Практическое изучение объектно-ориентированного программирования с применением электронного задачника Programming Taskbook for OOP

Абрамян М.Э.[1,2]

[1]*Южный федеральный университет, Институт математики, механики и компьютерных наук им. И. И. Воровича*

[2]*Университет МГУ-ППИ в Шэньчжэне (КНР)*

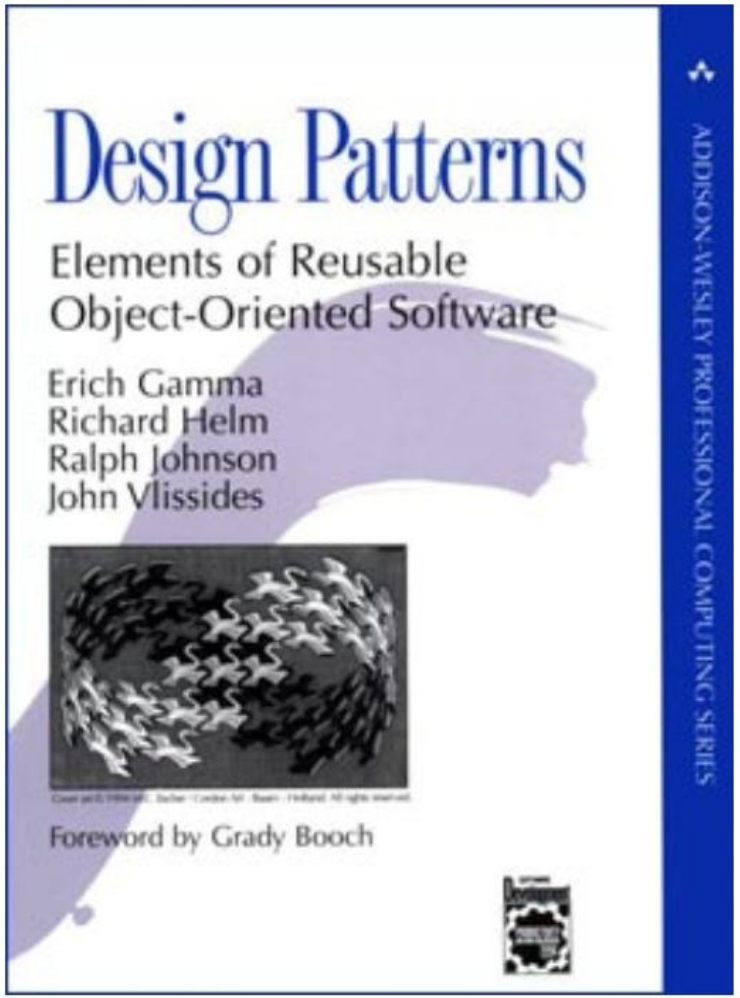
mabr@sfedu.ru

Доклад на пятой Всероссийской научно-методической конференции **«Использование системы программирования PascalABC.NET в обучении программированию»**

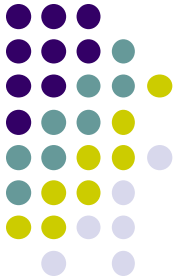
Ростов-на-Дону, 28-29 марта 2025 г.

Паттерны объектно-ориентированного проектирования

- **Паттерн** (англ. design pattern) в разработке программного обеспечения – повторяемая конструкция, представляющая собой решение проблемы проектирования в рамках некоторого часто возникающего контекста.
- Книга **Design Patterns** четырех авторов (Gamma, Helm, Johnson, Vlissides) представляет собой классическое описание 23 распространенных паттернов проектирования.

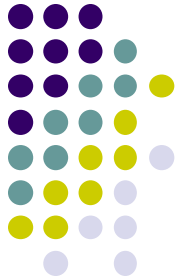


Обучение паттернам проектирования на примерах



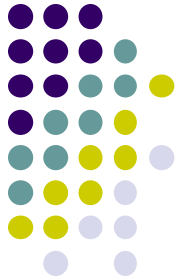
- Рассматривается модельная задача, для которой разрабатывается система классов, соответствующая изучаемому паттерну.
- Правильность системы проверяется путем обработки наборов тестовых данных.
- Для повышения надежности тестирования желательно **автоматически** генерировать исходные данные и проверять правильность полученных результатов.

Электронный задачник по паттернам проектирования



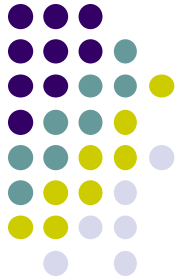
- Является расширением универсального задачника Programming Taskbook
- Предназначен для поддержки практической части курсов по ООП
- Позволяет изучить и реализовать на практике 21 паттерн проектирования
- Предусматривает дополнительную поддержку для языков PascalABC.NET, C++, C#, Java, Python, Ruby, Julia
- Интегрирован в среду программирования PascalABC.NET

Состав электронного задачника по паттернам проектирования (1)



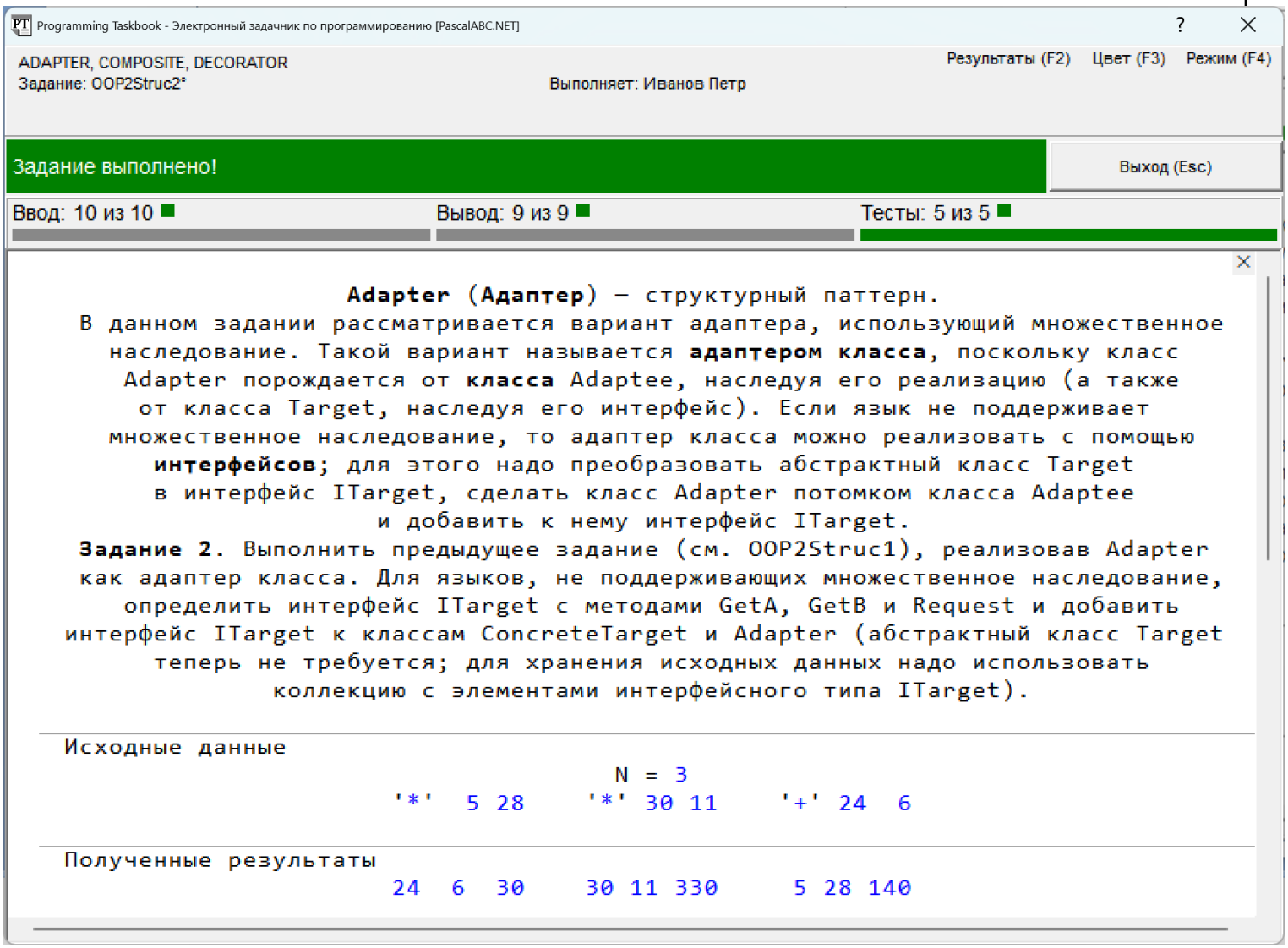
- Вводная группа **OOP0Begin** (12 заданий)
  - Создание классов и объектов, скрытие данных (**инкапсуляция**)
  - Порождение новых классов (**наследование**)
  - Возможность единообразной обработки объектов разных типов, входящих в одну иерархию (**полиморфизм**)
  - Использование **информации о типе времени выполнения** (RTTI)
  - Генерация и обработка **исключений**
  - **Шаблонные классы** и шаблонные функции
  - **Перегрузка операций**, приведение объекта к другим типам, глубокое копирование

Состав электронныого задачника по паттернам проектирования (2)

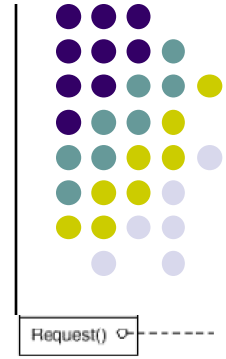


- **OOP1Creat** – порождающие паттерны (10)
  - Factory Method (3), Abstract Factory (2)
  - Singleton (1), Prototype (2), Builder (2)
- **OOP2Struc** – структурные паттерны (11)
  - Adapter (4), Composite (2), Decorator (2)
  - Proxy (1), Bridge (1), Flyweight (1)
- **OOP3Behav** – паттерны поведения (17)
  - Observer (2), Strategy (2), Template Method (2)
  - Iterator (1), Command (2), State (2)
  - Mediator (1), Chain of Responsibility (2), Visitor (1), Interpreter (2)

Окно электронного задачника с сообщением о решении задачи

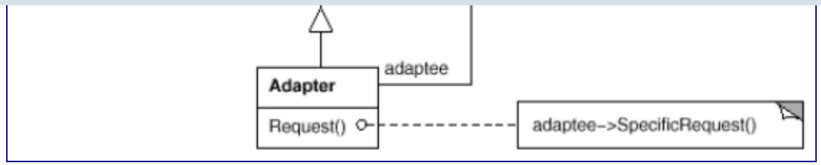


Представление формулировки задачи в формате html



PT4Tasks

file:///C:/PABCWork.NET/PT4Tasks.html



Adapter (**Адаптер**) — структурный паттерн.

Известен также под именем **Wrapper** (**Обертка**).

Частота использования: выше средней.

Назначение: преобразует интерфейс одного класса в интерфейс другого, который ожидают клиенты. Адаптер обеспечивает совместную работу классов с несовместимыми интерфейсами, которая без него была бы невозможна.

Участники:

• *Target* (*Целевой класс*) — определяет зависящий от предметной области интерфейс, которым пользуется Client;

• *Client* (*Клиент*) — вступает во взаимоотношения с объектами, удовлетворяющими интерфейсу Target;

• *Adaptee* (*Адаптируемый класс*) — определяет существующий интерфейс, который нуждается в адаптации;

• *Adapter* (*Адаптер*) — адаптирует интерфейс Adaptee к интерфейсу Target.

В данном задании рассматривается вариант адаптера, использующий композицию. Такой вариант называется *адаптером объекта*, поскольку адаптируется *объект* типа Adaptee, входящий в виде ссылочного поля в класс Adapter.

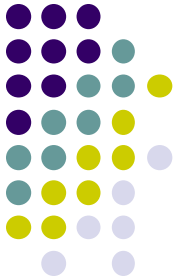
Задание 1. Абстрактный класс `Target` содержит три абстрактных метода: `GetA`, `GetB` и `Request` (не имеют параметров, возвращают значение целого типа). Класс `ConcreteTarget` является потомком класса `Target`; он содержит поля `a` и `b` целого типа, которые инициализируются в конструкторе, имеющем одноименные параметры. Методы `GetA` и `GetB` класса `ConcreteTarget` возвращают значения полей `a` и `b` соответственно, а метод `Request` возвращает сумму этих полей.

Класс `Adaptee` содержит два целочисленных поля `a` и `b`, конструктор с параметрами `a` и `b`, задающий значения этих полей, метод `GetAll`, возвращающий текущие значения полей (либо с помощью выходных параметров, либо с помощью возвращаемого значения — массива или кортежа), и метод `SpecificRequest` без параметров, возвращающий произведение полей `a` и `b`.

Реализовать класс `Adapter`, адаптирующий класс `Adaptee` к интерфейсу класса `Target`. Класс должен быть адаптером объекта: он порождается от класса `Target` и включает ссылку `ad` на экземпляр адаптируемого объекта `Adaptee`. Ссылка `ad` инициализируется в конструкторе класса `Adapter` путем вызова конструктора класса `Adaptee` с параметрами `a` и `b`, совпадающими с одноименными параметрами конструктора класса `Adapter`. Метод `Request` класса `Adapter` должен вызывать метод `SpecificRequest` объекта `ad`, а методы `GetA` и `GetB` — возвращать значения полей `a` и `b` объекта `ad`, используя его метод `GetAll`.

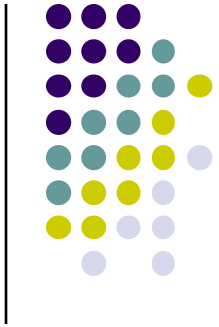
Тестирование разработанной системы классов. Дано целое число N (≤ 6) и набор из N троек (C, A, B), где C является символом «+» или «*», а элементы A и B являются целыми числами. Создать структуру данных (например, массив) с элементами-ссылками типа `Target` и заполнить ее объектами типа `ConcreteTarget` (для троек с символом «+») и `Adapter` (для троек с символом «*») с полями, равными A и B. Перебирая элементы полученного набора *в обратном порядке*, вывести для каждого из них значения полей `a`, `b` и результат выполнения метода `Request`.

Особенности заданий



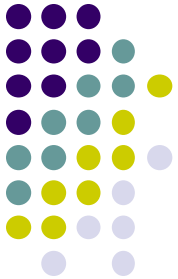
- Первое задание содержит описание **модельной системы классов**, связанной с обработкой простых наборов данных.
- Для большинства паттернов предусмотрено два задания, причем во втором задании обычно рассматривается задача с **реальным содержанием**.
- Например, для паттерна **Decorator** строятся классы для генерации суперпозиции различных функций и вычисления их значений, а для паттерна **Strategy** – классы-валидаторы, обеспечивающие проверку правильности различных наборов текстовых данных.

Файлы дополнений



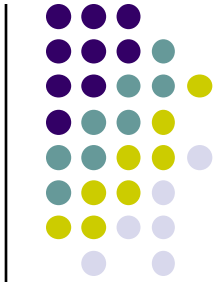
- **Файлы дополнений** – это вспомогательные файлы задачника, содержащие дополнительные примечания и тексты индивидуальных программ-заготовок на различных языках программирования для каждого задания.
- Файлы дополнений разработаны для всех групп задачника PT for OOP.
- Они включают тексты программ-заготовок на языках PascalABC.NET, C++, C#, Java, Python, Ruby, Julia.

Содержание файлов дополнений



- Для языков со статической типизацией (PascalABC.NET, C++, C#, Java) в программы-заготовки включались описания **абстрактных базовых классов**, для языков с динамической типизацией (Python, Ruby) приводились фрагменты описаний **конкретных классов**.
- В ряде случаев добавлялись комментарии, связанные с **особенностями реализации паттерна** на конкретном языке.
- Для всех заданий вводной группы OOP0Begin в файлы дополнений были включены комментарии, описывающие **особенности объектных моделей** каждого языка.

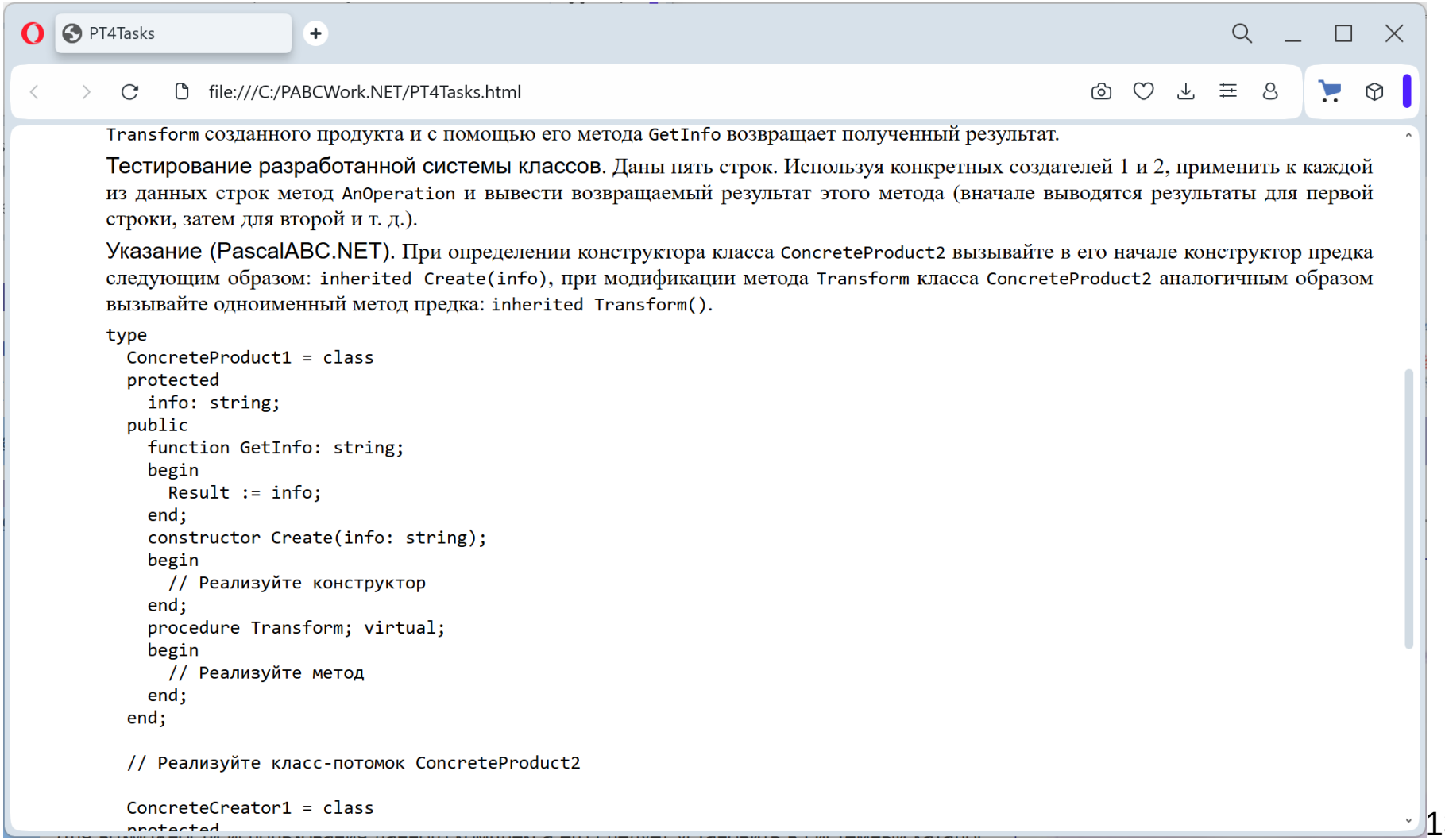
Формат файлов дополнений



- Файлы дополнений включают тексты примечаний, фрагменты программного кода и управляющие команды.

```
[2]
[text]
%{S}Указание (PascalABC.NET).%{s} При определении конструктора
   класса %{M}ConcreteProduct2%{m} вызывайте в его начале
   конструктор предка следующим образом: %{M}inherited
   Create(info)%{m}, при модификации метода %{M}Transform%{m}
   класса %{M}ConcreteProduct2%{m} аналогичным образом
   вызывайте одноименный метод предка: %{M}inherited
   Transform()%{m}.
[code]
type
  ConcreteProduct1 = class
  protected
    info: string;
```

Файлы дополнений в режиме html-документа



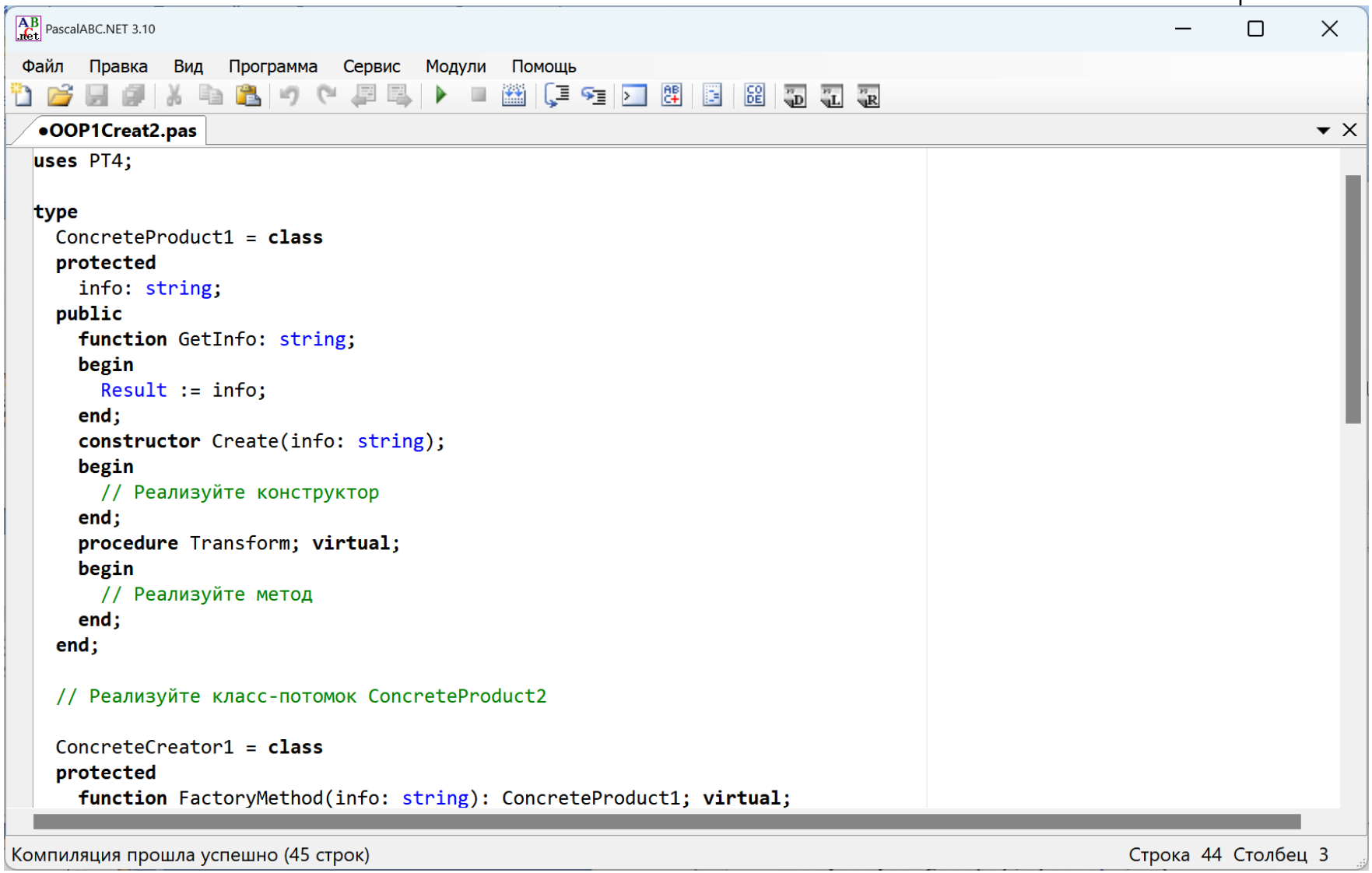
`Transform` созданного продукта и с помощью его метода `GetInfo` возвращает полученный результат.

Тестирование разработанной системы классов. Даны пять строк. Используя конкретных создателей 1 и 2, применить к каждой из данных строк метод `AnOperation` и вывести возвращаемый результат этого метода (вначале выводятся результаты для первой строки, затем для второй и т. д.).

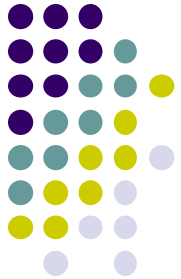
Указание (PascalABC.NET). При определении конструктора класса `ConcreteProduct2` вызывайте в его начале конструктор предка следующим образом: `inherited Create(info)`, при модификации метода `Transform` класса `ConcreteProduct2` аналогичным образом вызывайте одноименный метод предка: `inherited Transform()`.

```
type
  ConcreteProduct1 = class
  protected
    info: string;
  public
    function GetInfo: string;
    begin
      Result := info;
    end;
    constructor Create(info: string);
    begin
      // Реализуйте конструктор
    end;
    procedure Transform; virtual;
    begin
      // Реализуйте метод
    end;
  end;

  // Реализуйте класс-потомок ConcreteProduct2

  ConcreteCreator1 = class
  protected
```

Заготовка на языке PascalABC.NET

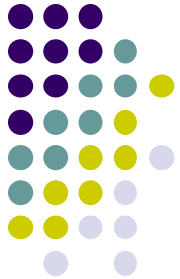


Дополнительные возможности



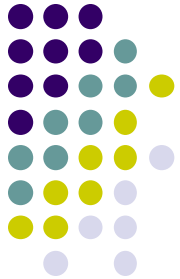
- Файлы дополнений являются обычными текстовыми файлами, поэтому любой преподаватель может их корректировать, добавляя или изменяя указания или тексты программ-заготовок для различных языков.
- В варианте файлов дополнений для языка C++ предполагается, что для работы с объектами используются **умные указатели** shared_ptr<T> и weak_ptr<T>.

Результаты использования задачника в учебных курсах



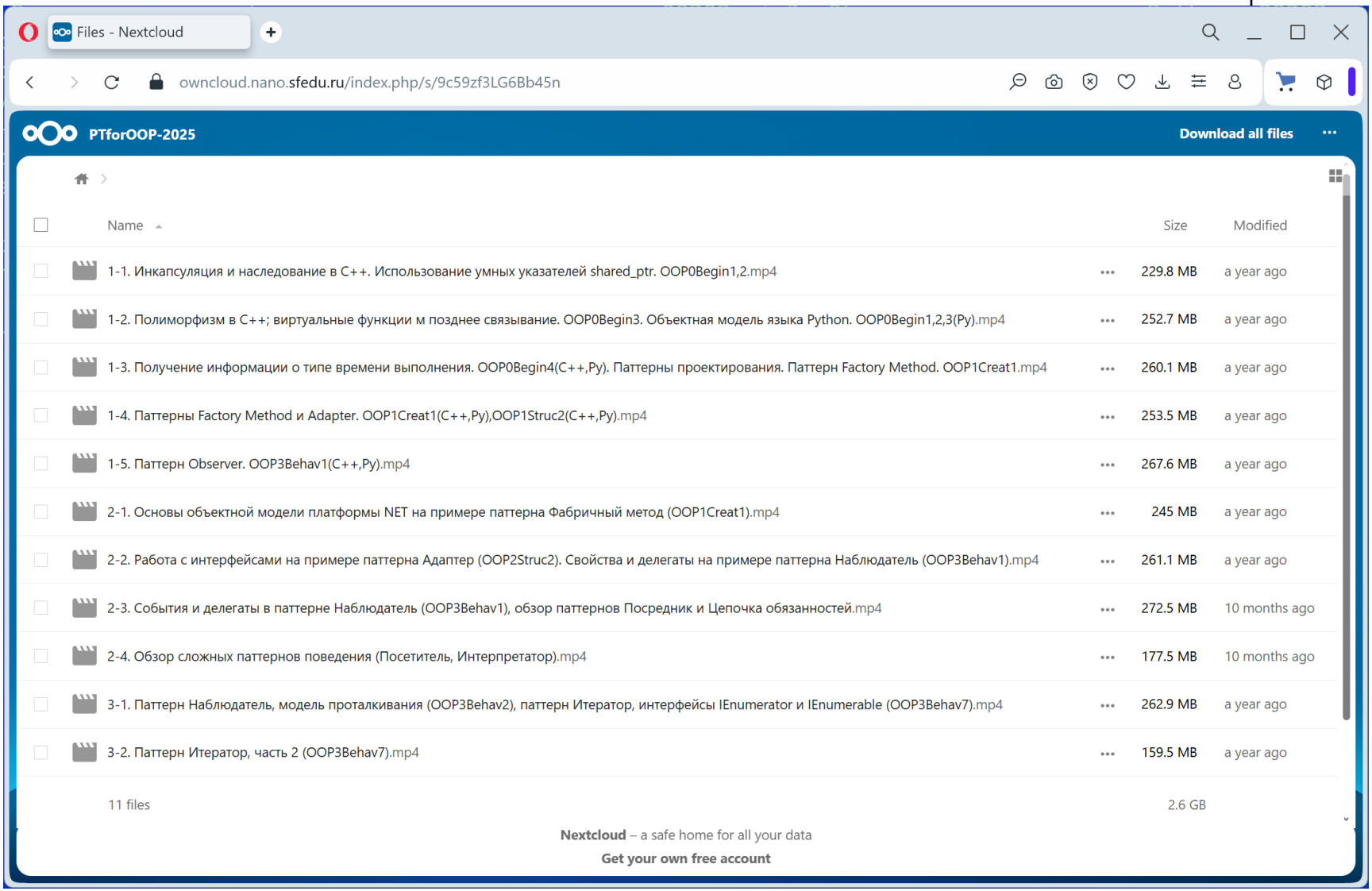
- Задачник использовался при проведении **лабораторных занятий** по дисциплинам:
  - «**Объектно-ориентированное программирование**» (C++, Python) и «**Языки программирования**» (C#), 3 курс факультета вычислительной математики и кибернетики Университета МГУ-ППИ в Шэньчжэне)
  - «**Языки программирования**» (C#), 1 год магистратуры мехмата ЮФУ, программа «Разработка компьютерных игр и мобильных приложений»
- Это позволило существенно увеличить количество выполненных заданий и упростило их проверку.
- Задачник также применялся на **лекциях**: обсуждение формулировок задач, программ-заготовок и тестовых данных позволило осветить различные аспекты рассматриваемого паттерна и продемонстрировать его работу на примерах.

Видеозаписи занятий, на которых использовался задачник PTforOOP

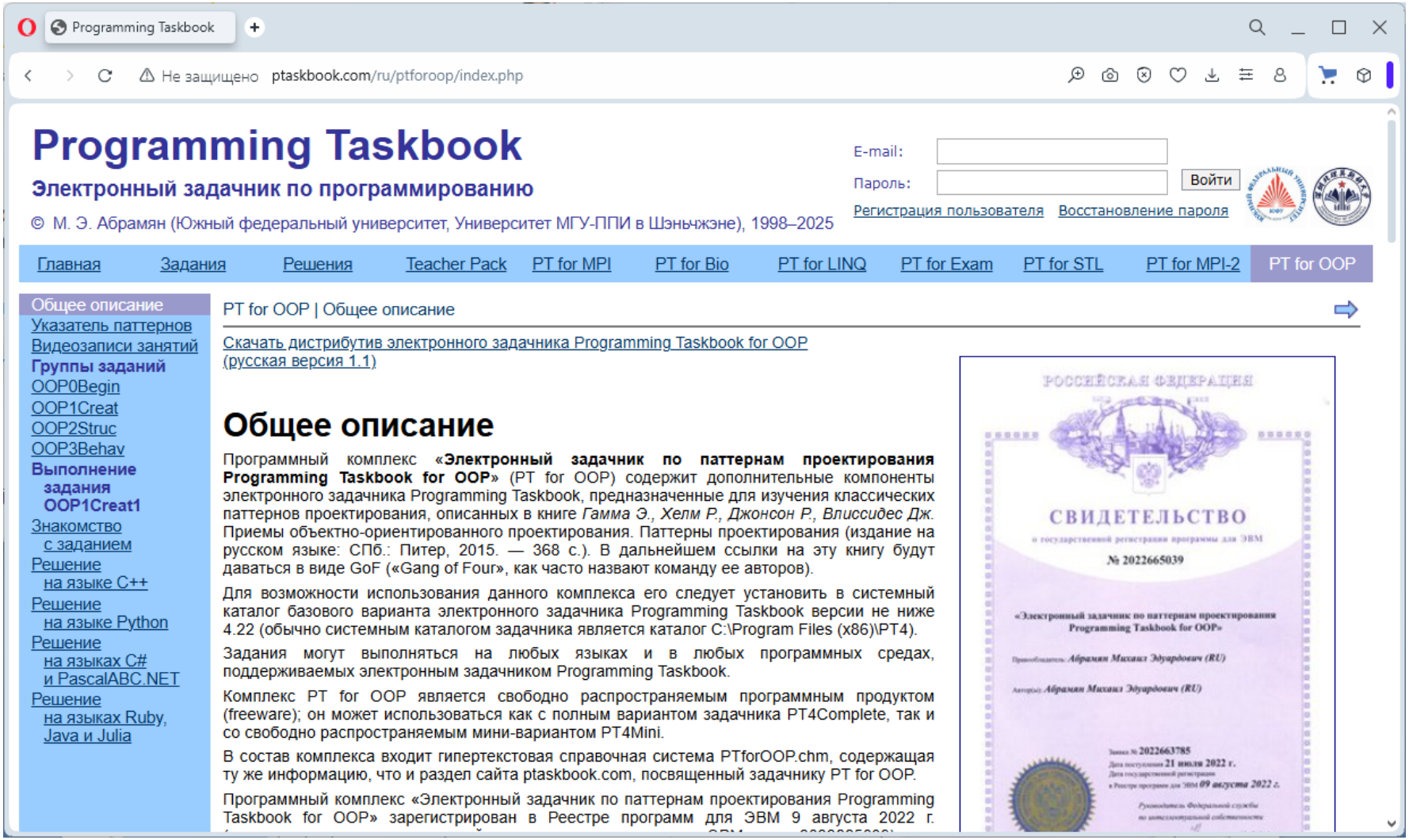


- Ссылка на облачное хранилище (ссылка имеется на сайте задачника): https://owncloud.nano.sfedu.ru/index.php/s/9c59zf3LG6Bb45n
- Общая продолжительность видеозаписей – около 15 часов
- Часть 1. Знакомство с ООП на примерах языков C++ и Python
  - Инкапсуляция и наследование в C++ (OOP0Begin1,2)
  - Полиморфизм в C++: виртуальные функции м позднее связывание (OOP0Begin3). Объектная модель языка Python (OOP0Begin1,2,3)
  - Получение информации о типе времени выполнения (OOP0Begin4). Паттерны проектирования: обзор. Паттерн «Фабричный метод» (OOP1Creat1)
  - Паттерны «Фабричный метод» и «Адаптер» (OOP1Creat1,OOP1Struc2)
  - Паттерн «Наблюдатель» (OOP3Behav1)
- Часть 2. Объектная модель языков платформы .NET на примере C#
  - Основы объектной модели платформы .NET на примере паттерна «Фабричный метод»
  - Работа с интерфейсами на примере паттерна «Адаптер». Свойства и делегаты на примере паттерна «Наблюдатель»
  - События в паттерне «Наблюдатель», обзор паттернов «Посредник» и «Цепочка обязанностей»
  - Обзор паттернов «Посетитель» и «Интерпретатор»
  - Паттерн «Наблюдатель», модель проталкивания (OOP3Behav2), паттерн «Итератор», интерфейсы IEnumerator и IEnumerable (OOP3Behav7)

Страница облачного хранилища с видеозаписями занятий



Раздел сайта ptaskbook.com с описанием задачника PT for OOP



Спасибо за внимание

Электронный задачник Programming Taskbook for OOP:
http://ptaskbook.com/ru/ptforoop/