

ЛЕКЦИЯ I © 2020 Нет Ит

JAVA OOP: Design Patterns

Теодор Костадинов



SOFTWARE
ACADEMY



Съдържание

1. Design Patterns
2. Singleton
3. Strategy
4. Observer
5. Decorator
6. GUI



What are Design Patterns



Често срещани проблеми, Често използвани решения

- Шаблоните за дизайн (Design Patterns) представляват концепция предназначена за разрешаване на често срещани проблеми в обектно-ориентираното програмиране.
- Използване на колективния опит за софтуерно проектиране за доказани решения на често срещани проблеми
- Поощряват reusability на кода, което води до по-качествен и лесен за поддръжка код
- Обща терминология, която помага на програмистите да се разбират лесно
- Концепцията предлага стандартни решения за архитектурни проблеми в програмирането.
- Не става въпрос за конкретни алгоритми или част от програмен код.
- Шаблоните за дизайн са независими от програмния език.
- Те представляват архитектурни решения на вече познати и много често срещани проблеми в програмирането.
- Решенията са по-скоро абстрактни, отколкото конкретни

Видове



CREATIONAL

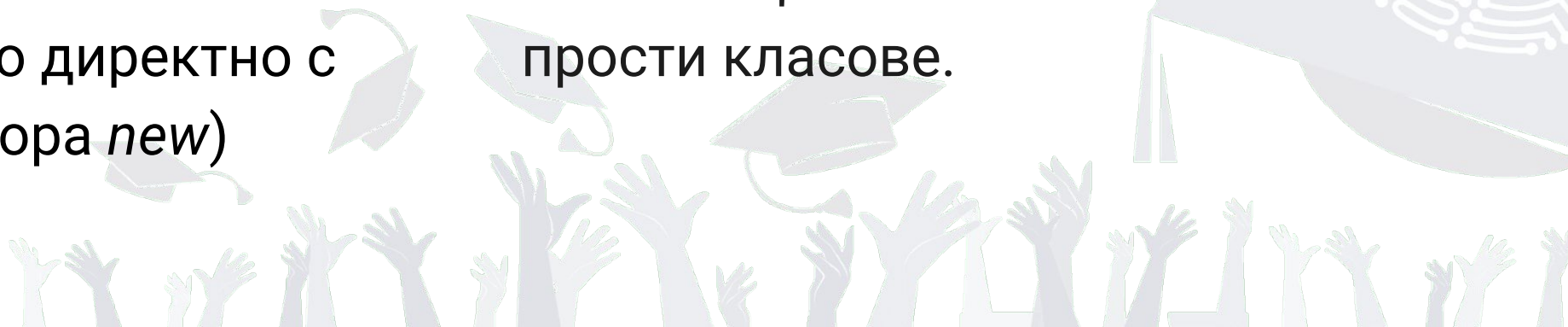
Осигуряват начин да се създават обекти на класове, скривайки логиката по създаването им (вместо директно с оператора *new*)

STRUCTURAL

Осигуряват различни начини за създаване на по-сложни класове чрез наследяване и композиция на по-прости класове.

BEHAVIORAL

Свързани са с комуникацията между различните обекти в системата

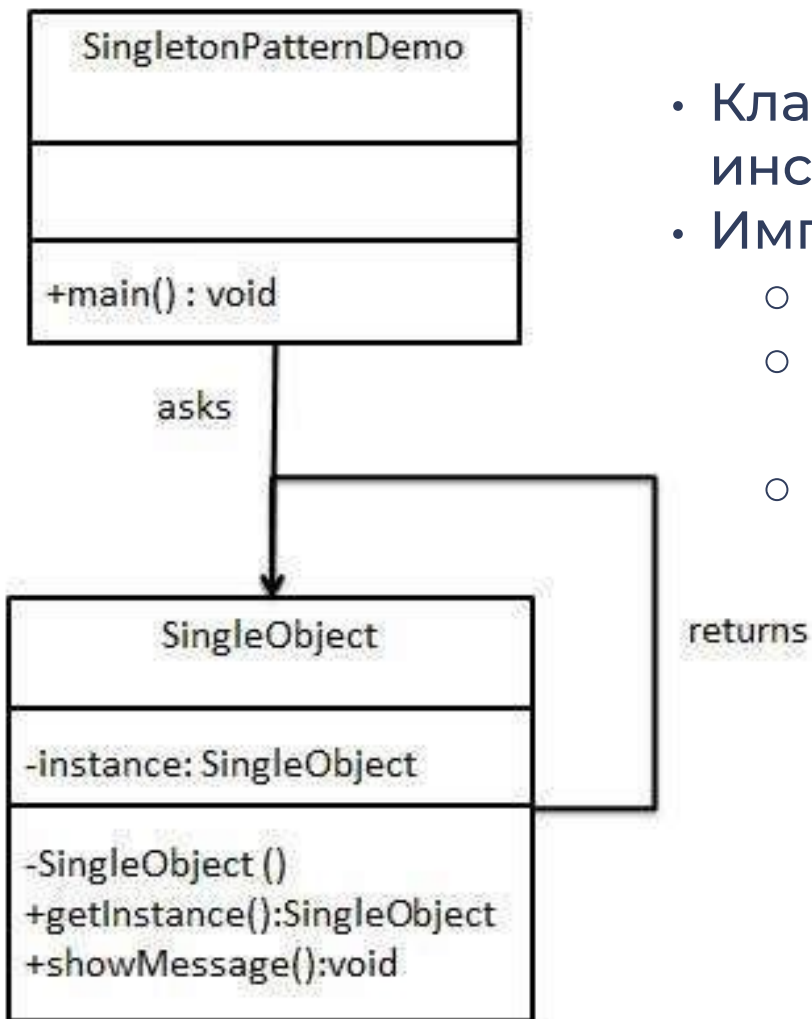


Design Pattern vs Design Principle

- Патърните решават един конкретен проблем
- Принципите могат да решават множество проблеми.
- Принципите ни учат как да пишем кода си като цяло, те не са съсредоточени върху един конкретен проблем.
- Един патърн може да използва един или няколко принципа в себе си.



Singleton



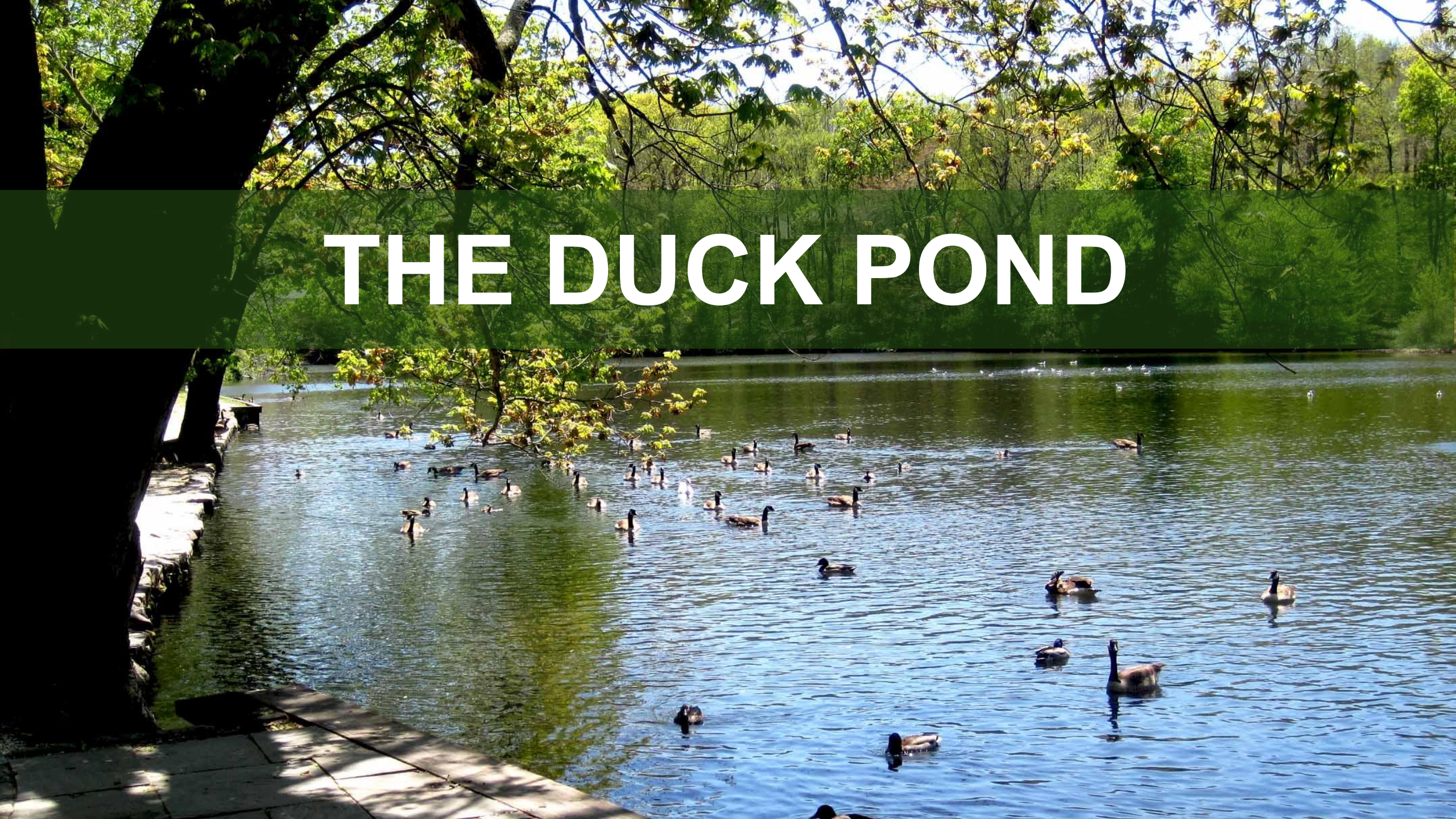
- Клас, от който може да съществува най-много една инстанция
- Имплементация:
 - `private` конструктор
 - `private static` член-променлива от тип същия клас, която реферира единствената инстанция на класа
 - `public static` метод, който връща инстанцията на класа



Design Pattern

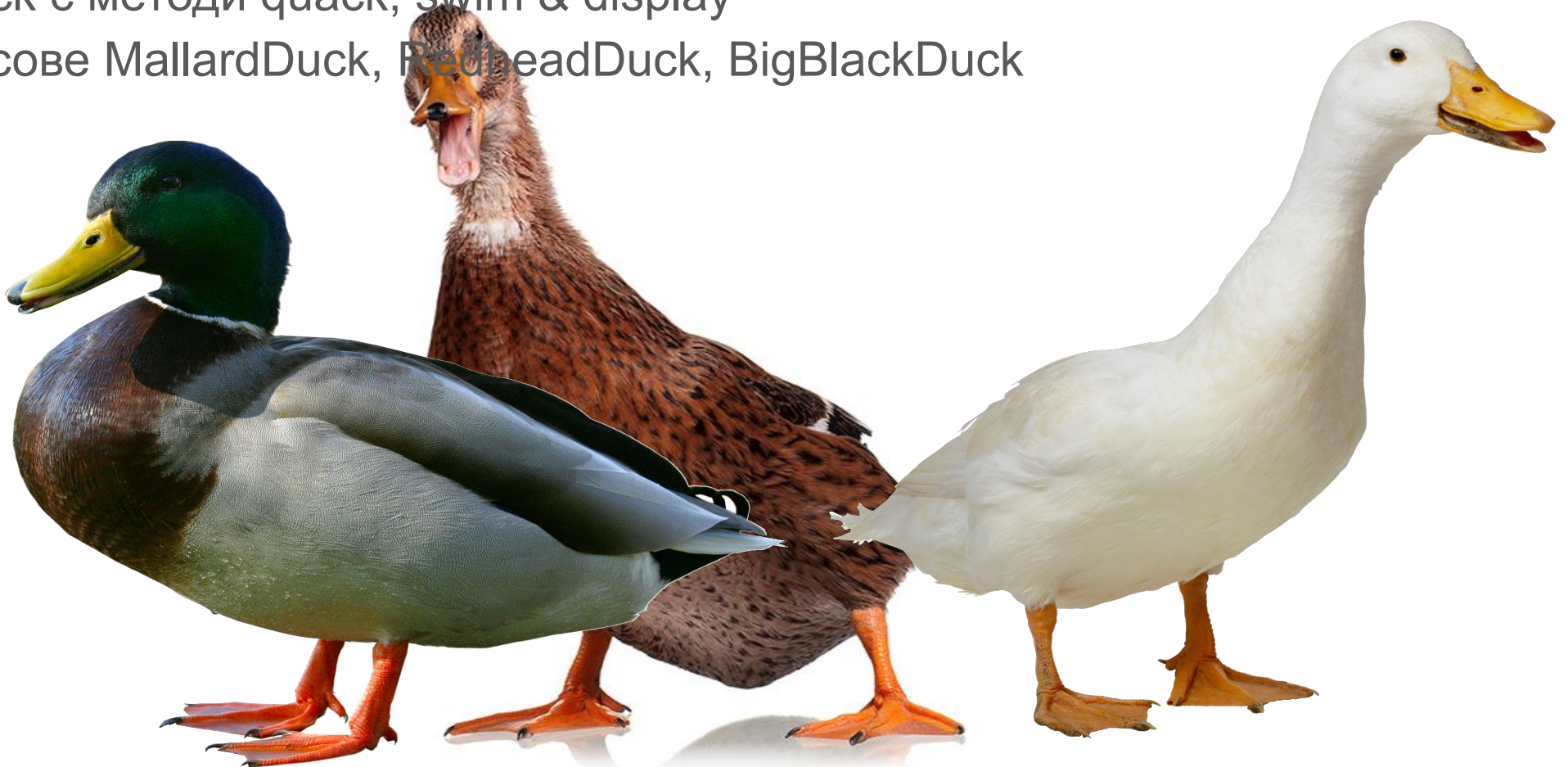
The Singleton Pattern ensures a class has only one instance, and provides a global point of access to it.

THE DUCK POND



Какво имаме в момента

- Работим над софтуер, който симулира езеро с патици
- Потребителите могат да си добавят патици и да ги управляват
- Клас Duck с методи quack, swim & display
- Под-класове MallardDuck, RedheadDuck, BigBlackDuck

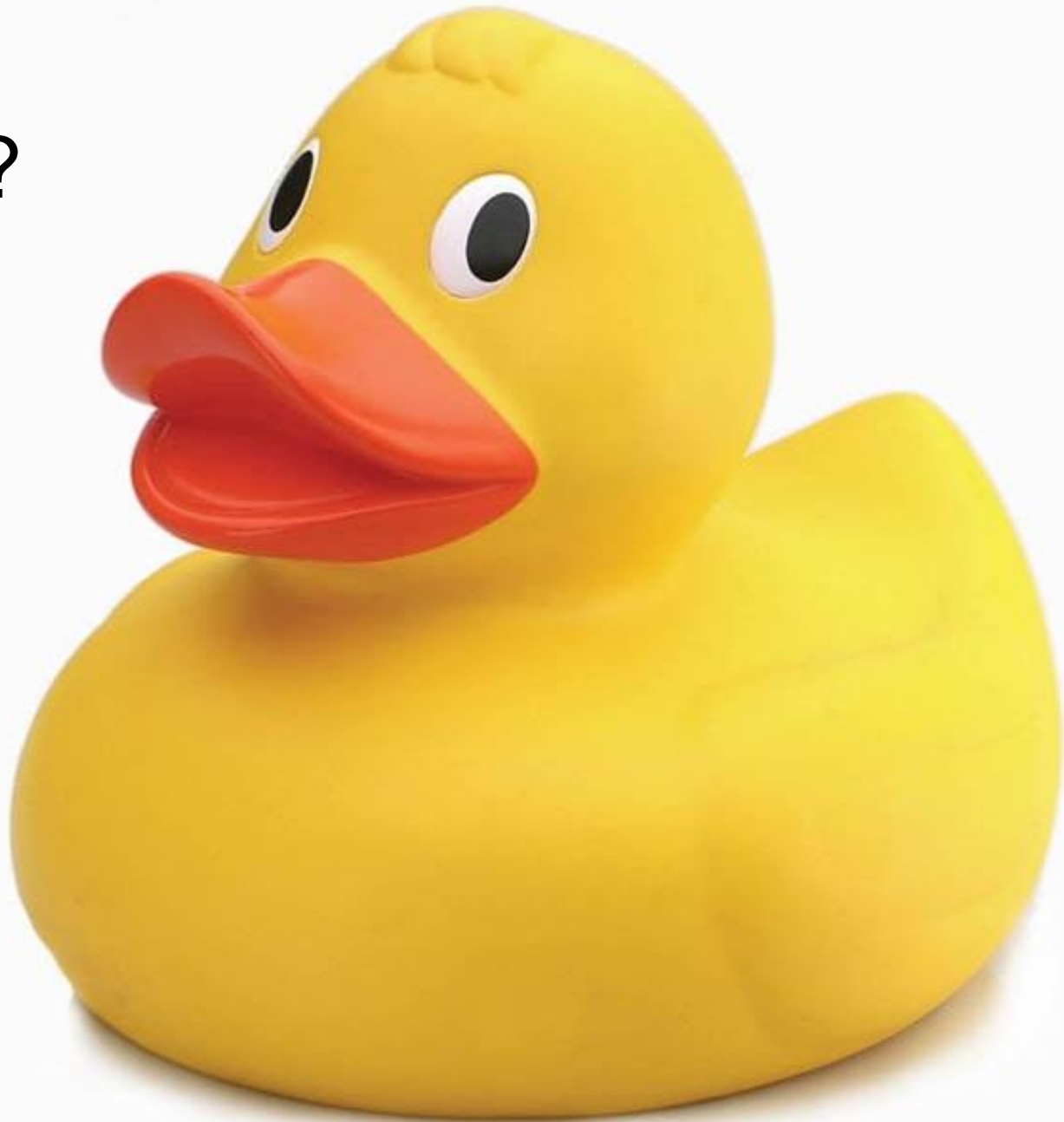


Какво искаме

- Да добавим поведение - `fly()`



Къде е проблемът?



Кои от следните са минуси при използването на наследяване в случая?

- Code is duplicated across subclasses.
- Runtime behavior changes are difficult.
- We can't make ducks dance.
- Hard to gain knowledge of all duck behaviors.
- Ducks can't fly and quack at the same time.
- Changes can unintentionally affect other ducks.



Ами сега?

Задача за упражнение

A green rectangular road sign with rounded corners and a white border of reflective dots. The word "Change" is written in large, white, sans-serif capital letters. The sign is mounted on two wooden posts. The background is a bright blue sky with scattered white clouds.

Change

Design Principle

Определете аспектите на системата, които ще се променят, и ги отделете от тези, които няма.

Design Principle



***Программируйте прямо
интерфейсы, не
конкретика.***

Design Principle



***Предпочитайте композиция
за сметка на наследяване.***



Design Pattern

The Strategy Pattern defines a family of algorithms, encapsulates each one, and makes them interchangeable. Strategy lets the algorithm vary independently from clients that use it.

Почивка

до 20:35



Проблемът с данните

- Имаме устройства, които мерят данни от въздуха
- Те изпращат данните по БТ към апп на телефона
- На телефона имаме бекграунд процес, който чете и записва данните
- Имам 5 различни екрана в аппа, които трябва да показват данните в реално време под една или друга форма
- Как да го направим?



Първоначално

```
public class WeatherData {  
  
    // instance variable declarations  
  
    public void measurementsChanged() {  
  
        float temp = getTemperature();  
        float humidity = getHumidity();  
        float pressure = getPressure();  
  
        currentConditionsDisplay.update(temp, humidity, pressure);  
        statisticsDisplay.update(temp, humidity, pressure);  
        forecastDisplay.update(temp, humidity, pressure);  
    }  
  
    // other WeatherData methods here  
}
```

Grab the most recent measurements by calling the WeatherData's getter methods (already implemented).

Now update the displays...

Call each display element to update its display, passing it the most recent measurements.

Based on our first implementation, which of the following are true?
(Choose all that apply.)



A.	We are coding to concrete implementations, not interfaces.
B.	For every new display element we need to alter code.
C.	We have no way to add (or remove) display elements at run time.
D.	The display elements don't implement a common interface.
E.	We haven't encapsulated the part that changes.
F.	We are violating encapsulation of the WeatherData class.



Design Pattern

The Observer Pattern defines a one-to-many dependency between objects so that when one object changes state, all of its dependents are notified and updated automatically.

Design Principle

*Стрежете се към слаби
връзки между обекти,
които си взаимодействат.*

The Starbucks example

- Имаме клас Beverage с абстрактен метод `cost()`
- Всеки под-клас трябва да задава собствен `cost()`
- Какво правим ако потребителя иска допълнително сметана?



Design Principle

***Класовете трябва да са
отворени за надграждане,
но затворени за променяне.***



Това е т.нар. open-closed принцип



Какво означава това за Starbucks?

- Взимаме BlackCoffee object
- Добавяме му Mocha object
- Добавяме му Whip object
- Викаме `cost()` и разчитаме всичко да е ок



So, a **DarkRoast** wrapped in **Mocha** and **Whip** is still a **Beverage** and we can do anything with it we can do with a **DarkRoast**, including call its `cost()` method.

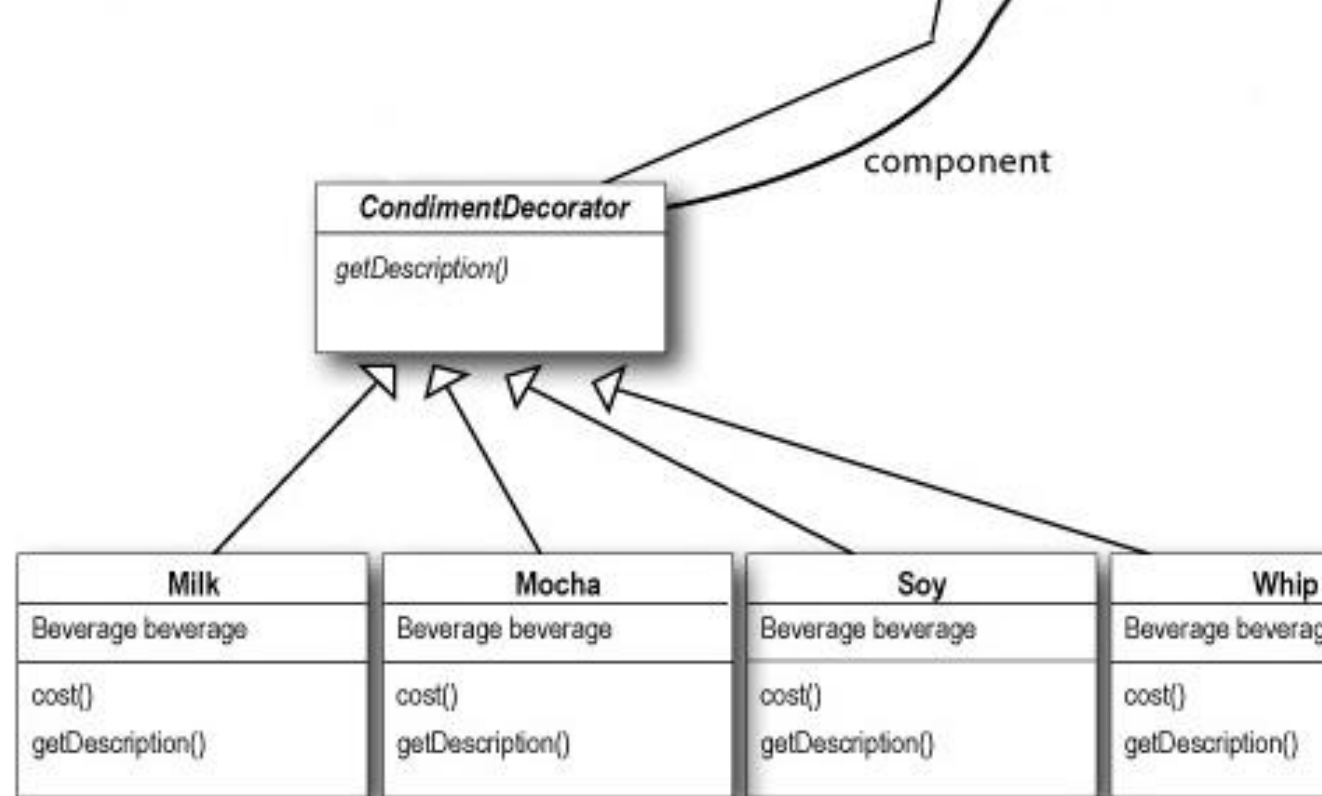
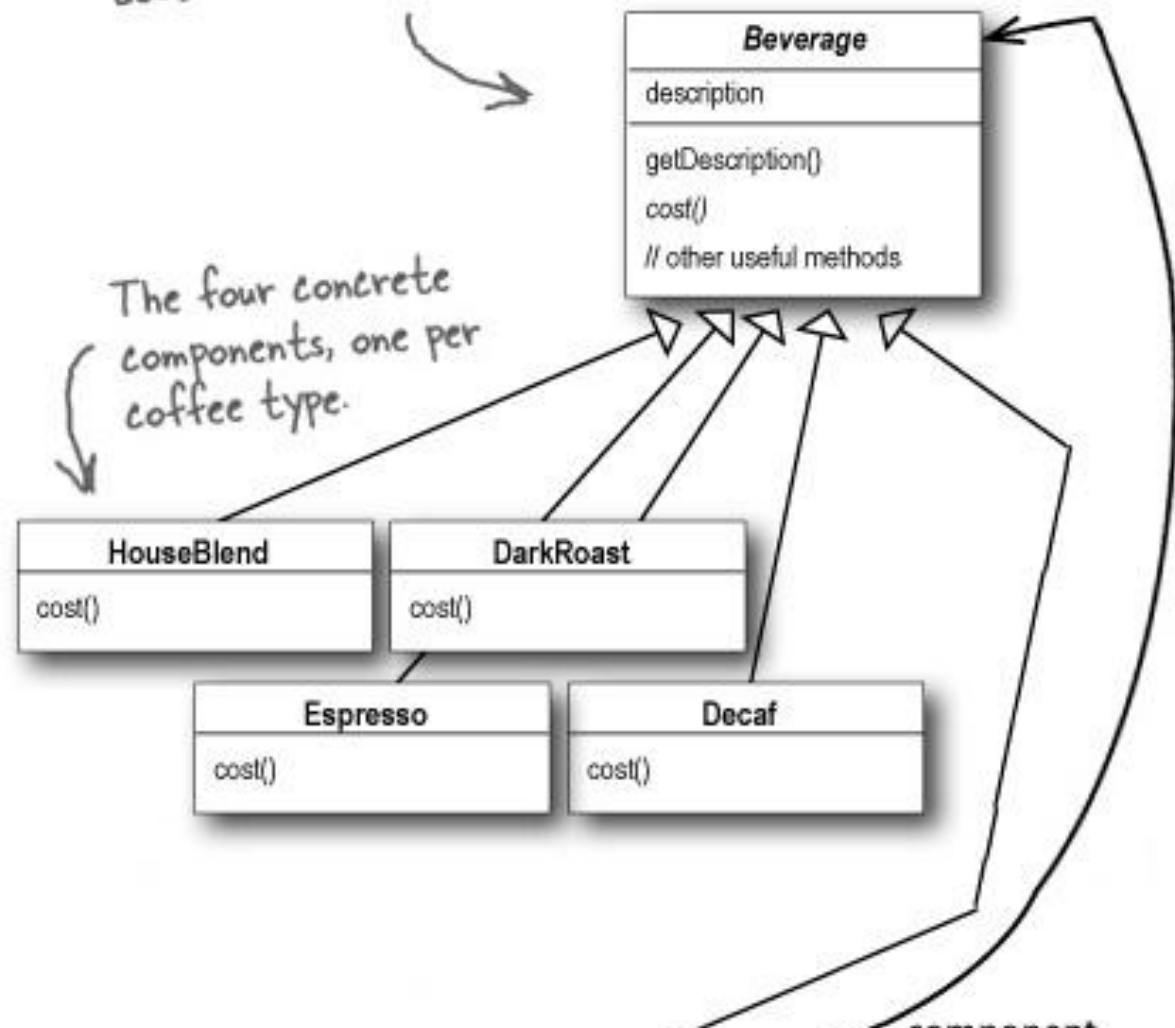
Какво знаем за декораторите до сега

- Имат същия родител като обектите, които декорират
- Може да се използва повече от един
- Предвид първото, можем да подменим оригиналния обект за сметка на декориран такъв
- Декоратора добавя собствено поведение, преди или след като делегира на обекта да свърши останалата работа
- Може да декорираме по всяко време, дори динамично

Да го приложим

Beverage acts as our abstract component class.

The four concrete components, one per coffee type.



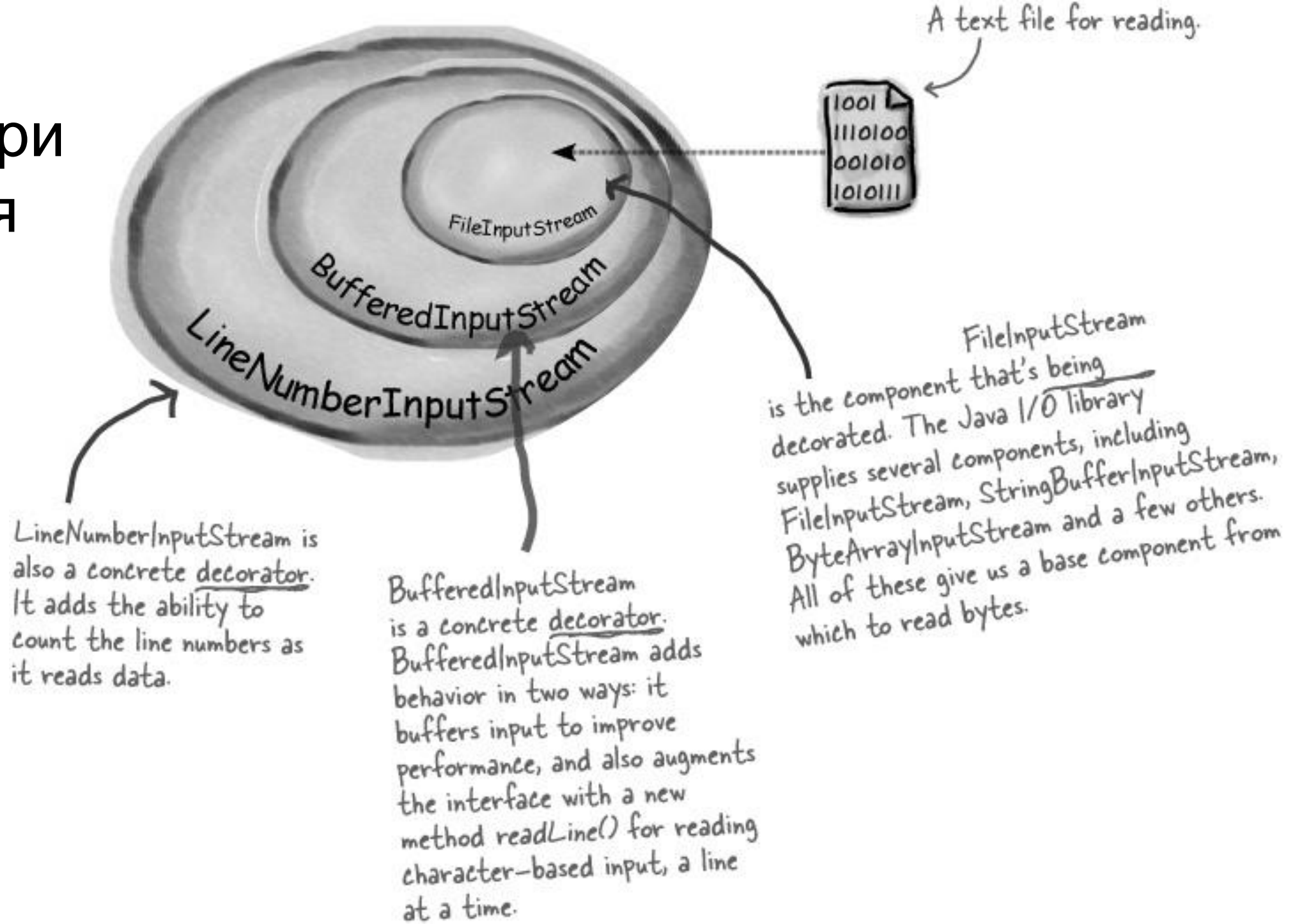
And here are our condiment decorators; notice they need to implement not only `cost()` but also `getDescription()`. We'll see why in a moment...



Design Patterns

The Decorator Pattern attaches additional responsibilities to an object dynamically. Decorators provide a flexible alternative to subclassing for extending functionality.

Декоратори в реалния ЖИВОТ



User Interface



Graphical User Interface

- Графичен потребителски интерфейс (GUI) е термин, използван не само в Java, но във всички езици на програмиране, които поддържат подобна функция.
- Графичният потребителски интерфейс на програмата представя лесен за използване от потребителя визуален дисплей.
- Той е съставен от графични компоненти, чрез които потребителят може да взаимодейства със страницата или приложението.
- Най-често GUI се пише на Front-end език (с HTML & CSS) и с библиотеки като React, Angular, Vue.js
- UI може да се пише и с джава (макар и да не е препоръчително) със Swing библиотеката



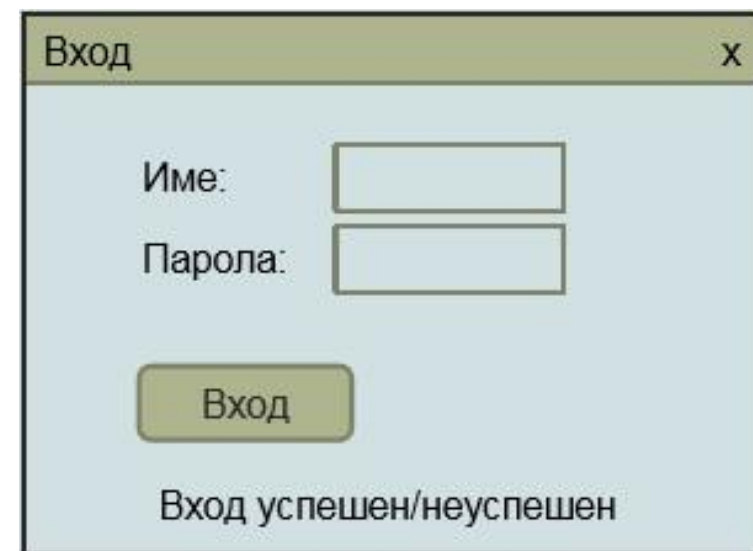
Swing



- Основен компонент на Swing е JFrame. Класовете, които го наследяват трябва да имат задължителен конструктор.
- При създаването на обект от такъв клас, се създава визуален прозорец на екрана.
- В класа на JFrame-а се декларираат свойствата на прозореца, както и вътрешните компоненти:
 - JButton
 - JTextArea
 - JProgressBar
 - BorderLayout
 - GridLayout
 - FlowLayout
 - CardLayout



- Направете форма за вход на потребител със Swing. Нека формата да има следния вид:
- При натискане на бутона “Вход”:
- Ако въведените име и парола са равни съответно “admin” и „1234“, да изписва в текстовото поле „Вход успешен“.
- Ако са различни, да изписва „Вход неуспешен“.
- (По-важно е да направите да работи както е указано, отколкото елементите да са подредени както на картинката.)



Ами сега?

Задача за упражнение

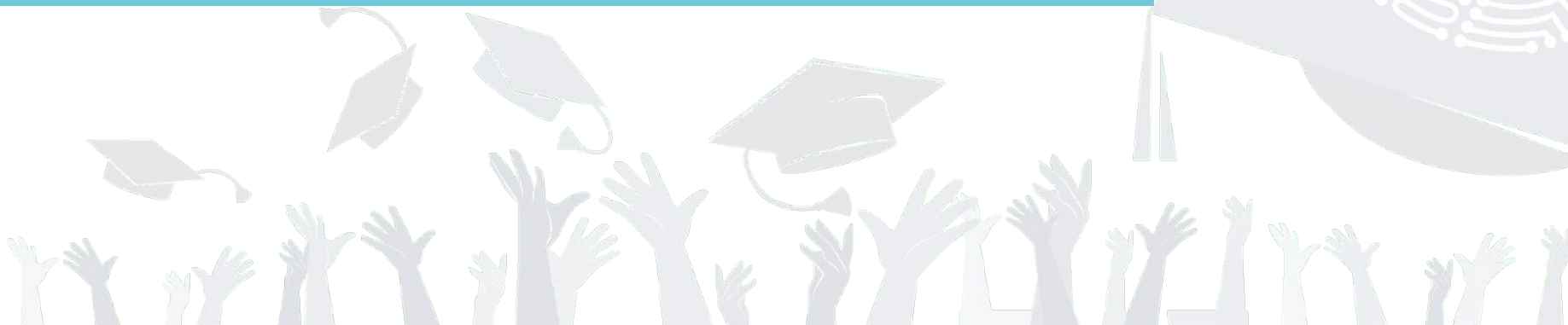
ВЪПРОСИ?



Резюме

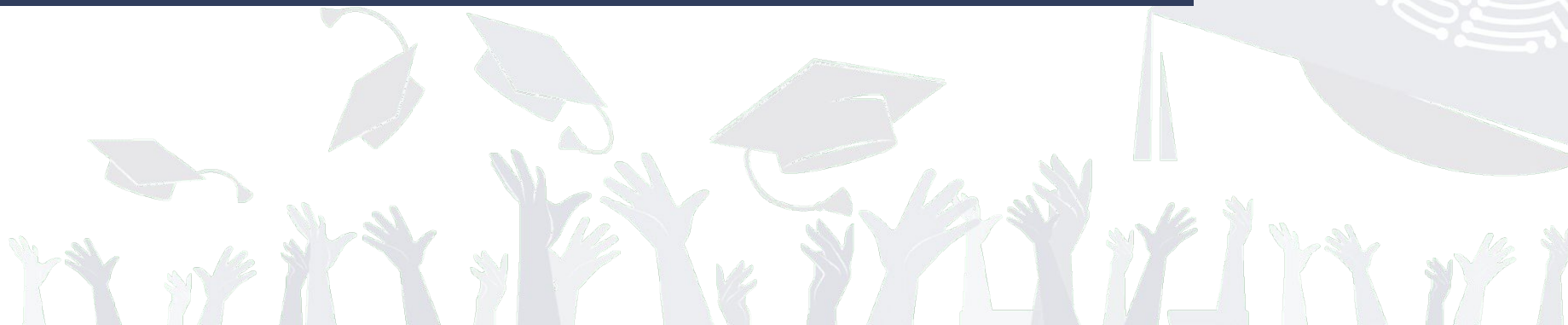


- Има много и всякакви дизайн патърни. Трябва да знаем, че съществуват, че да можем да ги ползваме.
- Swing е популярната библиотека за Java UI. И все пак, UI не се прави на джава.



Ресурси

- [GitHub Repo with Demos](#)
- [Book for Design Patterns](#)
- [Another DP lecture](#)
- [Gang of Four Book](#)



© 2020 Нет Ит

БЛАГОДАРЯ ЗА ВНИМАНИЕТО!



SOFTWARE
ACADEMY

