ISSN 0234-0453 • INFORMATICS AND EDUCATION • 2023 • Volume 38 No 2

DOI: 10.32517/0234-0453-2023-38-2-35-46

ЭЛЕКТРОННЫЙ ЗАДАЧНИК ПО ПАТТЕРНАМ ПРОЕКТИРОВАНИЯ: РЕАЛИЗАЦИЯ И ИСПОЛЬЗОВАНИЕ

М. Э. Абрамян[1,2] ✉

[1] ***Университет МГУ-ППИ в Шэньчжэне, Китай***

[2] ***Южный федеральный университет, г. Ростов-на-Дону, Россия***

✉ *mabr@sfedu.ru*

Аннотация

Работа посвящена проблемам, связанным с организацией практической составляющей учебных курсов по объектно-ориентированному программированию (ООП). Дается обзор существующих обучающих ресурсов по курсам ООП и отмечается отсутствие в них учебных заданий по паттернам проектирования — одному из основных компонентов современной теории ООП. Для решения отмеченной проблемы предлагается использовать специализированный обучающий ресурс — электронный задачник по паттернам объектно-ориентированного проектирования. Подробно описываются особенности задачника по паттернам проектирования, и приводится пример его реализации: задачник Programming Taskbook for OOP, включающий в себя задания, связанные с 21 классическим паттерном проектирования, а также вводную группу заданий, посвященную принципам и приемам ООП.

Новизна описанного подхода состоит в том, что, в отличие от традиционных задачников, Programming Taskbook for OOP не только содержит формулировки заданий, но и предоставляет учебной программе исходные данные и обеспечивает автоматическую проверку правильности предложенного решения. Еще одной важной особенностью задачника является его универсальность: задания можно выполнять на любых языках и в разных средах программирования, что позволяет легко адаптировать задачник для использования в различных учебных курсах уровня бакалавриата и магистратуры. Обсуждаются результаты применения задачника при проведении курсов по ООП.

Ключевые слова: электронный задачник, объектно-ориентированное программирование, паттерны проектирования.

Для цитирования:

Абрамян М. Э. Электронный задачник по паттернам проектирования: реализация и использование. *Информатика и образование.* 2023;38(2):35–46. DOI: 10.32517/0234-0453-2023-38-2-35-46

ELECTRONIC PROBLEM BOOK ON DESIGN PATTERNS: IMPLEMENTATION AND USE

M. E. Abramyan[1,2] ✉

[1] ***Shenzhen MSU-BIT University, China***

[2] ***Southern Federal University, Rostov-on-Don, Russia***

✉ *mabr@sfedu.ru*

Abstract

The article is dedicated to the problems associated with organizing the practical component of object-oriented programming (OOP) courses. A review of current learning resources for OOP courses reveals a lack of learning assignments related to design patterns, which are essential components of modern OOP theory. To solve this problem, the author proposes using a specialized training resource — an electronic problem book on object-oriented design patterns. The features of the design patterns e-taskbook are described in detail and an example of its implementation is given: the Programming Taskbook for OOP, which includes tasks related to 21 classic design patterns, as well as an introductory group of tasks dedicated to the principles and techniques of OOP.

The article emphasizes the novelty of this approach, noting that unlike traditional problem books, the Programming Taskbook for OOP not only contains the wording of tasks, but also provides the courseware with input data and automatic verification of the proposed solution. Another important feature of the book is its versatility: the tasks can be performed in different languages and different programming environments, which allows to easily adapt the book for use in various undergraduate and graduate courses. Finally, the author presents his experience using the Programming Taskbook in OOP courses.

Keywords: electronic problem book, object-oriented programming, design patterns.

For citation:

Abramyan M. E. Electronic problem book on design patterns: Implementation and use. *Informatics and Education.* 2023;38(2):35–46. (In Russian.) DOI: 10.32517/0234-0453-2023-38-2-35-46

ISSN 0234-0453 • ИНФОРМАТИКА И ОБРАЗОВАНИЕ • 2023 • Том 38 № 2

1. Паттерны проектирования и их изучение

Паттерн проектирования, или *шаблон проектирования* (*англ.* design pattern), в разработке программного обеспечения — повторяемая архитектурная конструкция, представляющая собой решение проблемы проектирования в рамках некоторого часто возникающего контекста*. Применение паттернов проектирования позволяет упростить и унифицировать разработку программ и повысить их надежность. Паттерны проектирования базируются на принципах объектно-ориентированного программирования (ООП) и развивают эти принципы для решения широкого набора стандартных задач.

Первое систематическое описание объектно-ориентированных паттернов проектирования было дано в книге «Приемы объектно-ориентированного проектирования. Паттерны проектирования» [1] четырех авторов (Эрих Гамма, Ричард Хелм, Ральф Джонсон, Джон Влиссидес — так называемая «банда четырех», Gang of Four). В этой книге, вышедшей в 1995 году, содержалось подробное описание 23 стандартных паттернов, снабженных именами и разбитых на три группы. Описание сопровождалось примерами реализации на C++, Smalltalk и некоторых других языках. В дальнейшем паттерны ООП успешно применялись и для других объектно-ориентированных языков программирования; в ряде книг рассматривались особенности реализации паттернов для конкретных языков, таких как Java [2], C# [3], Python [4], Ruby [5] и др.

В книге Мэтта Вайсфельда [6], посвященной базовым концепциям ООП, отмечается, что «паттерны проектирования являются одним из важных шагов, сделанных за последнее время для развития объектно-ориентированного подхода в разработке программного обеспечения». Следует отметить, что именно изучение паттернов проектирования позволяет в полной мере понять базовые принципы ООП и научиться их применять в различных стандартных ситуациях. Вскоре после выхода книги «банды четырех», которая в наше время по праву считается классикой, паттерны проектирования были включены в международный стандарт образования Computer Science Curricula [7], а вопросы, связанные с методикой обучения паттернам проектирования, стали активно обсуждаться (см., например, [8–11]).

Однако в имеющихся русскоязычных пособиях-практикумах по ООП паттерны проектирования практически никогда не рассматриваются. Обычно практикумы предлагают лабораторные работы по базовым принципам ООП (к которым относятся инкапсуляция, наследование, полиморфизм), а также включают в себя дополнительные темы, связанные с особенностями изучаемого языка программирования (как правило, это либо Pascal [12–13], либо C++ [14–15], либо C# [16–17]). В частности, для C++ изучается перегрузка операций и шаблонные классы, а для C# — интерфейсы и делегаты. Лишь в практикуме А. В. Щербакова [17] упоминается единственный паттерн проектирования Singleton — в связи с изучением статических членов классов.

Немногочисленные электронные практикумы по ООП [18] фактически представляют собой гипертекстовые аналоги «бумажных» практикумов, дополненные наборами обычных тестов, и также не содержат разделов, связанных с паттернами проектирования.

Разнообразные материалы по паттернам проектирования, в том числе русскоязычные, широко представлены в интернете. Однако подобные источники либо не включают в себя заданий по разработке программ, использующих паттерны, либо содержат простые упражнения, на основе которых нельзя построить практическую часть учебного курса. А среди открытых или платных онлайн-курсов, посвященных паттернам**, практическая часть обычно ограничивается наборами тестов или небольшим числом программных разработок, не охватывающих все изучаемые паттерны. Стоит также отметить, что формат онлайн-курсов, даже содержащих элементы проверяемых заданий, затрудняет их адаптацию к существующим вузовским курсам и их практическим составляющим.

Таким образом, можно выделить следующую проблему: с одной стороны, весьма желательно включить в практическую часть курса по ООП раздел, посвященный паттернам проектирования, а с другой стороны, отсутствуют образовательные ресурсы, которые позволили бы сравнительно легко выполнить требуемую адаптацию практической части курса по ООП.

Одним из путей решения отмеченной проблемы является разработка электронного обучающего комплекса — *электронного задачника по паттернам проектирования*, который включал бы в себя репрезентативный набор заданий по разработке паттернов и обеспечивал дополнительные возможности, связанные с выполнением этих заданий на различных языках.

Электронный задачник предоставляет учебной программе наборы исходных данных, проверяет правильность полученных результатов, диагностирует различные виды ошибок в программах и отображает на экране все данные, связанные с заданием [19]. Возможности электронного задачника упрощают для обучающихся процесс выполнения заданий и облегчают для преподавателя проверку полученных

* Шаблон проектирования. *Свободная энциклопедия «Википедия»*. https://ru.wikipedia.org/wiki/Шаблон_проектирования

** Онлайн-курс «ООП и паттерны проектирования в Python». *Class Central*. https://www.classcentral.com/course/oop-patterns-python-11232; Онлайн-курс «Шаблоны проектирования». *Coursera*. https://www.coursera.org/learn/design-patterns; Онлайн-курс «Паттерны в объектно-ориентированном программировании». *Бауманский учебный центр «Специалист»*. https://www.specialist.ru/course/pattern

решений. Примером электронного задачника является универсальный задачник по программированию Programming Taskbook*, на основе которого можно разрабатывать *расширения* — специализированные электронные задачники, посвященные изучению различных алгоритмов, библиотек и технологий программирования [20]. Одним из таких расширений является задачник Programming Taskbook for OOP** — задачник по паттернам проектирования, которому посвящена настоящая статья.

2. Общее описание электронного задачника Programming Taskbook for OOP

Электронный задачник по паттернам проектирования Programming Taskbook for OOP*** включает в себя 44 задания. Вводная группа OOP0Begin содержит 6 заданий, демонстрирующих принципы и приемы объектно-ориентированного программирования. Основную часть задачника составляют упражнения, посвященные паттернам проектирования. В них рассматривается 21 классический паттерн (из 23 паттернов, описанных в книге «Приемы объектно-ориентированного проектирования. Паттерны проектирования» [1]). С учетом принятой классификации эти задания разбиты на три группы:

- OOP1Creat (10 заданий, связанных с 5 *порождающими паттернами*);
- OOP2Struc (11 заданий, связанных с 6 *структурными паттернами*);
- OOP3Behav (17 заданий, связанных с 10 *паттернами поведения*).

В пределах каждой группы задания распределены в соответствии с частотой использования изучаемых паттернов; кроме того, первыми в каждой группе рассматриваются наиболее простые и распространенные паттерны, выделенные в особое подмножество [1]. Информация о частоте использования паттернов взята из книги «Design Patterns via C#. Приемы объектно-ориентированного проектирования» [3]. Также принималась во внимание степень детализации, с которой паттерны описывались в книге «Head First. Паттерны проектирования» [2].

Ниже приводится список паттернов, упорядоченный по номерам связанных с ними учебных заданий. В скобках указывается частота использования паттерна (по шкале от 1 — «низкая частота использования» до 5 — «высокая частота использования»). Если он относится к особой группе простых и распространенных паттернов, выделенной в [1], это отмечается символом «*»; если паттерну была посвящена отдельная глава в [2], это отмечается символом «+» (если символ «+» отсутствует, то это означает, что паттерн лишь кратко упомянут в последней главе книги [2]). После информации, приводимой в скобках, указываются номера заданий, связанных с данным паттерном.

Порождающие паттерны (группа OOP1Creat)
Подгруппа 1
- Factory Method — Фабричный метод (5*+) 1, 2, 3
- Abstract Factory — Абстрактная фабрика (5*+) 4, 5

Подгруппа 2
- Singleton — Одиночка (4+) 6
- Prototype — Прототип (3) 7, 8
- Builder — Строитель (2) 9, 10

Структурные паттерны (группа OOP2Struc)
Подгруппа 1
- Adapter — Адаптер (4*+) 1, 2, 3, 4
- Composite — Компоновщик (4*+) 5, 6
- Decorator — Декоратор (3*+) 7, 8

Подгруппа 2
- Proxy — Заместитель (4+) 9
- Bridge — Мост (3) 10
- Flyweight — Приспособленец (1) 11

Паттерны поведения (группа OOP3Behav)
Подгруппа 1
- Observer — Наблюдатель (5*+) 1, 2
- Strategy — Стратегия (4*+) 3, 4
- Template Method — Шаблонный метод (3*+) 5, 6

Подгруппа 2
- Iterator — Итератор (5+) 7
- Command — Команда (4+) 8, 9
- State — Состояние (3+) 10, 11

Подгруппа 3
- Mediator — Посредник (2) 12
- Chain of Responsibility — Цепочка обязанностей (2) 13, 14
- Visitor — Посетитель (1) 15
- Interpreter — Интерпретатор (1) 16, 17

Таким образом, первая подгруппа в каждой группе связана с паттернами, особо выделенными в книге «Приемы объектно-ориентированного проектирования. Паттерны проектирования» [1]; обычно это и есть наиболее часто используемые паттерны данной категории. Завершающая подгруппа содержит редко используемые и при этом, как правило, сложные паттерны. Средняя подгруппа группы OOP3Behav занимает промежуточное положение.

Подобная структура задачника упрощает его применение в различных курсах, посвященных объ-

* Электронный задачник по программированию Programming Taskbook. http://ptaskbook.com/ru/index.php

** Электронный задачник по паттернам проектирования Programming Taskbook for OOP. http://ptaskbook.com/ru/ptforoop/

*** Свидетельство о государственной регистрации программы для ЭВМ № 2022665039 Российская Федерация. Электронный задачник по паттернам проектирования Programming Taskbook for OOP: № 2022663785: заявлено 21.07.2022: опубликовано 09.08.2022 / Абрамян М. Э.; правообладатель Абрамян М. Э. Зарегистрировано в реестре программ для ЭВМ. EDN: RBVFAY. Режим доступа: https://new.fips.ru/registers-doc-view/fips_servlet?DB=EVM&DocNumber=2022665039&TypeFile=html

ISSN 0234-0453 • ИНФОРМАТИКА И ОБРАЗОВАНИЕ • 2023 • Том 38 № 2

ектно-ориентированному программированию. Если объем курса не позволяет изучить паттерны в полном объеме, можно ограничиться лишь основными, наиболее распространенными.

Поскольку задачник по паттернам ООП реализован как расширение универсального задачника Programming Taskbook, все входящие в него задания можно выполнять на любых языках программирования и во всех интегрированных средах разработки, поддерживаемых универсальным задачником (в настоящее время задачник поддерживает языки Pascal, C++, Java, C#, VB.NET, F#, Python, Ruby, Julia). Это позволяет использовать его в курсах по ООП, ориентированных на различные языки программирования, а также облегчает разработку практической части курса по ООП, в которой рассматривается *несколько* объектно-ориентированных языков (например, C++ и Python, C++ и C# и т. д.).

При одновременном рассмотрении нескольких языков может оказаться полезной еще одна особенность задачника: возможность выполнять задания на различных языках (C++, Java, C#, Python, Ruby, Julia) в *одном и том же* универсальном редакторе программного кода Visual Studio Code [21], который в настоящее время получил очень широкое распространение.

В задачнике Programming Taskbook for OOP активно используются *файлы дополнений*, возможность работы с которыми появилась в последних версиях универсального задачника Programming Taskbook [22]. В то время как формулировки заданий содержат общее описание паттерна, связанной с ним системы классов и наборов тестовых данных, файлы дополнений включают в себя информацию об особенностях реализации паттерна для *конкретного языка программирования*. Кроме того, применение файлов дополнений позволяет создавать особые программы-заготовки для каждого задания на различных языках. В задачнике Programming Taskbook for OOP подобные программы-заготовки с фрагментами описания классов, связанных с изучаемым паттерном, предусмотрены для основных современных объектно-ориентированных языков: C++, C#, Java, Python и Ruby. Файлы дополнений являются обычными текстовыми файлами, включающими в себя небольшой набор специальных управляющих команд, поэтому любой преподаватель может их корректировать, добавляя или изменяя указания или тексты программ-заготовок для различных языков.

3. Особенности заданий, включенных в вводную группу OOP0Begin

В первых трех заданиях вводной группы OOP0Begin рассматриваются базовые принципы ООП:

- инкапсуляция;
- наследование;
- полиморфизм.

Следующие три задания описывают дополнительные приемы и возможности ООП:

- получение информации о типе времени выполнения (RTTI);
- генерация и обработка исключений;
- работа с шаблонными классами.

На базовых принципах ООП основаны практически все паттерны проектирования; дополнительные приемы ООП не связаны с паттернами непосредственно, но полезны при разработке различных классов и их последующем использовании.

Каждое задание вводной группы снабжено не только программами-заготовками на языках C++, C#, Java, Python и Ruby, но и подробными примечаниями, в которых описываются особенности реализации соответствующего принципа или приема для конкретного языка.

Задания группы OOP0Begin образуют *серию*, в которой последовательно выполняются разработка и модификация иерархии из двух классов, связанных с реализацией простой структуры данных — стека. В качестве основы для выполнения очередного задания может использоваться программа с решением одного из предыдущих заданий.

4. Особенности заданий, связанных с изучением паттернов

Чтобы наглядно выявить взаимосвязи между классами, реализуемые в рамках того или иного паттерна, достаточно рассмотреть сравнительно простые классы, содержащие небольшой набор числовых или строковых полей и выполняющие простые действия по их обработке. Простота описанных в задании классов нередко приводит к тому, что для такого задания можно легко написать программу, которая будет правильно преобразовывать исходные данные в результирующие, *не применяя методы ООП*. При этом может возникнуть ощущение, что использование паттерна лишь *усложняет* решение поставленной задачи. Однако необходимо учитывать, что сравнительная простота требуемого в заданиях преобразования исходных данных обусловлена лишь необходимостью сосредоточить внимание на *ключевых особенностях* того или иного паттерна и главной целью при выполнении задания является не получение «максимально быстрым способом» правильного набора результатов, а реализация правильно функционирующей системы классов, связанной с изучаемым паттерном.

В начале первого задания для каждого паттерна приводятся связанные с ним общие сведения: варианты наименования, частота использования, назначение, — а также указываются участники и их роли в реализации паттерна. Кроме того, формулировка первого задания для каждого паттерна включает в себя рисунок с диаграммой классов.

В первом задании, связанном с каждым паттерном, выбираются такие имена основных классов, их ключевых полей и методов, которые совпадают

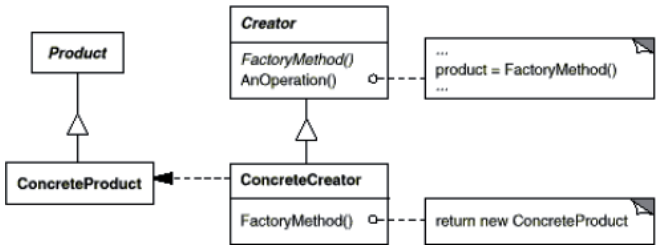
с «абстрактными» именами, используемыми при словесном описании паттерна и в его диаграмме классов. Это должно облегчить понимание взаимосвязи различных компонентов паттерна и упростить реализацию требуемой системы классов.

Несмотря на наличие рисунка с диаграммой классов и заготовки программы, уже содержащей описания некоторых классов и их членов, формулировка задания подробно описывает структуру каждого класса и связи между ними. Такое описание, с одной стороны, должно дополнительно прояснить смысл приводимой диаграммы классов, а с другой стороны, оно учитывает возможность выполнения задания на языке, для которого не предусмотрена автоматическая генерация развернутой программы-заготовки с фрагментами реализуемой системы классов.

В конце формулировки каждого задания описываются исходные данные, которые надо обработать с применением реализованной системы классов, и результаты, которые требуется получить в результате обработки исходных данных. Эта часть формулировки снабжается подзаголовком «Тестирование разработанной системы классов».

В качестве примера приведем задание OOP1Creat1, посвященное паттерну Factory Method. Это один из наиболее простых и часто используемых порождающих паттернов, наглядно иллюстрирующих принцип полиморфизма.

4.1. Задание OOP1Creat1



Factory Method (**Фабричный метод**) — порождающий паттерн.

Известен также под именем Virtual Constructor (Виртуальный конструктор).

Частота использования: высокая.

Назначение: определяет интерфейс для создания объекта, но оставляет подклассам решение о том, какой класс инстанцировать. Фабричный метод позволяет классу делегировать инстанцирование подклассам.

Участники:

- *Product* (*Продукт*) — определяет интерфейс объектов, создаваемых фабричным методом;
- *ConcreteProduct* (*Конкретный продукт*) — реализует интерфейс Product;
- *Creator* (*Создатель*) — объявляет фабричный метод, возвращающий объект типа Product; может также определять реализацию фабричного метода по умолчанию, возвращающую некоторый конкретный продукт; реализует методы, в которых используется объект Product, созданный фабричным методом;
- *ConcreteCreator* (*Конкретный создатель*) — замещает фабричный метод для создания конкретного продукта.

Задание 1. Реализовать две иерархии классов, в одну из которых входят абстрактный создатель Creator и два конкретных создателя ConcreteCreator1 и ConcreteCreator2, а в другую — абстрактный продукт Product и два конкретных продукта ConcreteProduct1 и ConcreteProduct2.

Абстрактный класс Product содержит два абстрактных метода, связанных с получением и преобразованием строки: метод GetInfo без параметров, возвращающий строку, и метод Transform без параметров, который ничего не возвращает. Классы ConcreteProduct1 и ConcreteProduct2 содержат строковое поле info, которое инициализируется в конструкторе с помощью одноименного параметра, после чего в конструкторе класса ConcreteProduct1 поле info преобразуется к нижнему регистру, а в конструкторе класса ConcreteProduct2 — к верхнему. Метод GetInfo в каждом подклассе возвращает текущее значение поля info, а метод Transform преобразует это поле следующим образом: для ConcreteProduct1 он добавляет дополнительный пробел после каждого непробельного символа поля info (кроме его последнего символа), а для ConcreteProduct2 он добавляет два дополнительных символа * (звездочка) после каждого символа, отличного от звездочки (кроме последнего символа).

Абстрактный класс Creator содержит абстрактный фабричный метод FactoryMethod(info) со строковым параметром info, возвращающий ссылку на объект Product. Этот метод определяется в классах ConcreteCreator1 и ConcreteCreator2, причем фабричный метод класса ConcreteCreator1 создает объект типа ConcreteProduct1, а фабричный метод класса ConcreteCreator2 создает объект типа ConcreteProduct2; в любом случае конструктору создаваемого объекта передается параметр info фабричного метода.

В абстрактном классе Creator дополнительно определить метод AnOperation(info), который создает продукт с помощью фабричного метода, передавая ему параметр info, дважды вызывает метод Transform созданного продукта и с помощью его метода GetInfo возвращает полученный результат. Использование фабричного метода в методе AnOperation приводит к тому, что выполнение метода AnOperation в подклассах класса Creator дает различные результаты, зависящие от свойств создаваемых продуктов, причем такое поведение реализуется *без изменения кода* метода AnOperation.

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.).

* * *

Напомним, что при тестировании предложенного решения электронный задачник генерирует и пересылает учебной программе набор исходных данных и проверяет правильность полученных ею результатов (далее, в разделе 6, будет приведено краткое описание процесса выполнения задания OOP1Creat1).

Для 13 паттернов (в частности, для всех паттернов, включенных в первую подгруппу каждой груп-

ISSN 0234-0453 • ИНФОРМАТИКА И ОБРАЗОВАНИЕ • 2023 • Том 38 № 2

пы) предусмотрено более одного задания (обычно два). Второе задание уже не сопровождается диаграммой классов; в нем, как правило, используются названия классов и их членов, соответствующие не общему описанию паттерна, а *специфике рассматриваемой задачи*. Во втором задании обычно делается попытка проиллюстрировать применение изучаемого паттерна к конкретной (хотя и простой) задаче *с реальным наполнением* (в то время как в первом задании описывается формальная система классов с несколько надуманной схемой взаимодействия, которая тем не менее наглядно демонстрирует ключевые особенности изучаемого паттерна). Подобная комбинация абстрактных и конкретных заданий, по мнению автора, способствует более глубокому усвоению идей, лежащих в основе каждого паттерна. Ниже приводятся примеры «конкретных» вторых заданий (в скобках указывается имя задания и связанный с ним паттерн):

- формирование зоопарка из животных определенной группы (OOP1Creat3, Factory Method);
- конструирование набора элементов управления, оформленных в различном стиле (OOP1Creat5, Abstract Factory);
- клонирование графических примитивов (OOP1Creat8, Prototype);
- генерация идентификаторов по их строковым описаниям с применением соглашений, принятых для различных языков программирования (OOP1Creat10, Builder);
- адаптация интерфейса класса, описывающего текстовый объект, к интерфейсу графических объектов (OOP2Struc3 и OOP2Struc4, Adapter);
- вычисление суммарной стоимости сложных устройств, состоящих из иерархически организованных компонентов (OOP2Struc6, Composite);
- построение суперпозиций различных функций и вычисление их значений (OOP2Struc8, Decorator);
- реализация набора классов-валидаторов и их использование для проверки правильности набора текстовых данных (OOP3Behav4, Strategy);
- реализация алгоритмов поиска минимумов и максимумов в наборе данных при применении различных правил сравнения элементов (OOP3Behav6, Template Method);
- реализация макрокоманд и системы отмены и восстановления предыдущих операций (OOP3Behav9, Command);
- моделирование работы автомата по продаже шариков (OOP3Behav11, State);
- конструирование символьной строки на основе грамматики, включающей в себя условные и циклические конструкции (OOP3Behav17, Interpreter).

В качестве примера конкретного задания приведем задание OOP1Creat3, связанное с ранее описанным паттерном Factory Method.

4.2. Задание OOP1Creat3

Factory Method (**Фабричный метод**) — порождающий паттерн.

Задание 3. Реализовать иерархию классов-животных с абстрактным предком Animal, содержащим метод GetInfo, который возвращает строковое значение, и шестью конкретными потомками: Lion, Tiger, Leopard (кошачьи, cats), Gorilla, Orangutan, Chimpanzee (человекообразные обезьяны, apes). Каждый конкретный класс содержит строковое поле name (имя животного), которое определяется в его конструкторе с помощью одноименного параметра. Метод GetInfo возвращает имя класса и имя животного, разделенные пробелом, например, Lion Tom.

Реализовать иерархию классов-создателей с абстрактным предком AnimalCreator и конкретными потомками CatCreator и ApeCreator. Фабричный метод CreateAnimal(ind, name) этих классов принимает параметр целого типа ind и строковый параметр name и возвращает объект типа Animal. Для конкретных классов CatCreator и ApeCreator параметр ind метода CreateAnimal определяет тип создаваемого животного по его индексу (0, 1 или 2) в группе кошачьих или обезьян, а параметр name — имя животного.

В классе AnimalCreator также определить метод GetZoo с двумя параметрами-массивами inds и names одинакового размера; массив inds содержит целые числа, массив names — строки (предполагается, что элементы массива inds всегда имеют значения в диапазоне от 0 до 2). Метод GetZoo возвращает массив объектов Animal того же размера, что и массивы inds и names; каждый элемент полученного массива определяется с помощью фабричного метода с параметрами, равными значениям соответствующих элементов массивов inds и names.

Тестирование разработанной системы классов. Дан набор из 4 пар (ind, name), где ind — целое число в диапазоне от 0 до 2, а name — строка. Сформировать массивы inds и names размера 4, содержащие числа и строки из исходного набора, и использовать их в методе GetZoo для создателей CatCreator и ApeCreator, получив в результате наборы кошачьих и обезьян размера 4. С помощью метода GetInfo вывести информацию о животных из каждого набора.

* * *

Таким образом, в задании OOP1Creat3 метод GetZoo является конкретным примером операции AnOperation, использующей фабричный метод (см. диаграмму классов для паттерна Factory Method, приведенную в задании OOP1Creat1).

Первые задания для паттернов State и Interpreter (OOP3Behav10 и OOP3Behav16) также могут быть отнесены к конкретным заданиям, поскольку в первом из них реализуется строковый парсер, распознающий ряд токенов, а во втором — грамматика, связанная с вычислением арифметического выражения.

Случаев, когда вторые задания не связаны с конкретной задачей, немного: это OOP3Behav2 (паттерн Observer) и OOP3Behav14 (паттерн Chain of Responsibility). В этих заданиях используются те же формальные наборы классов, что и в соответствующих первых заданиях, однако рассматриваются важные дополнительные модификации паттерна.

Если с паттерном связано единственное задание, то, как правило, в нем описывается формальная система классов, не связанная с конкретной предметной областью и решающая условную задачу по обработке числовых или строковых данных. Однако среди них тоже можно отметить задания с реальным содержанием: это реализация виртуального и защищающего заместителей в OOP2Struc9 (паттерн Proxy) и генерация различных вариантов заголовков в OOP2Struc10 (паттерн Bridge).

С двумя паттернами связано более двух заданий: это Factory Method — рассмотренный выше паттерн из группы OOP1Creat, для которого имеются три задания, — и Adapter — первый паттерн из группы OOP2Struc, для которого имеются четыре задания. Второе задание для паттерна Factory Method представляет собой модификацию первого задания, в которой демонстрируется вариант реализации паттерна, не использующий абстрактные классы.

Наличие четырех заданий для паттерна Adapter связано с тем, что данный паттерн имеет два варианта: первый вариант называется *адаптером объекта* и основан на включении адаптируемого объекта в класс-адаптер в виде ссылочного поля, второй вариант называется *адаптером класса* и основан на множественном наследовании (или на наследовании с дополнительным подключением интерфейсов).

5. Особенности формулировок заданий

Поскольку задания можно выполнять с использованием различных объектно-ориентированных языков, в их формулировках применяются только базовые понятия ООП:

- объекты;
- классы (в том числе абстрактные);
- методы (в том числе абстрактные и статические);
- поля (в том числе статические).

Практически во всех паттернах (кроме адаптера класса) применяется одиночное наследование; при описании адаптера класса указывается, что наряду с множественным наследованием (основным способом построения подобного адаптера) можно использовать наследование с подключением дополнительных интерфейсов (для языков, подобных C# и Java, в которых множественное наследование не реализовано).

В формулировках заданий не говорится о том, что методы, переопределяемые в подклассах, должны оформляться как виртуальные во всех классах, входящих в иерархию, поскольку это требование является фундаментом ООП (и, кроме того, в ряде языков переопределяемые методы автоматически делаются виртуальными).

Некоторые задания снабжены примечаниями, в которых описываются особенности реализации паттернов для конкретных языков. Например, в заданиях, связанных с паттерном Observer (OOP3Behav1–2), рассматривается классическая схема реализации этого паттерна, однако в дополнительных примечаниях к этим заданиям для языков C# и Java описываются варианты паттерна, основанные на применении специальных средств этих языков.

При разработке заданий требовалось учитывать, что в некоторых языках (например, в C#) все классы являются ссылочными типами, а в других (например, в C++) объекты не обязаны размещаться в динамической памяти. Как правило, при необходимости использования ссылки на объект об этом явно говорится в формулировке задания (хотя для ссылочных объектно-ориентированных языков подобное уточнение является излишним). Поскольку для языка C++ отсутствует механизм сборки мусора, а явное применение операций new/delete усложняет реализацию классов и нередко приводит к большим проблемам, в заготовках программ для языка C++ практически повсеместно используются *интеллектуальные указатели* shared_ptr. Кроме того, в заготовки для C++ включаются деструкторы классов, выводящие на экран отладочную информацию, с помощью которой можно проверить, все ли используемые в программе объекты были корректно разрушены.

В качестве примера приведем образцы программ-заготовок для заданий OOP1Creat1 и OOP1Creat3, формулировки которых содержатся в предыдущем разделе. Для задания OOP1Creat1 приводится заготовка на языке C++ (листинг 1), для задания OOP1Creat3 — на языке Python (листинг 2).

В функции Solve следует выполнять действия по вводу исходных данных, созданию и использованию объектов разработанных классов и выводу результатов (см. пример реализации этой функции в следующем разделе).

6. Использование задачника Programming Taskbook for OOP

Чтобы дать представление об особенностях выполнения заданий с применением задачника Programming Taskbook for OOP, приведем краткое описание процесса выполнения одного из рассмотренных выше заданий (OOP1Creat1), иллюстрируемое несколькими скриншотами. Заметим, что подробное описание решения OOP1Creat1 для пяти языков программирования (C++, Python, C#, Ruby и Java) содержится на сайте электронного задачника Programming Taskbook в разделе, посвященном задачнику Programming Taskbook for OOP*.

Для создания программы-заготовки на выбранном языке программирования и ее загрузки в требуемую интегрированную среду предназначена специальная программа PT4Load, входящая в состав задачника Programming Taskbook. Созданная программа-заготовка может быть сразу запущена на выполнение. На рисунке 1 приведен вид начальной части окна задачника с формулировкой задания OOP1Creat1 при запуске заготовки программы на

* http://ptaskbook.com/ru/ptforoop/

ISSN 0234-0453 • ИНФОРМАТИКА И ОБРАЗОВАНИЕ • 2023 • Том 38 № 2

```
#include <vector>
#include <memory>
#include "pt4.h"
using namespace std;

class Product
{
public:
    virtual string GetInfo() = 0;
    virtual void Transform() = 0;
    virtual ~Product()
    {
        Show("Product");
    }
};

// Implement the ConcreteProduct1
//   and ConcreteProduct2 descendant classes

class Creator
{
protected:
    virtual shared_ptr<Product>
      FactoryMethod(string info) = 0;
public:
    string AnOperation(string info);
    virtual ~Creator()
    {
        Show("Creator");
    }
};

string Creator::AnOperation(string info)
{
    auto p = FactoryMethod(info);
    p->Transform();
    p->Transform();
    return p->GetInfo();
}

// Implement the ConcreteCreator1
//   and ConcreteCreator2 descendant classes;
//   the AnOperation method should not be
//   overridden in these classes

void Solve()
{
    Task("OOP1Creat1");
}
```

Листинг 1. Заготовка для задания OOP1Creat1 на языке C++

Listing 1. The C++ template for the OOP1Creat1 task

```
from pt4 import *
class Lion:
    def __init__(self, name):
        self.__name = name
    def getInfo(self):
        return "Lion " + self.__name

# Implement the Tiger, Leopard, Gorilla,
#   Orangutan and Chimpanzee classes

class AnimalCreator:
    def getZoo(self, inds, names):
        zoo = []
        for i in range(len(inds)):
            zoo.append(self.createAnimal(inds[i],
              names[i]))
        return zoo

class CatCreator(AnimalCreator):
    def createAnimal(self, ind, name):
        pass
        # Implement the method

# Implement the ApeCreator descendant class

def solve():
    task("OOP1Creat3")
```

Листинг 2. Заготовка для задания OOP1Creat3 на языке Python

Listing 2. The Python template for the OOP1Creat3 task

языке C++ из редактора Visual Studio Code. В правой верхней части экрана отображается дополнительное окно с диаграммой классов, связанной с изучаемым паттерном Factory Method.

Формулировку задания можно отобразить в более наглядном виде в веб-браузере (рис. 2); для этого достаточно щелкнуть мышью на метке «Режим (F4)» в окне задачника.

На рисунке 3 приведено окно задачника, в котором скрыта формулировка задания и отображаются только исходные данные и пример их правильной обработки (в качестве исходных и результирующих данных в заданиях могут использоваться целые и вещественные числа, символы, строки и данные логического типа). В окне также отображается завершающий раздел с текстом программы-заготовки для данного задания. В этом разделе могут содержаться дополнительные указания по выполнению задания на выбранном языке.

Для успешного выполнения задания необходимо разработать описанную в нем систему классов и протестировать ее, обработав несколько наборов исходных данных, предоставляемых учебной программе самим задачником. Не приводя полный текст решения, который должен также включать в себя реализацию конкретных классов-продуктов ConcreteProduct1 и ConcreteProduct2, укажем в листинге 3 завершающий фрагмент, содержащий описание конкретных классов-создателей ConcreteCreator1 и ConcreteCreator2 и функцию Solve, в которой выполняется ввод исходных данных, создаются и используются экземпляры требуемых классов и выводятся полученные результаты. Следует обратить внимание на то, что для ввода и вывода данных предназначены специальные средства, реализованные в задачнике (в частности, для языка C++ можно использовать особый поток ввода-вывода pt).

Данный фрагмент демонстрирует применение паттерна Factory Method, благодаря которому удается обеспечить *различные действия* при выполнении метода AnOperation для различных классов-потомков *без переопределения этого метода в классах-потомках*.

На рисунке 4 приведен вид окна задачника при запуске программы с правильным решением задачи. После успешной проверки решения на пяти тестовых

ISSN 0234-0453 • INFORMATICS AND EDUCATION • 2023 • Volume 38 No 2

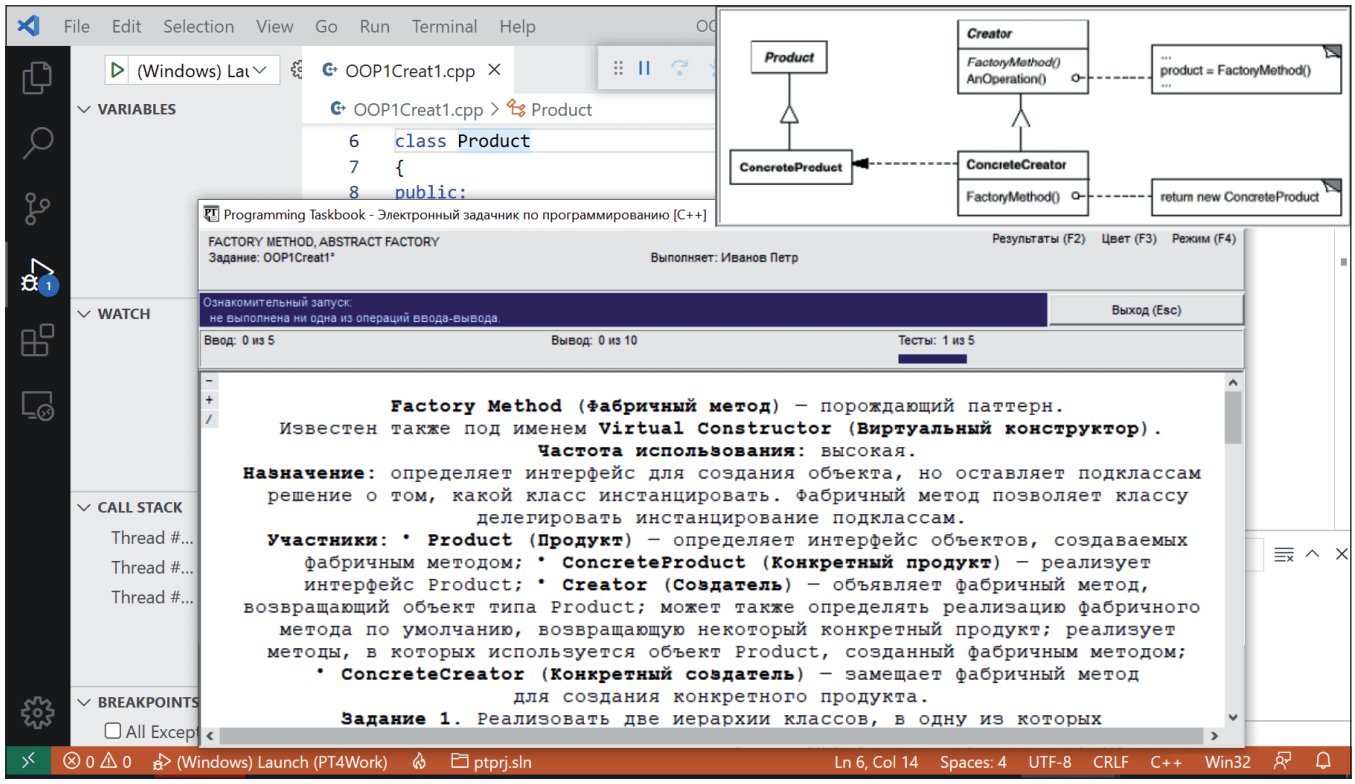


Рис. 1. Запуск программы-заготовки из редактора Visual Studio Code
Fig. 1. Launching a template program from the Visual Studio Code editor

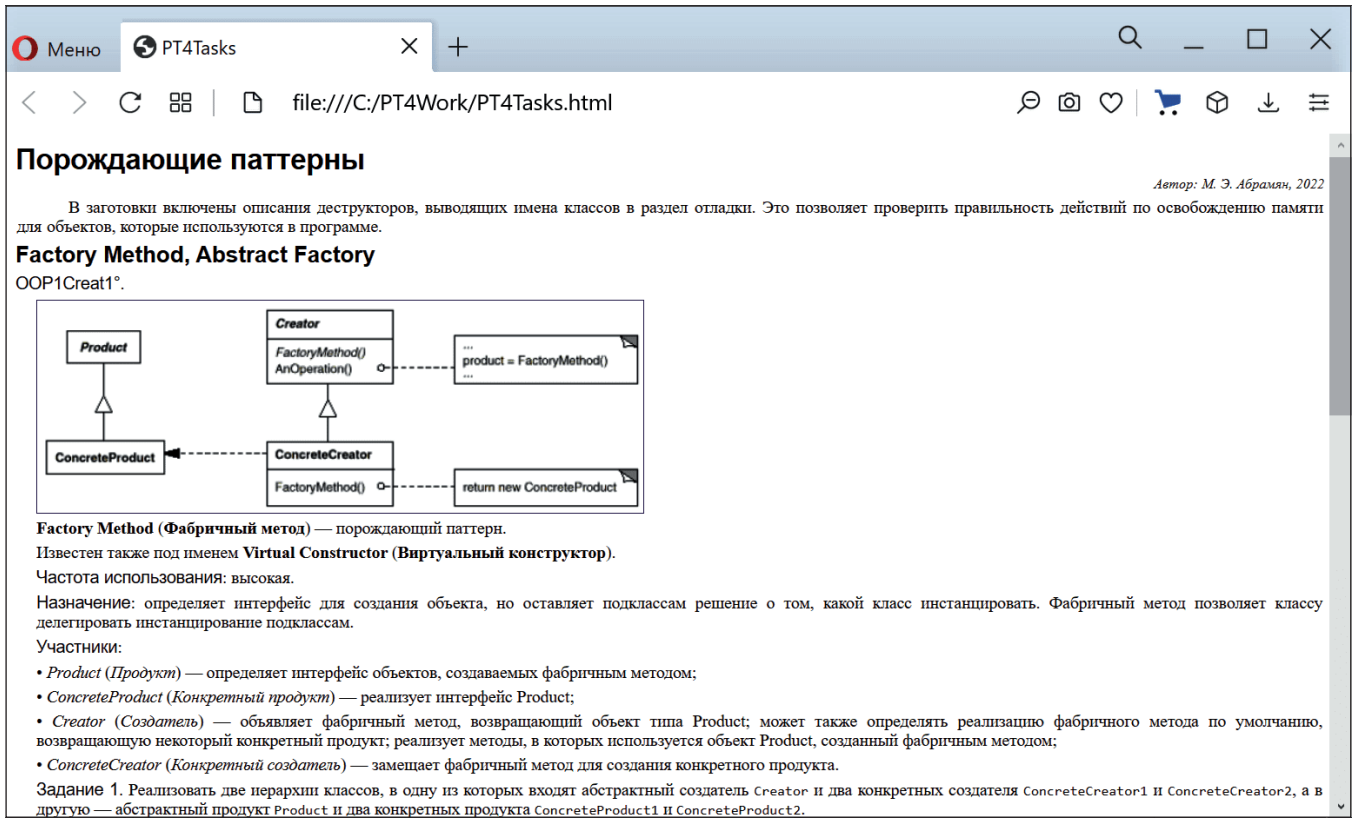


Рис. 2. Отображение формулировки задания в окне веб-браузера
Fig. 2. Displaying the task wording in the web-browser window

ISSN 0234-0453 • ИНФОРМАТИКА И ОБРАЗОВАНИЕ • 2023 • Том 38 № 2

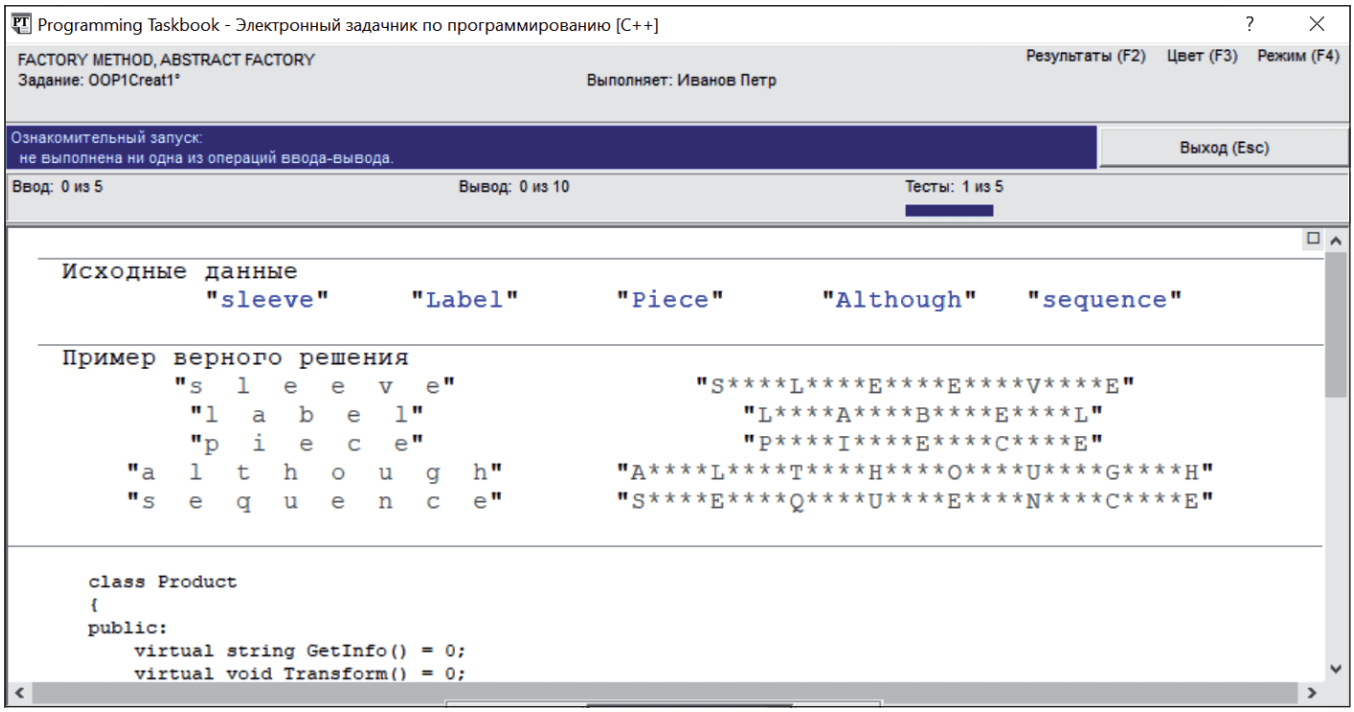


Рис. 3. Разделы окна задачника с исходными данными и примером правильного решения
Fig. 3. Sections of the taskbook window with the input data and an example of the correct solution

```
class ConcreteCreator1 : public Creator
{
protected:
    shared_ptr<Product> FactoryMethod(string info) override;
};

class ConcreteCreator2 : public Creator
{
protected:
    shared_ptr<Product> FactoryMethod(string info) override;
};

shared_ptr<Product> ConcreteCreator1::FactoryMethod(string info)
{
    return make_shared<ConcreteProduct1>(info);
}

shared_ptr<Product> ConcreteCreator2::FactoryMethod(string info)
{
    return make_shared<ConcreteProduct2>(info);
}

void Solve()
{
    Task("OOP1Creat1");
    ConcreteCreator1 c1;
    ConcreteCreator2 c2;
    for (int i = 0; i < 5; i++)
    {
        string s;
        pt >> s;
        pt << c1.AnOperation(s) << c2.AnOperation(s);
    }
}
```

Листинг 3. Завершающая часть программы с решением задачи OOP1Creat1 на C++
Listing 3. Final part of the C++ program with the solution of the OOP1Creat1 task

ISSN 0234-0453 • INFORMATICS AND EDUCATION • 2023 • Volume 38 No 2

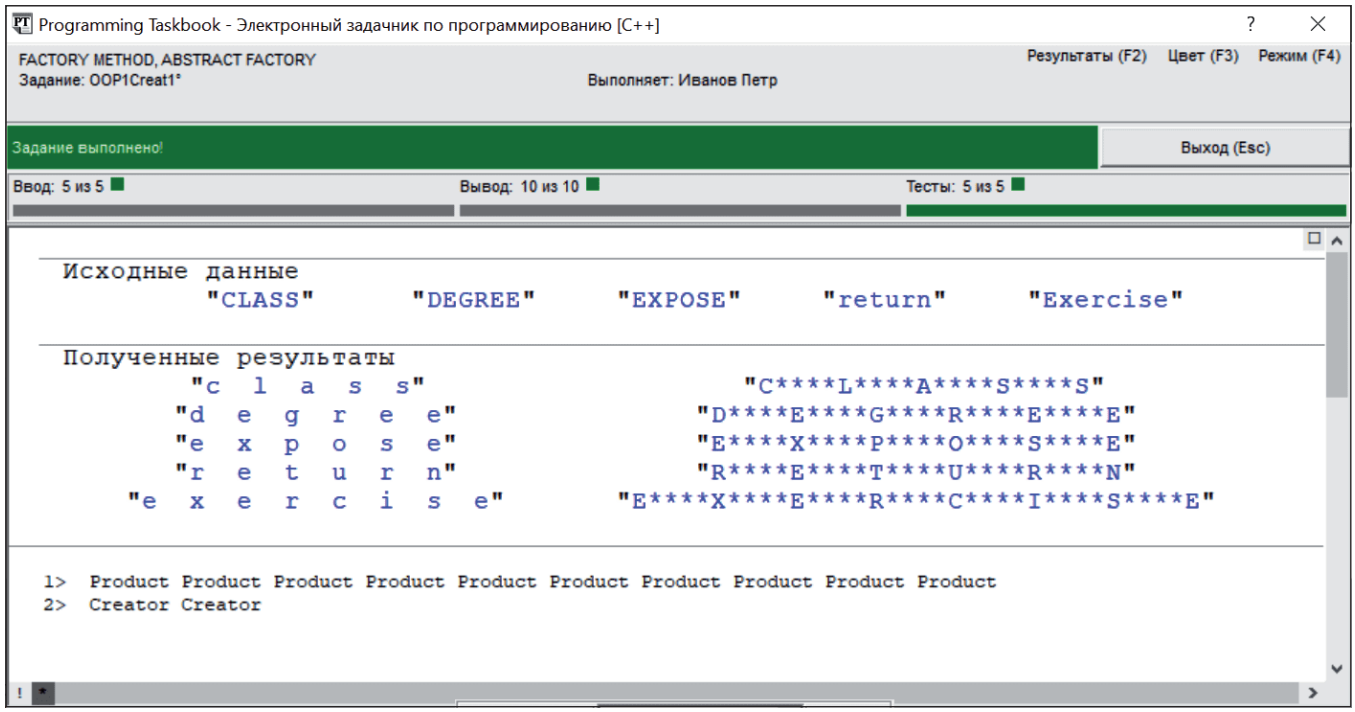


Рис. 4. Окно задачника с сообщением о правильном решении задачи OOP1Creat1
Fig. 4. The taskbook window with the message about the correct solution of the OOP1Creat1 task

наборах данных выводится сообщение о том, что задание выполнено. В случае использования языка C++ в завершающем разделе окна задачника выводится дополнительная информация, показывающая, что благодаря применению интеллектуальных указателей shared_ptr были вызваны деструкторы всех созданных в программе объектов.

7. Заключение

Применение задачника Programming Taskbook for OOP позволяет успешно решить проблему, сформулированную в разделе 1, а именно обеспечить методическую поддержку практической части курсов по объектно-ориентированному программированию, связанной с изучением паттернов проектирования.

Задачник использовался при проведении лабораторных занятий по дисциплинам «Объектно-ориентированное программирование» (3-й курс бакалавриата факультета вычислительной математики и кибернетики Университета МГУ-ППИ в Шэньчжэне) и «Языки программирования» (1-й год магистратуры Института математики, механики и компьютерных наук Южного федерального университета, программа «Разработка компьютерных игр и мобильных приложений»), что позволило существенно увеличить количество выполненных заданий и упростило проверку их правильности.

Помимо этого, задачник применялся на лекциях при изучении конкретных паттернов, поскольку обсуждение формулировок задач, программ-заготовок и тестовых данных позволило осветить различные аспекты рассматриваемого паттерна и продемонстрировать его работу на примерах. В качестве примеров на лекциях использовались также задания из вводной группы, что дало возможность наглядно проиллюстрировать основные принципы и приемы ООП.

Список источников / References

1. *Гамма Э., Хелм Р., Джонсон Р., Влиссидес Дж.* Приемы объектно-ориентированного проектирования. Паттерны проектирования. СПб.: Питер; 2019. 368 с.

[*Gamma E., Helm R., Johnson R., Vlissides J.* Design patterns. Elements of reusable object-oriented software. Saint Petersburg, Piter; 2019. 368 p. (In Russian)]

2. *Фримен Э., Робсон Э., Сьерра К., Бейтс Б.* Head First. Паттерны проектирования. СПб.: Питер; 2018. 657 с.

[*Freeman E., Robson E., Sierra K., Bates B.* Head First Design Patterns. Saint Petersburg, Piter; 2018. 657 p. (In Russian)]

3. *Шевчук А., Охрименко Д., Касьянов А.* Design Patterns via C#. Приемы объектно-ориентированного проектирования. Электронное издание; 2015. 288 с.

[*Shevchuk A., Okhrimenko D., Kasyanov A.* Design Patterns via C#. Object-oriented design techniques. Electronic edition; 2015. 288 p. (In Russian.)]

4. *Kasampalis S.* Mastering Python Design Patterns. Packt, Birmingham-Mumbai; 2015. 214 p.

5. *Olsen R.* Design Patterns in Ruby. Boston, Addison-Wesley; 2008. 384 p.

6. *Вайсфельд М.* Объектно-ориентированное мышление. СПб.: Питер; 2014, 304 с. Режим доступа: https://elibrary.ceiti.md/files/12/Vaysfeld_Obektno-orientirovannoe_myishlenie.pdf

[*Weisfeld M.* The object-oriented thought process. Saint Petersburg, Piter; 2014, 304 p. (In Russian.) Available at: https://elibrary.ceiti.md/files/12/Vaysfeld_Obektno-orientirovannoe_myishlenie.pdf]

7. *Astrachan O., Berry G., Cox L., Mitchener G.* Design patterns: An essential component of CS curricula. *Proc. 29th SIGCSE Technical Symposium on Computer Science Education*. 1998;30(1):153–160. DOI: 10.1145/273133.273182

8. *Stuurman S., Florijn G.* Experiences with teaching design patterns. *ACM SIGCSE Bulletin*. 2004;36(3):151–155. DOI: 10.1145/1026487.1008037

9. *Gestwicki P., Sun F.-S.* Teaching design patterns through computer game development. *Journal on Educational Resources in Computing*. 2008;8(1):2. DOI: 10.1145/1348713.1348715

10. *Muyan-Özçelik P.* A hands-on cross-platform mobile programming approach to teaching OOP concepts and design patterns. *Proc. of the 1st International Workshop on Software Engineering Curricula for Millennials*. 2017:33–39. DOI: 10.1109/SECM.2017.12

11. *Azimullah Z., Young Sun An, Denny P.* Evaluating an interactive tool for teaching design patterns. *Proc. of the Twenty-Second Australasian Computing Education Conf. (ACE'20)*. 2020:167–176. DOI: 10.1145/3373165.3373184

12. *Бабушкина И. А., Окулов С. М.* Практикум по объектно-ориентированному программированию. Учебное пособие. М.: БИНОМ. Лаборатория знаний; 2012. 372 с. EDN: RBATTD

[*Babushkina I. A., Okulov S. M.* Workshop on object-oriented programming. Tutorial. Moscow, BINOM. Laboratoriya znanij; 2012. 372 p. (In Russian.) EDN: RBATTD]

13. *Варфоломеева Т. Н., Ефимова И. Ю.* Лабораторный практикум по объектно-ориентированному программированию. М.: ФЛИНТА; 2019. 74 с.

[*Varfolomeeva T. N., Efimova I. Yu.* Laboratory workshop on object-oriented programming. Moscow, FLINTA; 2019. 74 p. (In Russian.)]

14. *Ашарина И. В., Крупская Ж. Ф.* Язык C++ и объектно-ориентированное программирование в C++. Лабораторный практикум. М.: Горячая линия — Телеком; 2016. 232 с.

[*Asharina I. V., Krupskaya Zh. F.* C++ language and object-oriented programming in C++. Laboratory workshop. Moscow, Hot Line — Telecom; 2016. 232 p. (In Russian.)]

15. *Наумов В. Ю., Гостевская О. В.* Объектно-ориентированное программирование на C++. Лабораторный практикум. Волгоград: ВолгГТУ; 2015. 64 с. EDN: RVFTXN

[*Naumov V. Yu., Gostevskaya O. V.* Object-oriented programming in C++. Laboratory workshop. Volgograd, Volgograd State Technical University; 2015. 64 p. (In Russian.) EDN: RVFTXN]

16. Объектно-ориентированное программирование: практикум. Сост. А. А. Халидов. Казань: Казанский государственный энергетический университет; 2018. 83 с. Режим доступа: https://lib.kgeu.ru/irbis64r_15/scan/186эл.pdf

[Object-oriented programming: A workshop. Comp. A. A. Khalidov. Kazan, Kazan State Power Engineering University; 2018. 83 c. (In Russian.) Available at: https://lib.kgeu.ru/irbis64r_15/scan/186эл.pdf]

17. Объектно-ориентированное программирование: лабораторный практикум. Сост. А. В. Щербаков. Минск: БНТУ; 2014. 38 с. Режим доступа: https://rep.bntu.by/handle/data/7771?show=full

[Object-oriented programming: Laboratory workshop. Comp. A. V. Scherbakov. Minsk, BNTU; 2014. 38 p. (In Russian.) Available at: https://rep.bntu.by/handle/data/7771?show=full]

18. *Мишуто К. А.* Электронный лабораторный практикум по дисциплине «Объектно-ориентированное программирование». *Точная наука*. 2018;(26):116–117. EDN: XRVQAX

[*Mishuto K. A.* Electronic laboratory practicum on discipline "Object-oriented programming". *Tochnaya Nauka*. 2018;(26):116–117. (In Russian.) EDN: XRVQAX]

19. *Абрамян М. Э.* Инструменты и методы разработки электронных образовательных ресурсов по компьютерным наукам: монография. Ростов-на-Дону; Таганрог: Южный федеральный университет; 2018. 260 с. EDN: YOWZAL

[*Abramyan M. E.* Tools and methods for developing educational software in computer science: monograph. Rostov-on-Don, Taganrog, Southern Federal University; 2018. 260 p. (In Russian.) EDN: YOWZAL]

20. *Абрамян М. Э.* Об архитектуре универсального электронного задачника по программированию. *Информатизация образования и науки*. 2015;(3(27)):134–149. EDN: RWVJUI

[*Abramyan M. E.* On the architecture of the universal problem book on programming. *Informatization of Education and Science*. 2015;(3(27)):134–149. (In Russian.) EDN: RWVJUI]

21. *Johnson B.* Visual Studio Code: End-to-End editing and debugging tools for web developers. Indianapolis, Wiley; 2019. 192 p.

22. *Абрамян М. Э.* Разработка компонентов электронных задачников по программированию для автоматизации проведения лабораторных занятий. *Информатизация образования и науки*. 2020;(4(48)):12–28. EDN: SVVAVB

[*Abramyan M. E.* Development of components of electronic programming problems books for automation of laboratory classes. *Informatization of Education and Science*. 2020;(4(48)):12–28. (In Russian.) EDN: SVVAVB]

Информация об авторе

Абрамян Михаил Эдуардович, канд. физ.-мат. наук, доцент, доцент факультета вычислительной математики и кибернетики, Университет МГУ-ППИ в Шэньчжэне, Китай; доцент кафедры алгебры и дискретной математики, Институт математики, механики и компьютерных наук им. И. И. Воровича, Южный федеральный университет, г. Ростов-на-Дону, Россия; *ORCID:* https://orcid.org/0000-0002-2802-6144; *e-mail:* mabr@sfedu.ru

Information about the author

Mikhail E. Abramyan, Candidate of Sciences (Physics and Mathematics), Docent, Associate Professor at the Faculty of Computational Mathematics and Cybernetics, Shenzhen MSU-BIT University, China; Associate Professor at the Department of Algebra and Discrete Mathematics, Institute for Mathematics, Mechanics, and Computer Science named after of I. I. Vorovich, Southern Federal University, Rostov-on-Don, Russia; *ORCID:* https://orcid.org/0000-0002-2802-6144; *e-mail:* mabr@sfedu.ru

Поступила в редакцию / Received: 06.11.22.
Поступила после рецензирования / Revised: 20.01.23.
Принята к печати / Accepted: 24.01.23.