTo StoredAt Author
MakeReport CalStockValue

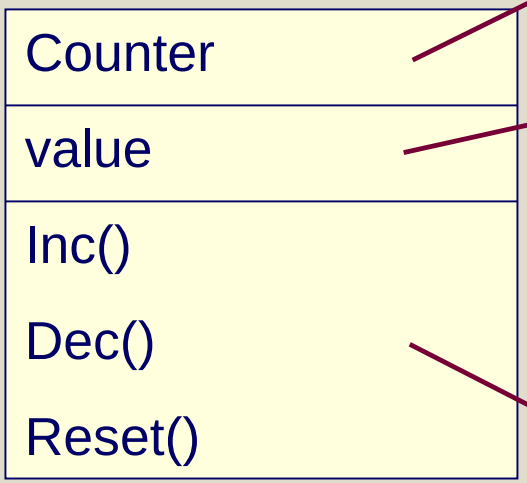
Publication
Number LoanedTo StoredAt Besoldung Editor
MakeReport CalStockValue

Journal
Number LoanedTo StoredAt Editor
MakeReport CalStockValue

Втора модификация на ООА модел

Разработка на C++ (проста диаграма)

ООА / ООП



• Декларациите на класовете въвеждат дефинирани от потребителите типове.

Specification of interface

```
class Counter
{
protected: unsigned int value;
  // attribute = member variable
public: //additionally necessary in C++
  Counter();  //Construktor
  ~Counter(); //Destruktor = delete
  unsigned int GetValue()
    //Reading attributes
  void SetValue (unsigned int newValue);
    //Writing
    //Operations = member functions
  void Inc();
  void Dec();
  void Reset();
};
```

Средство: генериране на рамка на кода

17. Обектно-ориентирано проектиране

a) Фази и дейности

b) Връзка към потребителския интерфейс

c) Повторно използване

d) Увеличаване на производителността

e) Проект на разработката

f) Рамки (Frameworks)

Рамка (Framework)

Цел:

- повторно използване на софтуер
- няма ограничение за използване на код (програми) повторно

Рамка: 4 дефиниции (1)

- Система от повторно използваеми и приспособими класове и приспособими клас библиотеки (възможност за настройка: нови под-класове, предефиниране и/или разработване на празни операции)

(Balzert 96)

→ Разработка ориентирана към дефиниции

- Множество от взаимодействащи си класове, които изграждат проект за повторно използване за специфичен тип софтуер… Рамката (framework) определя архитектурата на приложението… обхваща проектни решения, които са общи за даден приложен домейн.

(Gamma 94)

→ Софтуерна архитектура + части от разработката (приспособими клас библиотеки) – ориентирани към определена приложна област

Рамка (Framework): 4 дефиниции (2)

- *Общи софтуерни архитектури* с *общи части за изпълнение* (клас библиотеки) за конкретен клас проблеми.

(OBJEKTspektrum 1/96, S.13)

- Повторно използваеми основни спецификация, проект, код и тестови шаблони за част от приложение

(Firesmith, Dictionary of Object Technology 1995)

→ Идея: софтуерните рамки на приложения, включват всички фази от софтуерната разработка

→ *Общ софтуер (всички фази!) за конкретна приложна област*

Основни проектни принципи ООП (SOLID)

- **S**ingle Responsibility Principle (SRP). *„Класът трябва да има една единствена отговорност.“*
- **O**pen-Closed Principle (OCP). *“Модулът (компонента) трябва да бъде отворен за разширения, но затворен за модификации.“*
- **L**iskov Substitution Principle (LSP). *“Подкласовете трябва да заменят техните базови класове.“*
- **I**nterface Segregation Principle (ISP). *“Много специфични клиентски интерфейси са по-добре от един с обща цел.“*
- **D**ependency Inversion Principle (DIP). *“Зависимост от абстракции. Не зависимост от конкретики.”*