КАЗАНСКИЙ (ПРИВОЛЖСКИЙ) ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ

ИНСТИТУТ ВЫЧИСЛИТЕЛЬНОЙ МАТЕМАТИКИ И ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ (ИВМИИТ-ВМК)

Арабов М.К., Сабитов Ш. Р., Хайдаров М.

**Практикум по «ООП и паттерны программирования»**

**учебное пособие для бакалавров и магистрантов ВУЗов,   
специализирующихся в области информационных**

**технологий и программирования**

**Казань - 2024**

Арабов М.К., Сабитов Ш.Р., Хайдаров М.

Практикум по «ООП и паттерны программирования»: учебное пособие для бакалавров и магистрантов ВУЗов, специализирующихся в области информационных технологий и программирования /М.К. Арабов, Ш. Р. Сабитов, М. Хайдаров. - \_\_\_\_\_\_\_\_\_\_\_\_, 2024. \_310\_ с.

*В учебном пособии рассмотрены основные аспекты объектно-ориентированного программирования (ООП) и паттерны проектирования, с акцентом на решение практических задач. Пособие предназначено для бакалавров и магистрантов ВУЗов, специализирующихся в области информационных технологий и программирования. В отличие от традиционных теоретических учебников, это пособие ориентировано на решение задач с реальными примерами кода, которые позволяют учащимся не только понять концепции ООП, но и освоить их на практике.*

**Редактор:**

**Рецензенты**

© Арабов М.К., Сабитов Ш.Р., Хайдаров М. – 2024.

**Оглавление**

[Предисловие 6](#_Toc182147912)

[ГЛАВА 1. ОСНОВЫ ООП 7](#_Toc182147913)

[1. Классы и объекты 7](#_Toc182147914)

[Задачи для самостоятельной работы 11](#_Toc182147915)

[2. Наследование 15](#_Toc182147916)

[Задачи для самостоятельной работы 20](#_Toc182147917)

[3. Инкапсуляция 24](#_Toc182147918)

[Задачи для самостоятельной работы 29](#_Toc182147919)

[4. Композиция и агрегация 33](#_Toc182147920)

[Пример решений задач 33](#_Toc182147921)

[Композиция 33](#_Toc182147922)

[Задачи для самостоятельной работы 39](#_Toc182147923)

[Агрегация 42](#_Toc182147924)

[Задачи для самостоятельной работы 49](#_Toc182147925)

[5. Абстрактные классы 53](#_Toc182147926)

[Задачи для самостоятельной работы 57](#_Toc182147927)

[6. Интерфейсы 61](#_Toc182147928)

[Задачи для самостоятельной работы 65](#_Toc182147929)

[7. Полиморфизм 69](#_Toc182147930)

[Задачи для самостоятельной работы 79](#_Toc182147931)

[ГЛАВА 2. ПРИНЦИПЫ ООП 83](#_Toc182147932)

[1. Dependency Injection 83](#_Toc182147933)

[Задачи для самостоятельной работы 91](#_Toc182147934)

[2. Маппинг данных 95](#_Toc182147935)

[Задачи для самостоятельной работы 98](#_Toc182147936)

[3. Принципы SOLID 101](#_Toc182147937)

[Задачи для самостоятельной работы 110](#_Toc182147938)

[ГЛАВА 3. ПАТТЕРНЫ СОЗДАНИЯ 116](#_Toc182147939)

[1. Паттерн Singleton 116](#_Toc182147940)

[Задачи для самостоятельной работы 121](#_Toc182147941)

[2. Factory Method 125](#_Toc182147942)

[Задачи для самостоятельной работы 130](#_Toc182147943)

[3. Abstract Factory 134](#_Toc182147944)

[Задачи для самостоятельной работы 142](#_Toc182147945)

[4. Паттерн Builder 146](#_Toc182147946)

[Задачи для самостоятельной работы 151](#_Toc182147947)

[5. Паттерн Prototype 154](#_Toc182147948)

[Задачи для самостоятельной работы 159](#_Toc182147949)

[ГЛАВА 4. СТРУКТУРНЫЕ ПАТТЕРНЫ 163](#_Toc182147950)

[1. Паттерн Adapter 163](#_Toc182147951)

[Задачи для самостоятельной работы 168](#_Toc182147952)

[2. Паттерн Composite 172](#_Toc182147953)

[Задачи для самостоятельной работы 177](#_Toc182147954)

[3. Паттерн Proxy 180](#_Toc182147955)

[Задачи для самостоятельной работы 186](#_Toc182147956)

[4. Паттерн Bridge 190](#_Toc182147957)

[Задачи для самостоятельной работы 197](#_Toc182147958)

[5. Паттерн Decorator 202](#_Toc182147959)

[Задачи для самостоятельной работы 208](#_Toc182147960)

[6. Паттерн Facade 211](#_Toc182147961)

[Задачи для самостоятельной работы 219](#_Toc182147962)

[7. Паттерн Flyweight 223](#_Toc182147963)

[Задачи для самостоятельной работы 229](#_Toc182147964)

[ГЛАВА 5. ПОВЕДЕНЧЕСКИЕ ПАТТЕРНЫ 233](#_Toc182147965)

[1. Паттерн Observer 233](#_Toc182147966)

[Задачи для самостоятельной работы 240](#_Toc182147967)

[2. Паттерн Strategy 244](#_Toc182147968)

[Задачи для самостоятельной работы 248](#_Toc182147969)

[3. Паттерн Command 253](#_Toc182147970)

[Задачи для самостоятельной работы 257](#_Toc182147971)

[4. Паттерн Iterator 260](#_Toc182147972)

[Задачи для самостоятельной работы 266](#_Toc182147973)

[5. Паттерн State 270](#_Toc182147974)

[Задачи для самостоятельной работы 283](#_Toc182147975)

[6. Паттерн Mediator 287](#_Toc182147976)

[Задачи для самостоятельной работы 293](#_Toc182147977)

[7. Паттерн Chain of Responsibility 297](#_Toc182147978)

[Задачи для самостоятельной работы 303](#_Toc182147979)

[Литература 307](#_Toc182147980)

## Предисловие

В современном мире программирования концепции объектно-ориентированного программирования (ООП) и паттернов проектирования играют ключевую роль в создании эффективных, масштабируемых и поддерживаемых программных решений. Эти подходы значительно упрощают разработку программного обеспечения, а также способствуют созданию более гибких и адаптивных систем, что особенно важно в условиях быстро меняющихся требований и технологий. ООП позволяет создавать приложения, которые легко изменять и расширять, а паттерны проектирования помогают решать повторяющиеся задачи с использованием проверенных решений, что существенно улучшает качество кода и уменьшает время разработки.

Данное учебное пособие содержит множество задач и примеров, которые охватывают как основные концепции объектно-ориентированного программирования, так и важнейшие принципы и паттерны проектирования. В частности, рассмотрены принципы SOLID, которые служат основой для написания чистого и поддерживаемого кода, а также такие паттерны, как Singleton, Factory Method, Abstract Factory, Builder, Prototype, Adapter, Composite, Proxy, Bridge, Decorator, Facade, Flyweight, Observer, Strategy, Command, Iterator, State, Mediator, Chain of Responsibility и многие другие.

Эта книга предназначена для студентов, аспирантов и разработчиков, желающих углубить свои знания в области ООП и освоить основные паттерны проектирования, которые помогут эффективно решать реальные задачи программирования. Цель пособия — не только предоставить теоретическую базу, но и развить практические навыки, которые можно применять в повседневной разработке программного обеспечения.

Каждая глава включает в себя задачи для самостоятельной работы, которые помогут закрепить изучаемые концепции, а также примеры решений на различных языках программирования, таких как C++, C#, Java и Python. Это позволит читателю увидеть, как одни и те же концепции реализуются в разных средах и на различных платформах, а также развить гибкость мышления, необходимую для адаптации к новым языкам и технологиям. Такой подход способствует более глубокому пониманию как основ ООП, так и паттернов проектирования, предоставляя инструменты для создания высококачественного, расширяемого и поддерживаемого кода.

Пособие будет полезно не только для студентов и начинающих разработчиков, но и для опытных программистов, стремящихся улучшить свои навыки и повысить уровень профессионализма в решении сложных задач программирования.

Авторы надеются, что это пособие станет полезным инструментом для всех, кто хочет стать более компетентным разработчиком и успешно применять ООП и паттерны проектирования в своей практике. Мы будем благодарны всем, кто пришлет замечания и пожелания по электронному адресу cool.araby@mail.ru.

# ГЛАВА 1. ОСНОВЫ ООП

## 1. Классы и объекты

Пример решений задач

Задача 1: Определите класс Person, представляющий человека. Реализуйте конструктор, который принимает имя, возраст и пол. Добавьте метод display\_info(), который выводит информацию о человеке, и метод set\_age(), позволяющий изменять возраст.

|  |
| --- |
| С++ |
| #include <iostream>  #include <string>  class Person {  private:  std::string name; // Имя  int age; // Возраст  std::string gender; // Пол  public:  // Конструктор  Person(std::string name, int age, std::string gender) {  this->name = name;  this->age = age;  this->gender = gender;  }  // Метод для отображения информации  void displayInfo() {  std::cout << "Имя: " << name << ", Возраст: " << age << ", Пол: " << gender << std::endl;  }  // Метод для изменения возраста  void setAge(int newAge) {  if (newAge >= 0) {  age = newAge;  } else {  std::cout << "Возраст не может быть отрицательным." << std::endl;  }  }  };  // Пример использования класса  int main() {  Person person("Иван", 30, "Мужской");  person.displayInfo();  person.setAge(35);  person.displayInfo();  person.setAge(-5);  return 0;  } |
| C# |
| using System;  public class Person  {  // Поля класса для хранения имени, возраста и пола  private string name;  private int age;  private string gender;  // Конструктор класса, принимающий имя, возраст и пол  public Person(string name, int age, string gender)  {  this.name = name; // Инициализация имени  this.age = age; // Инициализация возраста  this.gender = gender; // Инициализация пола  }  // Метод для отображения информации о человеке  public void DisplayInfo()  {  Console.WriteLine($"Имя: {name}, Возраст: {age}, Пол: {gender}");  }  // Метод для изменения возраста  public void SetAge(int newAge)  {  if (newAge >= 0) // Проверка, что возраст не отрицательный  {  age = newAge; // Установка нового возраста  }  else  {  Console.WriteLine("Возраст не может быть отрицательным."); // Сообщение об ошибке  }  }  }  // Пример использования класса  class Program  {  static void Main()  {  // Создание объекта Person  Person person = new Person("Иван", 30, "Мужской");    // Отображение информации о человеке  person.DisplayInfo();    // Изменение возраста  person.SetAge(35);    // Отображение обновленной информации  person.DisplayInfo();    // Попытка установить отрицательный возраст  person.SetAge(-5);  }  } |
| Java |
| public class Person {  private String name; // Имя  private int age; // Возраст  private String gender; // Пол  // Конструктор  public Person(String name, int age, String gender) {  this.name = name;  this.age = age;  this.gender = gender;  }  // Метод для отображения информации  public void displayInfo() {  System.out.println("Имя: " + name + ", Возраст: " + age + ", Пол: " + gender);  }  // Метод для изменения возраста  public void setAge(int newAge) {  if (newAge >= 0) {  age = newAge;  } else {  System.out.println("Возраст не может быть отрицательным.");  }  }  // Пример использования класса  public static void main(String[] args) {  Person person = new Person("Иван", 30, "Мужской");  person.displayInfo();  person.setAge(35);  person.displayInfo();  person.setAge(-5);  }  } |
| Python |
| class Person:  def \_\_init\_\_(self, name, age, gender):  self.name = name # Имя  self.age = age # Возраст  self.gender = gender # Пол  def display\_info(self):  print(f"Имя: {self.name}, Возраст: {self.age}, Пол: {self.gender}")  def set\_age(self, new\_age):  if new\_age >= 0:  self.age = new\_age  else:  print("Возраст не может быть отрицательным.")  # Пример использования класса  if \_\_name\_\_ == "\_\_main\_\_":  person = Person("Иван", 30, "Мужской")  person.display\_info()  person.set\_age(35)  person.display\_info()  person.set\_age(-5) |

**Обратите внимание:** В ряде языков программирования, таких как Java и C# и т.п., метод displayInfo можно заменить на переопределение метода toString. Это можно сделать следующим образом:

Пример на Java:

### Задачи для самостоятельной работы

Задача 2: Создайте класс Document, который содержит название, автора и содержимое. Реализуйте методы update\_content(new\_content), который обновляет содержимое документа, и display\_info(), который выводит полную информацию о документе.

Задача 3: Определите класс Product с полями название, цена и количество на складе. Реализуйте метод total\_value(), который вычисляет общую стоимость товара на складе, и метод update\_price(new\_price), который обновляет цену товара.

Задача 4: Создайте класс CartItem с полями название продукта, количество и цена. Реализуйте метод update\_quantity(new\_quantity), который обновляет количество товара, и метод total\_cost(), который вычисляет общую стоимость товаров в корзине.

Задача 5: Определите класс Stack, который реализует стек. Реализуйте методы push(item), добавляющий элемент в стек; pop(), удаляющий верхний элемент; peek(), возвращающий верхний элемент без удаления; и is\_empty(), проверяющий, пуст ли стек.

Задача 6: Создайте класс Queue, реализующий очередь. Реализуйте методы enqueue(item), добавляющий элемент в очередь; dequeue(), удаляющий первый элемент; peek(), возвращающий первый элемент; и is\_empty(), проверяющий, пуста ли очередь.

Задача 7: Определите абстрактный класс Shape с абстрактным методом get\_area(). Создайте подклассы Circle и Rectangle, которые реализуют метод get\_area(), вычисляющий площадь фигуры.

Задача 8: Создайте базовый класс Vehicle с полями марка и модель. Реализуйте подклассы Car и Bicycle, которые добавляют свои специфические методы, например, метод honk() для автомобиля.

Задача 9: Определите класс Event с полями название, дата и место проведения. Реализуйте методы update\_location(new\_location), который обновляет место проведения, и display\_info(), который выводит информацию о событии.

Задача 10: Создайте класс Course с полями название курса, код курса и количество кредитов. Реализуйте методы update\_credits(new\_credits), который обновляет количество кредитов, и display\_info(), который выводит информацию о курсе.

Задача 11: Определите класс LeaveRequest с полями имя сотрудника, начальная дата и конечная дата. Реализуйте методы update\_dates(new\_start\_date, new\_end\_date), который обновляет даты отпуска, и display\_info(), который выводит информацию о заявке.

Задача 12: Создайте класс Profile с полями имя пользователя, электронная почта и дата рождения. Реализуйте методы update\_email(new\_email), который обновляет электронную почту, и display\_info(), который выводит информацию о профиле.

Задача 13: Определите класс Contact с полями имя, номер телефона и электронная почта. Реализуйте методы update\_phone(new\_phone), который обновляет номер телефона, и display\_info(), который выводит информацию о контакте.

Задача 14: Создайте класс Task с полями заголовок задачи, описание и срок выполнения. Реализуйте методы update\_description(new\_description), который обновляет описание задачи, и display\_info(), который выводит информацию о задаче.

Задача 15: Определите класс Computer с полями марка, модель и год выпуска. Реализуйте методы update\_model(new\_model), который обновляет модель, и display\_info(), который выводит информацию о компьютере.

Задача 16: Создайте класс Book с полями название, автор и ISBN. Реализуйте методы update\_author(new\_author), который обновляет автора, и display\_info(), который выводит информацию о книге.

Задача 17: Определите класс CurrencyRate с полями название валюты, код валюты и обменный курс. Реализуйте методы update\_rate(new\_rate), который обновляет обменный курс, и display\_info(), который выводит информацию о валюте.

Задача 18: Создайте класс Client с полями имя, идентификационный номер и баланс. Реализуйте методы update\_balance(amount), который обновляет баланс, и display\_info(), который выводит информацию о клиенте.

Задача 19: Определите класс Account с полями номер счёта и баланс. Реализуйте методы deposit(amount), который пополняет счёт, withdraw(amount), который списывает средства, и display\_info(), который выводит информацию о счёте.

Задача 20: Создайте класс Company с полями название и адрес. Реализуйте методы update\_address(new\_address), который обновляет адрес, и display\_info(), который выводит информацию о фирме.

Задача 21: Определите класс Employee с полями имя, должность и зарплата. Реализуйте методы update\_position(new\_position), который обновляет должность, и display\_info(), который выводит информацию о работнике.

Задача 22: Создайте класс Movie с полями название, режиссёр и год выпуска. Реализуйте методы update\_director(new\_director), который обновляет режиссёра, и display\_info(), который выводит информацию о фильме.

Задача 23: Определите класс University с полями название и количество студентов. Реализуйте методы add\_student(), который добавляет студента, и display\_info(), который выводит информацию о университете.

Задача 24: Создайте класс Payment с полями сумма и дата. Реализуйте методы update\_amount(new\_amount), который обновляет сумму, и display\_info(), который выводит информацию о платеже.

Задача 25: Определите класс Invoice с полями номер счёта и сумма. Реализуйте методы update\_amount(new\_amount), который обновляет сумму, и display\_info(), который выводит информацию о счёте.

Задача 26: Создайте класс Recipe с полями название, ингредиенты и время приготовления. Реализуйте методы update\_cooking\_time(new\_time), который обновляет время приготовления, и display\_info(), который выводит информацию о рецепте.

Задача 27: Определите класс Gadget с полями название, тип и цена. Реализуйте методы update\_price(new\_price), который обновляет цену, и display\_info(), который выводит информацию о гаджете.

Задача 28: Создайте класс Transaction с полями тип операции, сумма и дата. Реализуйте методы update\_amount(new\_amount), который обновляет сумму, и display\_info(), который выводит информацию о финансовой операции.

Задача 29: Определите класс House с полями адрес, количество комнат и площадь. Реализуйте методы update\_area(new\_area), который обновляет площадь, и display\_info(), который выводит информацию о доме.

Задача 30: Создайте класс Project с полями название, описание и дата начала. Реализуйте методы update\_description(new\_description), который обновляет описание, и display\_info(), который выводит информацию о проекте.

Задача 31: Определите класс Building с полями адрес, этажи и год постройки. Реализуйте методы update\_floors(new\_floors), который обновляет количество этажей, и display\_info(), который выводит информацию о здании.

Задача 32: Создайте класс TrainingCourse с полями название, продолжительность и стоимость. Реализуйте методы update\_cost(new\_cost), который обновляет стоимость, и display\_info(), который выводит информацию о курсе.

Задача 33: Определите класс System с полями название и версия. Реализуйте методы update\_version(new\_version), который обновляет версию, и display\_info(), который выводит информацию о системе.

Задача 34: Создайте класс Rental с полями объект аренды, дата начала и дата окончания. Реализуйте методы update\_dates(new\_start\_date, new\_end\_date), который обновляет даты аренды, и display\_info(), который выводит информацию о аренде.

Задача 35: Определите класс Map с полями название и масштаб. Реализуйте методы update\_scale(new\_scale), который обновляет масштаб, и display\_info(), который выводит информацию о карте.

## 2. Наследование

Пример решений задач

Задача 1. Определите класс Employee, который наследуется от класса Person. Класс Employee должен добавлять поля должность и зарплата, а также методы для обновления должности и отображения информации о работнике.

|  |
| --- |
| С++ |
| #include <iostream>  #include <string>  // Определяем класс Person  class Person {  protected:  std::string name; // Имя человека  int age; // Возраст человека  public:  // Конструктор для инициализации имени и возраста  Person(const std::string& name, int age) : name(name), age(age) {}  // Метод для отображения информации о человеке  void displayInfo() const {  std::cout << "Name: " << name << ", Age: " << age << std::endl;  }  };  // Определяем класс Employee, который наследуется от класса Person  class Employee : public Person {  private:  std::string position; // Должность работника  double salary; // Зарплата работника  public:  // Конструктор для инициализации имени, возраста, должности и зарплаты  Employee(const std::string& name, int age, const std::string& position, double salary)  : Person(name, age), position(position), salary(salary) {}  // Метод для обновления должности  void updatePosition(const std::string& newPosition) {  position = newPosition;  }  // Метод для отображения информации о работнике  void displayEmployeeInfo() const {  displayInfo(); // Вызов метода из базового класса  std::cout << "Position: " << position << ", Salary: " << salary << std::endl;  }  };  int main() {  // Создаем объект Employee  Employee emp("John Doe", 30, "Software Engineer", 60000);  // Отображаем информацию о работнике  emp.displayEmployeeInfo();  // Обновляем должность работника  emp.updatePosition("Senior Software Engineer");  // Отображаем обновленную информацию о работнике  emp.displayEmployeeInfo();  return 0;  } |
| C# |
| using System;  public class Person  {  protected string Name; // Имя человека  protected int Age; // Возраст человека  // Конструктор для инициализации имени и возраста  public Person(string name, int age)  {  Name = name;  Age = age;  }  // Метод для отображения информации о человеке  public void DisplayInfo()  {  Console.WriteLine($"Name: {Name}, Age: {Age}");  }  }  public class Employee : Person  {  private string Position; // Должность работника  private double Salary; // Зарплата работника  // Конструктор для инициализации имени, возраста, должности и зарплаты  public Employee(string name, int age, string position, double salary)  : base(name, age)  {  Position = position;  Salary = salary;  }  // Метод для обновления должности  public void UpdatePosition(string newPosition)  {  Position = newPosition;  }  // Метод для отображения информации о работнике  public void DisplayEmployeeInfo()  {  DisplayInfo(); // Вызов метода из базового класса  Console.WriteLine($"Position: {Position}, Salary: {Salary}");  }  }  class Program  {  static void Main()  {  // Создаем объект Employee  Employee emp = new Employee("John Doe", 30, "Software Engineer", 60000);  // Отображаем информацию о работнике  emp.DisplayEmployeeInfo();  // Обновляем должность работника  emp.UpdatePosition("Senior Software Engineer");  // Отображаем обновленную информацию о работнике  emp.DisplayEmployeeInfo();  }  } |
| Java |
| class Person {  protected String name; // Имя человека  protected int age; // Возраст человека  // Конструктор для инициализации имени и возраста  public Person(String name, int age) {  this.name = name;  this.age = age;  }  // Метод для отображения информации о человеке  public void displayInfo() {  System.out.println("Name: " + name + ", Age: " + age);  }  }  class Employee extends Person {  private String position; // Должность работника  private double salary; // Зарплата работника  // Конструктор для инициализации имени, возраста, должности и зарплаты  public Employee(String name, int age, String position, double salary) {  super(name, age);  this.position = position;  this.salary = salary;  }  // Метод для обновления должности  public void updatePosition(String newPosition) {  this.position = newPosition;  }  // Метод для отображения информации о работнике  public void displayEmployeeInfo() {  displayInfo(); // Вызов метода из базового класса  System.out.println("Position: " + position + ", Salary: " + salary);  }  }  public class Main {  public static void main(String[] args) {  // Создаем объект Employee  Employee emp = new Employee("John Doe", 30, "Software Engineer", 60000);  // Отображаем информацию о работнике  emp.displayEmployeeInfo();  // Обновляем должность работника  emp.updatePosition("Senior Software Engineer");  // Отображаем обновленную информацию о работнике  emp.displayEmployeeInfo();  }  } |
| Python |
| class Person:  def \_\_init\_\_(self, name, age):  self.name = name # Имя человека  self.age = age # Возраст человека  def display\_info(self):  print(f"Name: {self.name}, Age: {self.age}")  class Employee(Person):  def \_\_init\_\_(self, name, age, position, salary):  super().\_\_init\_\_(name, age) # Вызов конструктора базового класса  self.position = position # Должность работника  self.salary = salary # Зарплата работника  def update\_position(self, new\_position):  self.position = new\_position  def display\_employee\_info(self):  self.display\_info() # Вызов метода из базового класса  print(f"Position: {self.position}, Salary: {self.salary}")  if \_\_name\_\_ == "\_\_main\_\_":  # Создаем объект Employee  emp = Employee("John Doe", 30, "Software Engineer", 60000)  # Отображаем информацию о работнике  emp.display\_employee\_info()  # Обновляем должность работника  emp.update\_position("Senior Software Engineer")  # Отображаем обновленную информацию о работнике  emp.display\_employee\_info() |

**Обратите внимание:** В отличие от предыдущего примера, в этом случае используется модификатор доступа protected, чтобы поле класса было доступно в наследниках.

### Задачи для самостоятельной работы

Задача 2. Определите класс Recipe, который наследуется от класса Document. Класс Recipe должен добавлять поля ингредиенты и время приготовления, а также методы для обновления времени приготовления и отображения информации о рецепте.

Задача 3. Определите класс CartItem, который наследуется от класса Product. Класс CartItem должен добавлять поле количество и методы для обновления количества и вычисления общей стоимости товаров в корзине.

Задача 4. Определите класс Gadget, который наследуется от класса Product. Класс Gadget должен добавлять поле тип и методы для обновления типа и отображения информации о гаджете.

Задача 5. Определите класс Queue, который наследуется от класса Stack. Класс Queue должен переопределить методы для работы в виде очереди (FIFO).

Задача 6. Определите класс Circle, который наследуется от класса Shape. Класс Circle должен реализовать метод getArea() для вычисления площади круга.

Задача 7. Определите класс Rectangle, который наследуется от класса Shape. Класс Rectangle должен реализовать метод getArea() для вычисления площади прямоугольника.

Задача 8. Определите класс Car, который наследуется от класса Vehicle. Класс Car должен добавлять методы для работы с автомобильными функциями.

Задача 9. Определите класс Bicycle, который наследуется от класса Vehicle. Класс Bicycle должен добавлять методы для работы с велосипедными функциями.

Задача 10. Определите класс Course, который наследуется от класса Event. Класс Course должен добавлять поля код курса и количество кредитов, а также методы для обновления количества кредитов.

Задача 11. Определите класс LeaveRequest, который наследуется от класса Event. Класс LeaveRequest должен добавлять поля начальная дата и конечная дата, а также методы для обновления дат отпуска.

Задача 12. Определите класс Client, который наследуется от класса Profile. Класс Client должен добавлять поле баланс и методы для обновления баланса и отображения информации о клиенте.

Задача 13. Определите класс Contact, который наследуется от класса Profile. Класс Contact должен добавлять поля номер телефона и электронная почта, а также методы для обновления номера телефона.

Задача 14. Определите класс Project, который наследуется от класса Task. Класс Project должен добавлять поле дата начала и методы для обновления описания и отображения информации о проекте.

Задача 15. Определите класс Computer, который наследуется от класса Gadget. Класс Computer должен добавлять поле год выпуска и методы для обновления модели и отображения информации о компьютере.

Задача 16. Определите класс Book, который наследуется от класса Document. Класс Book должен добавлять поле ISBN и методы для обновления автора и отображения информации о книге.

Задача 17. Определите класс CurrencyRate, который наследуется от класса Document. Класс CurrencyRate должен добавлять поле код валюты и методы для обновления обменного курса и отображения информации о валюте.

Задача 18. Определите класс Account, который наследуется от класса Client. Класс Account должен добавлять методы для пополнения счёта, списания средств и отображения информации о счёте.

Задача 19. Определите класс Company, который наследуется от класса Document. Класс Company должен добавлять поле адрес и методы для обновления адреса и отображения информации о компании.

Задача 20. Определите класс Employee, который наследуется от класса Person. Класс Employee должен добавлять методы для обновления должности и отображения информации о работнике.

Задача 21. Определите класс Movie, который наследуется от класса Event. Класс Movie должен добавлять поля режиссёр и год выпуска, а также методы для обновления режиссёра и отображения информации о фильме.

Задача 22. Определите класс University, который наследуется от класса Company. Класс University должен добавлять поле количество студентов и методы для добавления студентов и отображения информации о университете.

Задача 23. Определите класс Payment, который наследуется от класса Transaction. Класс Payment должен добавлять поле сумма и методы для обновления суммы и отображения информации о платеже.

Задача 24. Определите класс Invoice, который наследуется от класса Payment. Класс Invoice должен добавлять методы для обновления суммы и отображения информации о платеже.

Задача 25. Определите класс Recipe, который наследуется от класса Document. Класс Recipe должен добавлять методы для обновления времени приготовления и отображения информации о рецепте.

Задача 26. Определите класс Gadget, который наследуется от класса Product. Класс Gadget должен добавлять методы для обновления цены и отображения информации о гаджете.

Задача 27. Определите класс Transaction, который наследуется от класса Invoice. Класс Transaction должен добавлять методы для обновления суммы и отображения информации о финансовой операции.

Задача 28. Определите класс House, который наследуется от класса Building. Класс House должен добавлять поле площадь и методы для обновления площади и отображения информации о доме.

Задача 29. Определите класс Project, который наследуется от класса Task. Класс Project должен добавлять поле дата начала и методы для обновления описания и отображения информации о проекте.

Задача 30. Определите класс Building, который наследуется от класса House. Класс Building должен добавлять поле этажи и методы для обновления этажей и отображения информации о здании.

Задача 31. Определите класс TrainingCourse, который наследуется от класса Course. Класс TrainingCourse должен добавлять поле продолжительность и методы для обновления стоимости и отображения информации о курсе.

Задача 32. Определите класс System, который наследуется от класса Software. Класс System должен добавлять поле версия и методы для обновления версии и отображения информации о системе.

Задача 33. Определите класс Rental, который наследуется от класса Transaction. Класс Rental должен добавлять поле объект аренды и методы для обновления дат аренды и отображения информации о аренде.

Задача 34. Определите класс Map, который наследуется от класса Document. Класс Map должен добавлять поле масштаб и методы для обновления масштаба и отображения информации о карте.

Задача 35. Определите класс FinancialProject, который наследуется от класса Project. Класс FinancialProject должен добавлять методы для работы с финансовыми операциями, такими как учет транзакций и управление бюджетом.

## 3. Инкапсуляция

Пример решений задач

Задача 1. Создать класс Employee, у которого поля name, position и salary будут приватными. Реализовать методы getName(), getPosition(), getSalary() для доступа к ним и методы setPosition(), setSalary() для изменения позиции и зарплаты.

|  |
| --- |
| С++ |
| #include <iostream>  #include <string>  // Класс Employee  class Employee {  private:  std::string name; // Имя работника  std::string position; // Должность  double salary; // Зарплата  public:  // Конструктор  Employee(std::string name, std::string position, double salary)  : name(name), position(position), salary(salary) {}  // Методы доступа  std::string getName() const {  return name;  }  std::string getPosition() const {  return position;  }  double getSalary() const {  return salary;  }  // Методы изменения  void setPosition(const std::string& newPosition) {  position = newPosition;  }  void setSalary(double newSalary) {  salary = newSalary;  }  };  int main() {  Employee emp("Alice", "Developer", 75000);  std::cout << "Name: " << emp.getName() << std::endl;  std::cout << "Position: " << emp.getPosition() << std::endl;  std::cout << "Salary: " << emp.getSalary() << std::endl;  emp.setPosition("Senior Developer");  emp.setSalary(85000);  std::cout << "Updated Position: " << emp.getPosition() << std::endl;  std::cout << "Updated Salary: " << emp.getSalary() << std::endl;  return 0;  } |
| C# |
| using System;  // Класс Employee  public class Employee  {  private string name; // Имя работника  private string position; // Должность  private double salary; // Зарплата  // Конструктор  public Employee(string name, string position, double salary)  {  this.name = name;  this.position = position;  this.salary = salary;  }  // Методы доступа  public string GetName()  {  return name;  }  public string GetPosition()  {  return position;  }  public double GetSalary()  {  return salary;  }  // Методы изменения  public void SetPosition(string newPosition)  {  position = newPosition;  }  public void SetSalary(double newSalary)  {  salary = newSalary;  }  }  class Program  {  static void Main()  {  Employee emp = new Employee("Alice", "Developer", 75000);  Console.WriteLine("Name: " + emp.GetName());  Console.WriteLine("Position: " + emp.GetPosition());  Console.WriteLine("Salary: " + emp.GetSalary());  emp.SetPosition("Senior Developer");  emp.SetSalary(85000);  Console.WriteLine("Updated Position: " + emp.GetPosition());  Console.WriteLine("Updated Salary: " + emp.GetSalary());  }  } |
| Java |
| // Класс Employee  class Employee {  private String name; // Имя работника  private String position; // Должность  private double salary; // Зарплата  // Конструктор  public Employee(String name, String position, double salary) {  this.name = name;  this.position = position;  this.salary = salary;  }  // Методы доступа  public String getName() {  return name;  }  public String getPosition() {  return position;  }  public double getSalary() {  return salary;  }  // Методы изменения  public void setPosition(String newPosition) {  position = newPosition;  }  public void setSalary(double newSalary) {  salary = newSalary;  }  }  public class Main {  public static void main(String[] args) {  Employee emp = new Employee("Alice", "Developer", 75000);  System.out.println("Name: " + emp.getName());  System.out.println("Position: " + emp.getPosition());  System.out.println("Salary: " + emp.getSalary());  emp.setPosition("Senior Developer");  emp.setSalary(85000);  System.out.println("Updated Position: " + emp.getPosition());  System.out.println("Updated Salary: " + emp.getSalary());  }  } |
| Python |
| # Класс Employee  class Employee:  def \_\_init\_\_(self, name, position, salary):  self.\_\_name = name # Имя работника  self.\_\_position = position # Должность  self.\_\_salary = salary # Зарплата  # Методы доступа  def get\_name(self):  return self.\_\_name  def get\_position(self):  return self.\_\_position  def get\_salary(self):  return self.\_\_salary  # Методы изменения  def set\_position(self, new\_position):  self.\_\_position = new\_position  def set\_salary(self, new\_salary):  self.\_\_salary = new\_salary  # Пример использования  if \_\_name\_\_ == "\_\_main\_\_":  emp = Employee("Alice", "Developer", 75000)  print("Name:", emp.get\_name())  print("Position:", emp.get\_position())  print("Salary:", emp.get\_salary())  emp.set\_position("Senior Developer")  emp.set\_salary(85000)  print("Updated Position:", emp.get\_position())  print("Updated Salary:", emp.get\_salary()) |

**Обратите внимание:** В методе сеттера следует учитывать, что он не просто устанавливает значение, а делает это в зависимости от прав доступа и целей задачи. Это позволит улучшить инкапсуляцию. Например, метод setSalary можно переписать следующим образом:

|  |
| --- |
| public void setSalary(double newSalary) {  if (hasAccess()) { // Проверка прав доступа  if (newSalary >= 0) {  salary = newSalary;  } else {  salary = 0; // Установка значения по умолчанию  }  }  } |

### Задачи для самостоятельной работы

Задача 2. Создать класс Recipe, у которого поля name, ingredients, cookingTime будут приватными. Реализовать методы для безопасного доступа и изменения этих полей, чтобы контролировать изменения ингредиентов и времени приготовления.

Задача 3. Создать класс CartItem, в котором поля productName, quantity и price будут приватными. Реализовать методы для получения их значений и изменения количества товара с проверкой на допустимые значения.

Задача 4. Создать класс Gadget, у которого поля name, type и price будут приватными. Реализовать методы для безопасного доступа и изменения цены, а также методы для получения названия и типа гаджета.

Задача 5. Создать класс Queue с приватным массивом для хранения элементов. Реализовать методы для добавления элемента в очередь и извлечения из неё, скрывая реализацию работы с массивом.

Задача 6. Создать класс Circle, в котором поля radius и color будут приватными. Реализовать методы для изменения радиуса и цвета с проверкой допустимых значений.

Задача 7. Создать класс Rectangle, в котором поля width, height и color будут приватными. Реализовать методы для доступа и изменения этих полей, контролируя, чтобы ширина и высота были положительными.

Задача 8. Создать класс Car, у которого поля brand, model и year будут приватными. Реализовать методы для доступа к этим полям и изменения модели с проверкой на валидность введенных данных.

Задача 9. Создать класс Bicycle, в котором поля type, gearCount и color будут приватными. Реализовать методы для безопасного доступа и изменения типа велосипеда и количества передач.

Задача 10. Создать класс Course, в котором поля courseName, courseCode и credits будут приватными. Реализовать методы для доступа и изменения количества кредитов с проверкой на допустимые значения.

Задача 11. Создать класс LeaveRequest, у которого поля employeeName, startDate, endDate будут приватными. Предоставить методы для безопасного изменения дат отпуска и доступа к имени сотрудника.

Задача 12. Создать класс Client, в котором поля name, idNumber и balance будут приватными. Реализовать методы для безопасного изменения баланса и доступа к имени и идентификационному номеру клиента.

Задача 13. Создать класс Contact, у которого поля name, phoneNumber и email будут приватными. Реализовать методы для изменения номера телефона и получения информации о контакте с проверкой корректности вводимых данных.

Задача 14. Создать класс Project, в котором поля title, description и startDate будут приватными. Реализовать методы для безопасного изменения описания и даты начала проекта.

Задача 15. Создать класс Computer, у которого поля brand, model и releaseYear будут приватными. Реализовать методы для изменения модели и получения информации о компьютере с валидацией вводимых данных.

Задача 16. Создать класс Book, в котором поля title, author и ISBN будут приватными. Реализовать методы для безопасного изменения автора и доступа к информации о книге.

Задача 17. Создать класс CurrencyRate, у которого поля currencyName, currencyCode и exchangeRate будут приватными. Реализовать методы для безопасного изменения обменного курса и доступа к информации о валюте.

Задача 18. Создать класс Account, в котором поля accountNumber и balance будут приватными. Реализовать методы для безопасного пополнения и списания средств, а также получения информации о счёте.

Задача 19. Создать класс Company, в котором поля companyName и address будут приватными. Реализовать методы для безопасного изменения адреса и доступа к названию компании.

Задача 20. Создать класс Employee, в котором поля name, position и salary будут приватными. Реализовать методы для изменения позиции и зарплаты с проверкой валидности данных и получения информации о работнике.

Задача 21. Создать класс Movie, у которого поля title, director и releaseYear будут приватными. Реализовать методы для изменения режиссёра и получения информации о фильме.

Задача 22. Создать класс University, в котором поля name и studentCount будут приватными. Реализовать методы для добавления студентов и получения информации о университете, контролируя изменение количества студентов.

Задача 23. Создать класс Payment, в котором поля amount и date будут приватными. Реализовать методы для безопасного изменения суммы платежа и доступа к дате платежа.

Задача 24. Создать класс Invoice, у которого поля invoiceNumber и amount будут приватными. Реализовать методы для безопасного изменения суммы и получения информации о счёте.

Задача 25. Создать класс Recipe, в котором поля name, ingredients и cookingTime будут приватными. Реализовать методы для безопасного изменения времени приготовления и ингредиентов.

Задача 26. Создать класс Gadget, в котором поля name, type и price будут приватными. Реализовать методы для безопасного изменения цены и доступа к информации о гаджете.

Задача 27. Создать класс Transaction, в котором поля transactionType, amount и date будут приватными. Реализовать методы для безопасного изменения суммы и доступа к информации о транзакции.

Задача 28. Создать класс House, в котором поля address, roomCount и area будут приватными. Реализовать методы для безопасного изменения площади и количества комнат, а также доступа к информации о доме.

Задача 29. Создать класс Project, в котором поля title, description и startDate будут приватными. Реализовать методы для безопасного изменения описания и даты начала проекта.

Задача 30. Создать класс Building, в котором поля address, floors и yearBuilt будут приватными. Реализовать методы для безопасного изменения количества этажей и года постройки.

Задача 31. Создать класс TrainingCourse, в котором поля courseName, duration и cost будут приватными. Реализовать методы для безопасного изменения стоимости и длительности курса, а также доступа к его названию.

Задача 32. Создать класс System, в котором поля name и version будут приватными. Реализовать методы для безопасного изменения версии и доступа к информации о системе.

Задача 33. Создать класс Rental, в котором поля rentalObject, startDate и endDate будут приватными. Реализовать методы для безопасного изменения дат аренды и доступа к информации об объекте аренды.

Задача 34. Создать класс Map, в котором поля name и scale будут приватными. Реализовать методы для безопасного изменения масштаба и доступа к названию карты.

Задача 35. Создать класс FinancialOperation, у которого поля operationType, amount и date будут приватными. Реализовать методы для безопасного изменения суммы и доступа к информации о финансовой операции.

## 4. Композиция и агрегация

### Пример решений задач

Композиция

Задача 1. Книга и глава. Создайте класс `Книга`, который содержит информацию о книге: название и автора. Также создайте класс `Глава`, представляющий отдельную главу в книге. У класса `Книга` должно быть поле, содержащее список глав, и метод для добавления новой главы с указанием названия и количества страниц. Глава не может существовать без книги, и должна включать поля `название` и `количество страниц`.

|  |
| --- |
| С++ |
| #include <iostream>  #include <string>  #include <vector>  class Book {  public:  // Конструктор книги, принимает название книги и имя автора  Book(const std::string& title, const std::string& author)  : title\_(title), author\_(author) {}  // Метод для добавления новой главы  void addChapter(const std::string& chapterTitle, int pageCount) {  chapters\_.emplace\_back(chapterTitle, pageCount);  }  // Метод для вывода информации о книге и её главах  void printInfo() const {  std::cout << "Книга: " << title\_ << "\nАвтор: " << author\_ << std::endl;  std::cout << "Список глав:" << std::endl;  for (const auto& chapter : chapters\_) {  std::cout << "- " << chapter.getTitle()  << " (" << chapter.getPageCount() << " страниц)"  << std::endl;  }  }  private:  // Вложенный класс Chapter для представления главы книги  class Chapter {  public:  Chapter(const std::string& title, int pageCount)  : title\_(title), pageCount\_(pageCount) {}  const std::string& getTitle() const { return title\_; }  int getPageCount() const { return pageCount\_; }  private:  std::string title\_;  int pageCount\_;  };  std::string title\_; // Название книги  std::string author\_; // Автор книги  std::vector<Chapter> chapters\_; // Список глав  };  int main() {  // Создаем объект книги  Book myBook("Война и мир", "Лев Толстой");  // Добавляем главы  myBook.addChapter("Часть 1", 100);  myBook.addChapter("Часть 2", 120);  myBook.addChapter("Часть 3", 140);  // Выводим информацию о книге и главах  myBook.printInfo();  return 0;  } |
| C# |
| using System;  using System.Collections.Generic;  public class Book  {  // Поля для названия книги и автора  public string Title { get; private set; }  public string Author { get; private set; }    // Список глав  private List<Chapter> chapters;  // Конструктор для инициализации книги  public Book(string title, string author)  {  Title = title;  Author = author;  chapters = new List<Chapter>();  }  // Метод для добавления новой главы  public void AddChapter(string chapterTitle, int pageCount)  {  chapters.Add(new Chapter(chapterTitle, pageCount));  }  // Метод для вывода информации о книге и главах  public void PrintInfo()  {  Console.WriteLine($"Книга: {Title}");  Console.WriteLine($"Автор: {Author}");  Console.WriteLine("Список глав:");  foreach (var chapter in chapters)  {  Console.WriteLine($"- {chapter.Title} ({chapter.PageCount} страниц)");  }  }  // Вложенный класс Chapter, который представляет главу книги  private class Chapter  {  public string Title { get; private set; }  public int PageCount { get; private set; }  public Chapter(string title, int pageCount)  {  Title = title;  PageCount = pageCount;  }  }  }  public class Program  {  public static void Main()  {  // Создаем объект книги  Book myBook = new Book("Война и мир", "Лев Толстой");  // Добавляем главы  myBook.AddChapter("Часть 1", 100);  myBook.AddChapter("Часть 2", 120);  myBook.AddChapter("Часть 3", 140);  // Выводим информацию о книге и главах  myBook.PrintInfo();  }  } |
| Java |
| import java.util.ArrayList;  import java.util.List;  class Book {  private String title; // Название книги  private String author; // Автор книги  private List<Chapter> chapters; // Список глав  // Конструктор книги  public Book(String title, String author) {  this.title = title;  this.author = author;  this.chapters = new ArrayList<>(); // Инициализация списка глав  }  // Метод для добавления новой главы  public void addChapter(String chapterTitle, int pageCount) {  chapters.add(new Chapter(chapterTitle, pageCount));  }  // Метод для вывода информации о книге и главах  public void printInfo() {  System.out.println("Книга: " + title);  System.out.println("Автор: " + author);  System.out.println("Список глав:");  for (Chapter chapter : chapters) {  System.out.println("- " + chapter.getTitle() + " (" + chapter.getPageCount() + " страниц)");  }  }  // Вложенный класс Chapter, представляющий главу книги  private class Chapter {  private String title; // Название главы  private int pageCount; // Количество страниц в главе  // Конструктор главы  public Chapter(String title, int pageCount) {  this.title = title;  this.pageCount = pageCount;  }  // Геттеры для получения данных о главе  public String getTitle() {  return title;  }  public int getPageCount() {  return pageCount;  }  }  public static void main(String[] args) {  // Создаем книгу  Book book = new Book("Война и мир", "Лев Толстой");  // Добавляем главы в книгу  book.addChapter("Часть 1", 100);  book.addChapter("Часть 2", 120);  book.addChapter("Часть 3", 150);  // Выводим информацию о книге и главах  book.printInfo();  }  } |
| Python |
| class Book:  def \_\_init\_\_(self, title, author):  self.title = title # Название книги  self.author = author # Автор книги  self.chapters = [] # Список глав  def add\_chapter(self, chapter\_title, page\_count):  """Добавить новую главу в книгу"""  new\_chapter = self.Chapter(chapter\_title, page\_count)  self.chapters.append(new\_chapter)  def print\_info(self):  """Вывести информацию о книге и главах"""  print(f"Книга: {self.title}")  print(f"Автор: {self.author}")  print("Список глав:")  for chapter in self.chapters:  print(f"- {chapter.title} ({chapter.page\_count} страниц)")  class Chapter:  def \_\_init\_\_(self, title, page\_count):  self.title = title # Название главы  self.page\_count = page\_count # Количество страниц в главе  # Пример использования  # Создаем книгу  book = Book("Война и мир", "Лев Толстой")  # Добавляем главы  book.add\_chapter("Часть 1", 100)  book.add\_chapter("Часть 2", 120)  book.add\_chapter("Часть 3", 150)  # Выводим информацию о книге и главах  book.print\_info() |

### Задачи для самостоятельной работы

Задача 2. Компьютер, процессор и оперативная память. Разработайте класс `Компьютер`, который включает в себя компоненты `Процессор` и `Оперативная память`. Класс `Компьютер` должен иметь методы для получения характеристик, включая скорость процессора и объем памяти. Класс `Процессор` должен содержать поля `модель` и `скорость`, а класс `Оперативная память` — поле `емкость`. Процессор и память не должны существовать отдельно от компьютера.

Задача 3. Шкаф и полка. Создайте класс `Шкаф`, который содержит список полок. У класса `Шкаф` должен быть метод для добавления полки и подсчета количества полок. Класс `Полка` должен содержать список предметов, который можно пополнять методами добавления предметов. Полка не может существовать без шкафа.

Задача 4. Автомобиль и двигатель. Реализуйте класс `Автомобиль`, который включает двигатель. У автомобиля должны быть методы `запуск` и `остановка` двигателя. Двигатель, описываемый классом `Двигатель`, должен содержать поле `мощность` и метод для вывода информации о мощности. Двигатель не может существовать отдельно от автомобиля.

Задача 5. Человек и сердце. Создайте класс `Человек`, который содержит сердце. У человека должен быть метод для проверки, жив ли он (работает ли сердце). Класс `Сердце` должен содержать поле `частота сердцебиения` и метод для реализации биения сердца. Сердце не может существовать без человека.

Задача 6. Мобильный телефон и батарея. Постройте класс `Мобильный телефон`, который включает батарею. Телефон должен иметь метод для проверки состояния батареи. Класс `Батарея` должен включать `емкость` и `уровень заряда`, а также метод для подзарядки. Батарея не может существовать отдельно от телефона.

Задача 7. Дом и комната. Разработайте класс `Дом`, который содержит список комнат. В классе `Дом` реализуйте метод для добавления комнат с указанием их типа, например, "спальня" или "кухня". Класс `Комната` должен иметь поле `тип комнаты` и метод для возвращения его описания. Комната не может существовать отдельно от дома.

Задача 8. Здание и этаж. Создайте класс `Здание`, в котором может быть несколько этажей. Реализуйте метод для добавления нового этажа с указанием его номера. Класс `Этаж` должен содержать поле `номер этажа` и метод для описания этажа. Этаж не может существовать отдельно от здания.

Задача 9. Фруктовый сок и фрукт. Постройте класс `Фруктовый сок`, который включает в себя фрукты. Реализуйте метод для добавления фруктов (например, яблок или апельсинов) в сок. Класс `Фрукт` должен иметь поле `тип фрукта`. Фрукты не могут существовать отдельно от сока.

Задача 10. Машина и топливный бак. Создайте класс `Машина`, содержащий топливный бак. В классе `Машина` должны быть методы для заправки. Класс `Топливный бак` должен иметь поля `емкость` и `уровень топлива`, а также метод для его наполнения. Топливный бак не может существовать отдельно от машины.

Задача 11. Лес и дерево. Создайте класс `Лес`, содержащий множество деревьев. В классе `Лес` реализуйте метод для добавления нового дерева с указанием его вида. Класс `Дерево` должен включать поле `вид`. Деревья не могут существовать отдельно от леса.

Задача 12. Компания и отдел. Создайте класс `Компания`, который может содержать несколько отделов. Реализуйте метод для добавления отдела с его названием. Класс `Отдел` должен иметь поле `название отдела`. Отделы не могут существовать без компании.

Задача 13. Команда и игрок. Постройте класс `Команда`, который включает список игроков. В классе `Команда` должен быть метод для добавления игроков с указанием их имен и позиций. Класс `Игрок` должен иметь поля `имя` и `позиция`. Игроки не могут существовать без команды.

Задача 14. Аэропорт и самолет. Реализуйте класс `Аэропорт`, который содержит самолеты. Класс `Аэропорт` должен иметь метод для добавления самолета. Класс `Самолет` должен содержать поле `модель самолета`. Самолет не может существовать вне аэропорта.

Задача 15. Ресторан и меню. Постройте класс `Ресторан`, включающий меню. У ресторана должен быть метод для отображения блюд из меню. Класс `Меню` должен содержать список блюд и метод для добавления нового блюда. Меню не может существовать без ресторана.

Задача 16. Музей и экспонат. Создайте класс `Музей`, содержащий список экспонатов. Реализуйте метод для добавления нового экспоната. Класс `Экспонат` должен иметь поле `название экспоната`. Экспонаты не могут существовать вне музея.

Задача 17. Издательство и журнал. Создайте класс `Издательство`, которое издает журналы. Класс `Журнал` должен содержать поле `название`. Журнал не может существовать вне издательства.

Задача 18. Театр и спектакль. Постройте класс `Театр`, включающий в себя спектакли. В театре реализуйте метод для добавления спектакля с названием и временем начала. Класс `Спектакль` должен содержать поле `название спектакля`. Спектакль не может существовать без театра.

Задача 19. Фабрика и продукт. Создайте класс `Фабрика`, которая выпускает продукты. Класс `Фабрика` должен содержать метод для создания продукта. Класс `Продукт` должен иметь поле `название продукта`. Продукты не могут существовать без фабрики.

Задача 20. Кинотеатр и фильм. Создайте класс `Кинотеатр`, в котором идут фильмы. Реализуйте метод для добавления фильма. Класс `Фильм` должен содержать поле `название фильма`. Фильмы не могут существовать вне кинотеатра.

Задача 21. Больница и палата. Создайте класс `Больница`, который содержит палаты. Класс `Больница` должен иметь метод для добавления палат с указанием их номеров и количества коек. Класс `Палата` должен содержать поля `номер палаты` и `количество коек`. Палата не может существовать без больницы.

Задача 22. Школа и класс. Реализуйте класс `Школа`, включающий в себя классы (группы учеников). Класс `Школа` должен иметь метод для добавления новых классов с указанием их номеров или имен. Класс `Класс` должен содержать поле `название` и метод для добавления учеников. Классы не могут существовать вне школы.

Задача 23. Магазин и товар. Создайте класс `Магазин`, содержащий список товаров. Реализуйте метод для добавления товаров в магазин. Класс `Товар` должен иметь поля `название товара` и `цена`. Товар не может существовать без магазина.

Задача 24. Университет и факультет. Постройте класс `Университет`, который состоит из факультетов. В классе `Университет` реализуйте метод для добавления факультетов. Класс `Факультет` должен включать поля `название факультета` и `список студентов`. Факультеты не могут существовать без университета.

Задача 25. Велосипед и колесо. Разработайте класс `Велосипед`, состоящий из колес. Класс `Велосипед` должен содержать метод для добавления колес и проверки их состояния. Класс `Колесо` должен содержать поле `диаметр`. Колеса не могут существовать отдельно от велосипеда.

Задача 26. Парк и аттракцион. Создайте класс `Парк`, который включает в себя аттракционы. У класса `Парк` должен быть метод для добавления аттракционов. Класс `Аттракцион` должен иметь поле `название` и метод для запуска аттракциона. Аттракционы не могут существовать вне парка.

Задача 27. Поезд и вагон. Создайте класс `Поезд`, состоящий из вагонов. В классе `Поезд` реализуйте метод для добавления вагонов. Класс `Вагон` должен содержать поле `номер вагона`. Вагоны не могут существовать отдельно от поезда.

Задача 28. Торт и корж. Постройте класс `Торт`, состоящий из коржей. Класс `Торт` должен иметь метод для добавления коржей. Класс `Корж` должен содержать поле `вкус`. Коржи не могут существовать вне торта.

Задача 29. Сад и растение. Создайте класс `Сад`, который содержит растения. Реализуйте метод для добавления растений с указанием их видов. Класс `Растение` должен содержать поле `вид растения`. Растения не могут существовать без сада.

Задача 30. Оркестр и музыкальный инструмент. Создайте класс `Оркестр`, включающий в себя музыкальные инструменты. В классе `Оркестр` реализуйте метод для добавления инструмента. Класс `Инструмент` должен содержать поле `название`. Инструменты не могут существовать вне оркестра.

Задача 31. Город и район. Реализуйте класс `Город`, состоящий из районов. Класс `Город` должен содержать метод для добавления районов. Класс `Район` должен иметь поле `название`. Районы не могут существовать вне города.

Задача 32. Зоопарк и вольер. Создайте класс `Зоопарк`, содержащий вольеры. У зоопарка должен быть метод для добавления вольеров с указанием их номера и типа животных. Класс `Вольер` должен включать поле `тип животных`. Вольеры не могут существовать вне зоопарка.

Задача 33. Галерея и картина. Создайте класс `Галерея`, который содержит картины. Реализуйте метод для добавления картин с указанием автора и названия. Класс `Картина` должен содержать поля `автор` и `название`. Картины не могут существовать вне галереи.

Задача 34. Банк и счет. Разработайте класс `Банк`, содержащий счета клиентов. Класс `Банк` должен иметь метод для добавления счета клиента. Класс `Счет` должен включать поля `владелец` и `баланс`. Счет не может существовать вне банка.

Задача 35. Ферма и животное. Создайте класс `Ферма`, включающий животных. У фермы должен быть метод для добавления животных с указанием их вида. Класс `Животное` должен иметь поле `вид`. Животные не могут существовать без фермы.

Агрегация

Задача 1. Класс и студент. Создайте класс `Класс`, содержащий список студентов. Реализуйте методы для добавления студента и вывода списка студентов. Класс `Студент` должен содержать поля `имя` и `возраст`. Студенты могут существовать независимо от класса.

|  |
| --- |
| С++ |
| #include <iostream>  #include <vector>  #include <string>  // Класс "Студент"  class Student {  public:  // Конструктор для инициализации имени и возраста студента  Student(const std::string& name, int age) : name(name), age(age) {}  // Метод для получения имени студента  std::string getName() const {  return name;  }  // Метод для получения возраста студента  int getAge() const {  return age;  }  private:  std::string name; // Имя студента  int age; // Возраст студента  };  // Класс "Группа"  class Group {  public:  // Метод для добавления студента в группу  void addStudent(const Student& student) {  students.push\_back(student);  }  // Метод для вывода списка студентов в группе  void displayStudents() const {  std::cout << "Список студентов в группе:" << std::endl;  for (const auto& student : students) {  std::cout << "Имя: " << student.getName() << ", Возраст: " << student.getAge() << std::endl;  }  }  private:  std::vector<Student> students; // Список студентов в группе  };  int main() {  // Создание студентов  Student student1("Иван", 20);  Student student2("Мария", 22);  Student student3("Алексей", 21);  // Создание группы и добавление студентов  Group group1;  group1.addStudent(student1);  group1.addStudent(student2);  group1.addStudent(student3);  // Вывод списка студентов в группе  group1.displayStudents();  return 0;  } |
| C# |
| using System;  using System.Collections.Generic;  // Класс "Студент"  public class Student  {  public string Name { get; set; } // Имя студента  public int Age { get; set; } // Возраст студента  // Конструктор для инициализации студента  public Student(string name, int age)  {  Name = name;  Age = age;  }  // Метод для вывода информации о студенте  public void DisplayInfo()  {  Console.WriteLine($"Имя: {Name}, Возраст: {Age}");  }  }  // Класс "Группа"  public class Group  {  private List<Student> students; // Список студентов  // Конструктор для инициализации списка студентов  public Group()  {  students = new List<Student>();  }  // Метод для добавления студента в группу  public void AddStudent(Student student)  {  students.Add(student);  }  // Метод для вывода списка студентов  public void DisplayStudents()  {  Console.WriteLine("Список студентов в группе:");  foreach (var student in students)  {  student.DisplayInfo();  }  }  }  class Program  {  static void Main()  {  // Создание студентов  Student student1 = new Student("Иван", 20);  Student student2 = new Student("Мария", 22);  Student student3 = new Student("Алексей", 21);  // Создание группы и добавление студентов  Group group1 = new Group();  group1.AddStudent(student1);  group1.AddStudent(student2);  group1.AddStudent(student3);  // Вывод списка студентов  group1.DisplayStudents();  }  } |
| Java |
| import java.util.ArrayList;  import java.util.List;  // Класс "Студент"  class Student {  private String name; // Имя студента  private int age; // Возраст студента  // Конструктор для инициализации студента  public Student(String name, int age) {  this.name = name;  this.age = age;  }  // Метод для получения имени студента  public String getName() {  return name;  }  // Метод для получения возраста студента  public int getAge() {  return age;  }  // Метод для вывода информации о студенте  public void displayInfo() {  System.out.println("Имя: " + name + ", Возраст: " + age);  }  }  // Класс "Группа"  class Group {  private List<Student> students; // Список студентов  // Конструктор для инициализации списка студентов  public Group() {  students = new ArrayList<>();  }  // Метод для добавления студента в группу  public void addStudent(Student student) {  students.add(student);  }  // Метод для вывода списка студентов в группе  public void displayStudents() {  System.out.println("Список студентов в группе:");  for (Student student : students) {  student.displayInfo();  }  }  }  public class Main {  public static void main(String[] args) {  // Создание студентов  Student student1 = new Student("Иван", 20);  Student student2 = new Student("Мария", 22);  Student student3 = new Student("Алексей", 21);  // Создание группы и добавление студентов  Group group1 = new Group();  group1.addStudent(student1);  group1.addStudent(student2);  group1.addStudent(student3);  // Вывод списка студентов в группе  group1.displayStudents();  }  } |
| Python |
| # Класс "Студент"  class Student:  def \_\_init\_\_(self, name, age):  self.name = name # Имя студента  self.age = age # Возраст студента  def get\_name(self):  return self.name # Возвращает имя студента  def get\_age(self):  return self.age # Возвращает возраст студента  def display\_info(self):  print(f"Имя: {self.name}, Возраст: {self.age}")  # Класс "Группа"  class Group:  def \_\_init\_\_(self):  self.students = [] # Список студентов  def add\_student(self, student):  self.students.append(student) # Добавляем студента в группу  def display\_students(self):  print("Список студентов в группе:")  for student in self.students:  student.display\_info() # Выводим информацию о каждом студенте  # Пример использования  if \_\_name\_\_ == "\_\_main\_\_":  # Создание студентов  student1 = Student("Иван", 20)  student2 = Student("Мария", 22)  student3 = Student("Алексей", 21)  # Создание группы и добавление студентов  group1 = Group()  group1.add\_student(student1)  group1.add\_student(student2)  group1.add\_student(student3)  # Вывод списка студентов в группе  group1.display\_students() |

### Задачи для самостоятельной работы

Задача 2. Проект и сотрудник. Реализуйте класс `Проект`, в котором работает несколько сотрудников. Класс `Проект` должен иметь метод для добавления сотрудника и метод для вывода команды. Класс `Сотрудник` должен иметь поля `имя` и `должность`. Сотрудники могут существовать независимо от проекта.

Задача 3. Книга и читатель. Создайте класс `Библиотека`, содержащий список книг и читателей. Реализуйте метод для добавления читателя и книги. Класс `Книга` и `Читатель` могут существовать независимо от библиотеки.

Задача 4. Заказ и товар. Реализуйте класс `Заказ`, в который можно добавлять товары. Класс `Товар` должен содержать `название` и `цену`. Товары могут существовать независимо от заказа.

Задача 5. Игровая команда и игрок. Создайте класс `Команда`, который включает в себя игроков. Класс `Игрок` должен иметь поля `имя` и `позиция`. Игроки могут существовать независимо от команды.

Задача 6. Университет и преподаватель. Создайте класс `Университет`, в котором есть список преподавателей. У преподавателей есть поля `имя` и `специальность`. Преподаватели могут существовать независимо от университета.

Задача 7. Курс и студент. Постройте класс `Курс`, который содержит студентов, посещающих этот курс. У класса `Студент` есть поля `имя` и `возраст`. Студенты могут существовать вне курса.

Задача 8. Отель и клиент. Создайте класс `Отель`, содержащий список клиентов, которые могут бронировать комнаты. У класса `Клиент` есть поля `имя` и `контактные данные`. Клиенты могут существовать независимо от отеля.

Задача 9. Комната и мебель. Реализуйте класс `Комната`, в котором может быть установлена мебель. У класса `Мебель` есть поля `тип` и `размер`. Мебель может существовать независимо от комнаты.

Задача 10. Лаборатория и оборудование. Создайте класс `Лаборатория`, в которой есть оборудование. У класса `Оборудование` есть поля `название` и `функция`. Оборудование может существовать вне лаборатории.

Задача 11. Фестиваль и участник. Реализуйте класс `Фестиваль`, включающий список участников. У класса `Участник` есть поля `имя` и `вид искусства`. Участники могут существовать независимо от фестиваля.

Задача 12. Город и здание. Создайте класс `Город`, который содержит здания. Класс `Здание` должен иметь поля `название` и `адрес`. Здания могут существовать независимо от города.

Задача 13. Школа и ученик. Создайте класс `Школа`, который включает список учеников. У класса `Ученик` есть поля `имя` и `возраст`. Ученики могут существовать независимо от школы.

Задача 14. Банк и клиент. Реализуйте класс `Банк`, в котором есть клиенты. У класса `Клиент` есть поля `имя` и `баланс`. Клиенты могут существовать независимо от банка.

Задача 15. Фирма и проект. Создайте класс `Фирма`, содержащий список проектов. У класса `Проект` есть поля `название` и `бюджет`. Проекты могут существовать независимо от фирмы.

Задача 16. Клиент и банковская карта. Реализуйте класс `Клиент`, у которого могут быть несколько банковских карт. У класса `Карта` есть поля `номер` и `лимит`. Карты могут существовать вне клиента.

Задача 17. Концерт и музыкальная группа. Постройте класс `Концерт`, в котором могут выступать музыкальные группы. У класса `Музыкальная группа` есть поля `название` и `жанр`. Группы могут существовать независимо от концерта.

Задача 18. Путешествие и достопримечательность. Создайте класс `Путешествие`, в котором есть достопримечательности, которые посетят путешественники. У класса `Достопримечательность` есть поля `название` и `описание`. Достопримечательности могут существовать вне путешествия.

Задача 19. Производственная линия и продукт. Создайте класс `Производственная линия`, которая выпускает продукты. Класс `Продукт` должен иметь поля `название` и `серийный номер`. Продукты могут существовать независимо от производственной линии.

Задача 20. Сотрудник и рабочий проект. Реализуйте класс `Сотрудник`, который может работать над несколькими проектами. У класса `Проект` есть поля `название` и `дедлайн`. Проекты могут существовать независимо от сотрудников.

Задача 21. Ученый и исследование. Создайте класс `Ученый`, у которого может быть несколько текущих исследований. Класс `Исследование` должен иметь поля `тема` и `цель`. Исследования могут существовать вне учёного.

Задача 22. Водитель и автомобиль. Реализуйте класс `Водитель`, который может водить разные автомобили. У класса `Автомобиль` есть поля `модель` и `номерной знак`. Автомобили могут существовать независимо от водителей.

Задача 23. Турист и страна. Создайте класс `Турист`, который может посещать разные страны. У класса `Страна` есть поля `название` и `столица`. Страны могут существовать независимо от туристов.

Задача 24. Студент и учебный предмет. Создайте класс `Студент`, который может изучать несколько предметов. У класса `Предмет` есть поля `название` и `оценка`. Предметы могут существовать независимо от студента.

Задача 25. Фитнес-клуб и тренер. Реализуйте класс `Фитнес-клуб`, который может сотрудничать с разными тренерами. У класса `Тренер` есть поля `имя` и `специализация`. Тренеры могут существовать вне фитнес-клуба.

Задача 26. Тур и экскурсовод. Создайте класс `Тур`, который может проводиться разными экскурсоводами. У класса `Экскурсовод` есть поля `имя` и `языки`. Экскурсоводы могут существовать независимо от тура.

Задача 27. Музыкальный альбом и песня. Создайте класс `Альбом`, в который входят песни. У класса `Песня` есть поля `название` и `продолжительность`. Песни могут существовать независимо от альбома.

Задача 28. Учебное заведение и курс. Создайте класс `Учебное заведение`, где преподаются курсы. У класса `Курс` есть поля `название` и `программа`. Курсы могут существовать независимо от учебного заведения.

Задача 29. Конференция и доклад. Постройте класс `Конференция`, на которой представляются доклады. У класса `Доклад` есть поля `название` и `автор`. Доклады могут существовать независимо от конференции.

Задача 30. Парк аттракционов и аттракцион. Реализуйте класс `Парк аттракционов`, содержащий разные аттракционы. У класса `Аттракцион` есть поля `название` и `время работы`. Аттракционы могут существовать вне парка.

Задача 31. Учитель и учебный класс. Создайте класс `Учитель`, который может преподавать в нескольких классах. У класса `Класс` есть поля `номер` и `уровень знаний`. Классы могут существовать независимо от учителя.

Задача 32. Автор и книга. Создайте класс `Автор`, который может писать несколько книг. У класса `Книга` есть поля `название` и `год публикации`. Книги могут существовать независимо от автора.

Задача 33. Медицинский центр и врач. Создайте класс `Медицинский центр`, в котором работают врачи. У класса `Врач` есть поля `имя` и `специализация`. Врачи могут существовать вне медицинского центра.

Задача 34. Фильмотека и фильм. Реализуйте класс `Фильмотека`, который включает фильмы. У класса `Фильм` есть поля `название` и `жанр`. Фильмы могут существовать независимо от фильмотеки.

Задача 35. Библиотека и автор. Создайте класс `Библиотека`, содержащий авторов книг. У класса `Автор` есть поля `имя` и `биография`. Авторы могут существовать вне библиотеки.

## 5. Абстрактные классы

Пример решений задач

Задача 1. Создать абстрактный класс Animal с абстрактными методами makeSound() и move(). Реализовать подклассы Dog и Bird, которые реализуют методы makeSound() и move().

|  |
| --- |
| С++ |
| #include <iostream>  using namespace std;  // Абстрактный класс Animal  class Animal {  public:  // Абстрактные методы  virtual void makeSound() = 0; // Чисто виртуальная функция  virtual void move() = 0; // Чисто виртуальная функция  };  // Подкласс Dog  class Dog : public Animal {  public:  void makeSound() override {  cout << "Гав!" << endl; // Реализация метода makeSound  }  void move() override {  cout << "Собака бегает." << endl; // Реализация метода move  }  };  // Подкласс Bird  class Bird : public Animal {  public:  void makeSound() override {  cout << "Чирик!" << endl; // Реализация метода makeSound  }  void move() override {  cout << "Птица летит." << endl; // Реализация метода move  }  };  int main() {  Dog dog;  Bird bird;  dog.makeSound(); // Вызов метода makeSound для собаки  dog.move(); // Вызов метода move для собаки  bird.makeSound(); // Вызов метода makeSound для птицы  bird.move(); // Вызов метода move для птицы  return 0;  } |
| C# |
| using System;  // Абстрактный класс Animal  public abstract class Animal {  // Абстрактные методы  public abstract void MakeSound(); // Абстрактный метод  public abstract void Move(); // Абстрактный метод  }  // Подкласс Dog  public class Dog : Animal {  public override void MakeSound() {  Console.WriteLine("Гав!"); // Реализация метода MakeSound  }  public override void Move() {  Console.WriteLine("Собака бегает."); // Реализация метода Move  }  }  // Подкласс Bird  public class Bird : Animal {  public override void MakeSound() {  Console.WriteLine("Чирик!"); // Реализация метода MakeSound  }  public override void Move() {  Console.WriteLine("Птица летит."); // Реализация метода Move  }  }  class Program {  static void Main() {  Dog dog = new Dog();  Bird bird = new Bird();  dog.MakeSound(); // Вызов метода MakeSound для собаки  dog.Move(); // Вызов метода Move для собаки  bird.MakeSound(); // Вызов метода MakeSound для птицы  bird.Move(); // Вызов метода Move для птицы  }  } |
| Java |
| // Абстрактный класс Animal  abstract class Animal {  // Абстрактные методы  public abstract void makeSound(); // Абстрактный метод  public abstract void move(); // Абстрактный метод  }  // Подкласс Dog  class Dog extends Animal {  public void makeSound() {  System.out.println("Гав!"); // Реализация метода makeSound  }  public void move() {  System.out.println("Собака бегает."); // Реализация метода move  }  }  // Подкласс Bird  class Bird extends Animal {  public void makeSound() {  System.out.println("Чирик!"); // Реализация метода makeSound  }  public void move() {  System.out.println("Птица летит."); // Реализация метода move  }  }  public class Main {  public static void main(String[] args) {  Dog dog = new Dog();  Bird bird = new Bird();  dog.makeSound(); // Вызов метода makeSound для собаки  dog.move(); // Вызов метода move для собаки  bird.makeSound(); // Вызов метода makeSound для птицы  bird.move(); // Вызов метода move для птицы  }  } |
| Python |
| from abc import ABC, abstractmethod  # Абстрактный класс Animal  class Animal(ABC):  @abstractmethod  def make\_sound(self):  pass # Абстрактный метод  @abstractmethod  def move(self):  pass # Абстрактный метод  # Подкласс Dog  class Dog(Animal):  def make\_sound(self):  print("Гав!") # Реализация метода make\_sound  def move(self):  print("Собака бегает.") # Реализация метода move  # Подкласс Bird  class Bird(Animal):  def make\_sound(self):  print("Чирик!") # Реализация метода make\_sound  def move(self):  print("Птица летит.") # Реализация метода move  # Пример использования  if \_\_name\_\_ == "\_\_main\_\_":  dog = Dog()  bird = Bird()  dog.make\_sound() # Вызов метода make\_sound для собаки  dog.move() # Вызов метода move для собаки  bird.make\_sound() # Вызов метода make\_sound для птицы  bird.move() # Вызов метода move для птицы |

### Задачи для самостоятельной работы

Задача 2. Создать абстрактный класс ElectronicDevice с абстрактным методом turnOn(). Реализовать подклассы Smartphone и Laptop, которые реализуют метод turnOn().

Задача 3. Определить абстрактный класс 2DShape с абстрактным методом calculatePerimeter(). Реализовать подклассы Square и Triangle, которые реализуют метод calculatePerimeter().

Задача 4. Создать класс Square, наследующий абстрактный класс 2DShape, с полем sideLength. Реализовать методы для вычисления периметра и площади квадрата.

Задача 5. Определить абстрактный класс Clock с абстрактным методом showTime(). Реализовать подклассы DigitalClock и AnalogClock, которые реализуют метод showTime().

Задача 6. Создать абстрактный класс Fruit с абстрактным методом taste(). Реализовать подклассы Apple и Banana, которые реализуют метод taste().

Задача 7. Определить абстрактный класс Transport с абстрактным методом getFuelEfficiency(). Реализовать подклассы Car и Motorcycle, которые реализуют метод getFuelEfficiency().

Задача 8. Создать абстрактный класс Plan с абстрактным методом calculateCost(). Реализовать подклассы BasicPlan и PremiumPlan, которые реализуют метод calculateCost().

Задача 9. Определить абстрактный класс Plant с абстрактным методом grow(). Реализовать подклассы Flower и Tree, которые реализуют метод grow().

Задача 10. Создать абстрактный класс Ticket с абстрактным методом calculatePrice(). Реализовать подклассы ConcertTicket и MovieTicket, которые реализуют метод calculatePrice().

Задача 11. Создать класс Circle, наследующий абстрактный класс 2DShape, с полем radius. Реализовать методы для вычисления периметра и площади круга.

Задача 12. Создать абстрактный класс DisplayDevice с абстрактным методом display(). Реализовать подклассы Monitor и Projector, которые реализуют метод display().

Задача 13. Определить абстрактный класс User с абстрактными методами login() и logout(). Реализовать подклассы AdminUser и RegularUser, которые реализуют методы login() и logout().

Задача 14. Создать абстрактный класс LearningCourse с абстрактным методом getDuration(). Реализовать подклассы OnlineCourse и InPersonCourse, которые реализуют метод getDuration().

Задача 15. Определить абстрактный класс ListItem с абстрактными методами getName() и getPrice(). Реализовать подклассы BookItem и ElectronicsItem, которые реализуют методы getName() и getPrice().

Задача 16. Создать абстрактный класс CarPart с абстрактным методом replace(). Реализовать подклассы Engine и Wheel, которые реализуют метод replace().

Задача 17. Определить абстрактный класс Room с абстрактными методами calculateArea() и paint(). Реализовать подклассы LivingRoom и Bedroom, которые реализуют методы calculateArea() и paint().

Задача 18. Создать абстрактный класс Bank с абстрактным методом calculateInterest(). Реализовать подклассы SavingsBank и InvestmentBank, которые реализуют метод calculateInterest().

Задача 19. Определить абстрактный класс Function с абстрактным методом evaluate(x). Реализовать подклассы LinearFunction и QuadraticFunction, которые реализуют метод evaluate(x).

Задача 20. Создать абстрактный класс ProjectTask с абстрактными методами startTask() и completeTask(). Реализовать подклассы DevelopmentTask и TestingTask, которые реализуют методы startTask() и completeTask().

Задача 21. Определить абстрактный класс Journal с абстрактными методами publish() и getIssueNumber(). Реализовать подклассы MonthlyJournal и WeeklyJournal, которые реализуют методы publish() и getIssueNumber().

Задача 22. Создать абстрактный класс Game с абстрактным методом play(). Реализовать подклассы BoardGame и CardGame, которые реализуют метод play().

Задача 23. Определить абстрактный класс ChartElement с абстрактным методом draw(). Реализовать подклассы BarChart и LineChart, которые реализуют метод draw().

Задача 24. Создать абстрактный класс AudioDevice с абстрактным методом playSound(). Реализовать подклассы Headphones и Speakers, которые реализуют метод playSound().

Задача 25. Определить абстрактный класс Photo с абстрактным методом edit(). Реализовать подклассы Portrait и Landscape, которые реализуют метод edit().

Задача 26. Создать абстрактный класс WrestlingMatch с абстрактным методом startMatch(). Реализовать подклассы SingleMatch и TagTeamMatch, которые реализуют метод startMatch().

Задача 27. Определить абстрактный класс Tool с абстрактными методами use() и maintain(). Реализовать подклассы Hammer и Screwdriver, которые реализуют методы use() и maintain().

Задача 28. Создать абстрактный класс SportEvent с абстрактным методом schedule(). Реализовать подклассы FootballMatch и BasketballGame, которые реализуют метод schedule().

Задача 29. Определить абстрактный класс DocumentFormat с абстрактными методами save() и load(). Реализовать подклассы PDFFormat и WordFormat, которые реализуют методы save() и load().

Задача 30. Создать абстрактный класс TouristSpot с абстрактным методом getDescription(). Реализовать подклассы Mountain и Beach, которые реализуют метод getDescription().

Задача 31. Определить абстрактный класс Event с абстрактным методом organize(). Реализовать подклассы Conference и Workshop, которые реализуют метод organize().

Задача 32. Создать абстрактный класс SystemComponent с абстрактным методом initialize(). Реализовать подклассы Database и WebServer, которые реализуют метод initialize().

Задача 33. Определить абстрактный класс Form с абстрактными методами submit() и validate(). Реализовать подклассы RegistrationForm и FeedbackForm, которые реализуют методы submit() и validate().

Задача 34. Создать абстрактный класс ManagementSystem с абстрактными методами addRecord() и removeRecord(). Реализовать подклассы EmployeeManagementSystem и InventoryManagementSystem, которые реализуют методы addRecord() и removeRecord().

Задача 35. Определить абстрактный класс InputDevice с абстрактным методом receiveInput(). Реализовать подклассы Keyboard и Mouse, которые реализуют метод receiveInput().

## 6. Интерфейсы

Пример решений задач

Задача 1. Создать интерфейс IIdentifiable, который содержит метод getId(). Создать класс Person, реализующий этот интерфейс. Класс должен содержать поля name, age и id. Реализовать метод для отображения информации о человеке и метод для изменения возраста.

|  |
| --- |
| С++ |
| #include <iostream>  #include <string>  // Интерфейс IIdentifiable  class IIdentifiable {  public:  virtual int getId() const = 0; // Чисто виртуальный метод для получения ID  };  // Класс Person, реализующий интерфейс IIdentifiable  class Person : public IIdentifiable {  private:  std::string name;  int age;  int id;  public:  // Конструктор класса Person  Person(std::string name, int age, int id) : name(name), age(age), id(id) {}  // Реализация метода getId  int getId() const override {  return id;  }  // Метод для отображения информации о человеке  void displayInfo() const {  std::cout << "Name: " << name << ", Age: " << age << ", ID: " << id << std::endl;  }  // Метод для изменения возраста  void changeAge(int newAge) {  age = newAge;  }  };  int main() {  Person person("Alice", 30, 1);  person.displayInfo();  person.changeAge(31);  person.displayInfo();  return 0;  } |
| C# |
| using System;  // Интерфейс IIdentifiable  public interface IIdentifiable  {  int GetId(); // Метод для получения ID  }  // Класс Person, реализующий интерфейс IIdentifiable  public class Person : IIdentifiable  {  private string name;  private int age;  private int id;  // Конструктор класса Person  public Person(string name, int age, int id)  {  this.name = name;  this.age = age;  this.id = id;  }  // Реализация метода GetId  public int GetId()  {  return id;  }  // Метод для отображения информации о человеке  public void DisplayInfo()  {  Console.WriteLine($"Name: {name}, Age: {age}, ID: {id}");  }  // Метод для изменения возраста  public void ChangeAge(int newAge)  {  age = newAge;  }  }  class Program  {  static void Main()  {  Person person = new Person("Alice", 30, 1);  person.DisplayInfo();  person.ChangeAge(31);  person.DisplayInfo();  }  } |
| Java |
| // Интерфейс IIdentifiable  interface IIdentifiable {  int getId(); // Метод для получения ID  }  // Класс Person, реализующий интерфейс IIdentifiable  class Person implements IIdentifiable {  private String name;  private int age;  private int id;  // Конструктор класса Person  public Person(String name, int age, int id) {  this.name = name;  this.age = age;  this.id = id;  }  // Реализация метода getId  @Override  public int getId() {  return id;  }  // Метод для отображения информации о человеке  public void displayInfo() {  System.out.println("Name: " + name + ", Age: " + age + ", ID: " + id);  }  // Метод для изменения возраста  public void changeAge(int newAge) {  age = newAge;  }  public static void main(String[] args) {  Person person = new Person("Alice", 30, 1);  person.displayInfo();  person.changeAge(31);  person.displayInfo();  }  } |
| Python |
| # Интерфейс IIdentifiable  from abc import ABC, abstractmethod  class IIdentifiable(ABC):  @abstractmethod  def get\_id(self):  pass # Метод для получения ID  # Класс Person, реализующий интерфейс IIdentifiable  class Person(IIdentifiable):  def \_\_init\_\_(self, name, age, id):  self.name = name  self.age = age  self.id = id  # Реализация метода get\_id  def get\_id(self):  return self.id  # Метод для отображения информации о человеке  def display\_info(self):  print(f"Name: {self.name}, Age: {self.age}, ID: {self.id}")  # Метод для изменения возраста  def change\_age(self, new\_age):  self.age = new\_age  # Пример использования  person = Person("Alice", 30, 1)  person.display\_info()  person.change\_age(31)  person.display\_info() |

### Задачи для самостоятельной работы

Задача 2. Создать интерфейс IDocument, который содержит методы updateContent(String content) и getInfo(). Создать класс Document, реализующий этот интерфейс. Класс должен содержать поля title, author и content.

Задача 3. Создать интерфейс IProduct, который содержит методы getTotalValue() и updatePrice(double price). Создать класс Product, реализующий этот интерфейс. Класс должен содержать поля name, price и quantity.

Задача 4. Создать интерфейс ICartItem, который содержит методы updateQuantity(int quantity) и getTotalPrice(). Создать класс CartItem, реализующий этот интерфейс. Класс должен содержать поля productName, quantity и price.

Задача 5. Создать интерфейс IStack, который содержит методы push(T item), pop(), peek() и isEmpty(). Создать класс Stack<T>, реализующий этот интерфейс.

Задача 6. Создать интерфейс IQueue, который содержит методы enqueue(T item), dequeue(), peek() и isEmpty(). Создать класс Queue<T>, реализующий этот интерфейс.

Задача 7. Создать интерфейс IShape, который содержит метод getArea(). Создать абстрактный класс Shape и конкретные реализации Circle и Rectangle, которые реализуют интерфейс IShape.

Задача 8. Создать интерфейс IVehicle, который содержит методы start() и stop(). Создать абстрактный класс Vehicle, и конкретные реализации Car и Bicycle, которые реализуют интерфейс IVehicle.

Задача 9. Создать интерфейс IEvent, который содержит методы updateLocation(String location) и getDetails(). Создать класс Event, реализующий этот интерфейс. Класс должен содержать поля title, date и location.

Задача 10. Создать интерфейс ICourse, который содержит методы updateCredits(int credits) и getCourseInfo(). Создать класс Course, реализующий этот интерфейс. Класс должен содержать поля name, code и credits.

Задача 11. Создать интерфейс ILeaveRequest, который содержит методы updateDates(LocalDate start, LocalDate end) и getRequestInfo(). Создать класс LeaveRequest, реализующий этот интерфейс. Класс должен содержать поля employeeName, startDate и endDate.

Задача 12. Создать интерфейс IProfile, который содержит методы updateEmail(String email) и getProfileInfo(). Создать класс Profile, реализующий этот интерфейс. Класс должен содержать поля username, email и birthDate.

Задача 13. Создать интерфейс IContact, который содержит методы updatePhoneNumber(String phoneNumber) и getContactInfo(). Создать класс Contact, реализующий этот интерфейс. Класс должен содержать поля name, phoneNumber и email.

Задача 14. Создать интерфейс ITask, который содержит методы updateDescription(String description) и getTaskDetails(). Создать класс Task, реализующий этот интерфейс. Класс должен содержать поля title, description и dueDate.

Задача 15. Создать интерфейс IComputer, который содержит методы updateModel(String model) и getComputerInfo(). Создать класс Computer, реализующий этот интерфейс. Класс должен содержать поля brand, model и year.

Задача 16. Создать интерфейс IBook, который содержит методы updateAuthor(String author) и getBookInfo(). Создать класс Book, реализующий этот интерфейс. Класс должен содержать поля title, author и isbn.

Задача 17. Создать интерфейс ICurrencyRate, который содержит методы updateExchangeRate(double rate) и getRateInfo(). Создать класс CurrencyRate, реализующий этот интерфейс. Класс должен содержать поля currencyName, currencyCode и exchangeRate.

Задача 18. Создать интерфейс IClient, который содержит методы updateBalance(double balance) и getClientInfo(). Создать класс Client, реализующий этот интерфейс. Класс должен содержать поля name, id и balance.

Задача 19. Создать интерфейс IAccount, который содержит методы deposit(double amount), withdraw(double amount) и getAccountInfo(). Создать класс Account, реализующий этот интерфейс. Класс должен содержать поля accountNumber и balance.

Задача 20. Создать интерфейс ICompany, который содержит методы updateAddress(String address) и getCompanyInfo(). Создать класс Company, реализующий этот интерфейс. Класс должен содержать поля name и address.

Задача 21. Создать интерфейс IEmployee, который содержит методы updatePosition(String position) и getEmployeeInfo(). Создать класс Employee, реализующий этот интерфейс. Класс должен содержать поля name, position и salary.

Задача 22. Создать интерфейс IMovie, который содержит методы updateDirector(String director) и getMovieDetails(). Создать класс Movie, реализующий этот интерфейс. Класс должен содержать поля title, director и releaseYear.

Задача 23. Создать интерфейс IUniversity, который содержит методы addStudent() и getUniversityInfo(). Создать класс University, реализующий этот интерфейс. Класс должен содержать поля name и studentCount.

Задача 24. Создать интерфейс IPayment, который содержит методы updateAmount(double amount) и getPaymentDetails(). Создать класс Payment, реализующий этот интерфейс. Класс должен содержать поля amount и date.

Задача 25. Создать интерфейс IInvoice, который содержит методы updateAmount(double amount) и getInvoiceDetails(). Создать класс Invoice, реализующий этот интерфейс. Класс должен содержать поля invoiceNumber и amount.

Задача 26. Создать интерфейс IRecipe, который содержит методы updatePreparationTime(int time) и getRecipeDetails(). Создать класс Recipe, реализующий этот интерфейс. Класс должен содержать поля title, ingredients и preparationTime.

Задача 27. Создать интерфейс IGadget, который содержит методы updatePrice(double price) и getGadgetInfo(). Создать класс Gadget, реализующий этот интерфейс. Класс должен содержать поля name, type и price.

Задача 28. Создать интерфейс ITransaction, который содержит методы updateAmount(double amount) и getTransactionDetails(). Создать класс Transaction, реализующий этот интерфейс. Класс должен содержать поля transactionType, amount и date.

Задача 29. Создать интерфейс IHouse, который содержит методы updateArea(double area) и getHouseInfo(). Создать класс House, реализующий этот интерфейс. Класс должен содержать поля address, roomCount и area.

Задача 30. Создать интерфейс IProject, который содержит методы updateDescription(String description) и getProjectDetails(). Создать класс Project, реализующий этот интерфейс. Класс должен содержать поля title, description и startDate.

Задача 31. Создать интерфейс IBuilding, который содержит методы updateFloors(int floors) и getBuildingDetails(). Создать класс Building, реализующий этот интерфейс. Класс должен содержать поля address, floors и yearBuilt.

Задача 32. Создать интерфейс ITrainingCourse, который содержит методы updateCost(double cost) и getCourseDetails(). Создать класс TrainingCourse, реализующий этот интерфейс. Класс должен содержать поля title, duration и cost.

Задача 33. Создать интерфейс ISystem, который содержит методы updateVersion(String version) и getSystemInfo(). Создать класс System, реализующий этот интерфейс. Класс должен содержать поля name и version.

Задача 34. Создать интерфейс IRental, который содержит методы updateDates(LocalDate startDate, LocalDate endDate) и getRentalDetails(). Создать класс Rental, реализующий этот интерфейс. Класс должен содержать поля rentalObject, startDate и endDate.

Задача 35. Создать интерфейс IMap, который содержит методы updateScale(double scale) и getMapInfo(). Создать класс Map, реализующий этот интерфейс. Класс должен содержать поля title и scale.

## 7. Полиморфизм

Пример решений задач

Задача 1. Определите метод getDetails() в классе Employee, который будет выводить информацию о работнике. Создайте несколько подклассов, таких как Manager, Developer, и переопределите метод getDetails() для каждого подкласса, чтобы выводить специфическую информацию.

|  |
| --- |
| С++ |
| #include <iostream>  #include <string>  // Базовый класс Employee  class Employee {  protected:  std::string name;  int id;  public:  Employee(std::string name, int id) : name(name), id(id) {}  virtual void getDetails() {  std::cout << "Employee Name: " << name << ", ID: " << id << std::endl;  }  };  // Подкласс Manager  class Manager : public Employee {  public:  Manager(std::string name, int id) : Employee(name, id) {}  void getDetails() override {  std::cout << "Manager Name: " << name << ", ID: " << id << " - Responsible for overseeing teams." << std::endl;  }  };  // Подкласс Developer  class Developer : public Employee {  public:  Developer(std::string name, int id) : Employee(name, id) {}  void getDetails() override {  std::cout << "Developer Name: " << name << ", ID: " << id << " - Responsible for writing code." << std::endl;  }  };  int main() {  Employee\* emp1 = new Manager("Alice", 101);  Employee\* emp2 = new Developer("Bob", 102);  emp1->getDetails(); // Выводит информацию о менеджере  emp2->getDetails(); // Выводит информацию о разработчике  delete emp1;  delete emp2;  return 0;  } |
| C# |
| using System;  // Базовый класс Employee  public class Employee  {  protected string Name;  protected int Id;  public Employee(string name, int id)  {  Name = name;  Id = id;  }  public virtual void GetDetails()  {  Console.WriteLine($"Employee Name: {Name}, ID: {Id}");  }  }  // Подкласс Manager  public class Manager : Employee  {  public Manager(string name, int id) : base(name, id) {}  public override void GetDetails()  {  Console.WriteLine($"Manager Name: {Name}, ID: {Id} - Responsible for overseeing teams.");  }  }  // Подкласс Developer  public class Developer : Employee  {  public Developer(string name, int id) : base(name, id) {}  public override void GetDetails()  {  Console.WriteLine($"Developer Name: {Name}, ID: {Id} - Responsible for writing code.");  }  }  class Program  {  static void Main()  {  Employee emp1 = new Manager("Alice", 101);  Employee emp2 = new Developer("Bob", 102);  emp1.GetDetails(); // Выводит информацию о менеджере  emp2.GetDetails(); // Выводит информацию о разработчике  }  } |
| Java |
| // Базовый класс Employee  class Employee {  protected String name;  protected int id;  public Employee(String name, int id) {  this.name = name;  this.id = id;  }  public void getDetails() {  System.out.println("Employee Name: " + name + ", ID: " + id);  }  }  // Подкласс Manager  class Manager extends Employee {  public Manager(String name, int id) {  super(name, id);  }  @Override  public void getDetails() {  System.out.println("Manager Name: " + name + ", ID: " + id + " - Responsible for overseeing teams.");  }  }  // Подкласс Developer  class Developer extends Employee {  public Developer(String name, int id) {  super(name, id);  }  @Override  public void getDetails() {  System.out.println("Developer Name: " + name + ", ID: " + id + " - Responsible for writing code.");  }  }  public class Main {  public static void main(String[] args) {  Employee emp1 = new Manager("Alice", 101);  Employee emp2 = new Developer("Bob", 102);  emp1.getDetails(); // Выводит информацию о менеджере  emp2.getDetails(); // Выводит информацию о разработчике  }  } |
| Python |
| # Базовый класс Employee  class Employee:  def \_\_init\_\_(self, name, id):  self.name = name  self.id = id  def get\_details(self):  print(f"Employee Name: {self.name}, ID: {self.id}")  # Подкласс Manager  class Manager(Employee):  def get\_details(self):  print(f"Manager Name: {self.name}, ID: {self.id} - Responsible for overseeing teams.")  # Подкласс Developer  class Developer(Employee):  def get\_details(self):  print(f"Developer Name: {self.name}, ID: {self.id} - Responsible for writing code.")  # Пример использования  if \_\_name\_\_ == "\_\_main\_\_":  emp1 = Manager("Alice", 101)  emp2 = Developer("Bob", 102)  emp1.get\_details() # Выводит информацию о менеджере  emp2.get\_details() # Выводит информацию о разработчике |

Задача 1.1. Определите интерфейс EmployeeInterface, который будет содержать метод getDetails(), выводящий информацию о работнике. Реализуйте несколько классов, таких как Manager и Developer, которые будут реализовывать этот интерфейс. В каждом классе переопределите метод getDetails() для вывода специфической информации о работнике. Также создайте функцию, которая принимает объект типа EmployeeInterface и выводит его детали, используя механизм обратного вызова.

|  |
| --- |
| С++ |
| #include <iostream>  #include <string>  // Определяем интерфейс  class EmployeeInterface {  public:  virtual std::string getDetails() const = 0; // Чисто виртуальный метод  virtual ~EmployeeInterface() {} // Виртуальный деструктор  };  // Класс Manager  class Manager : public EmployeeInterface {  private:  std::string name;  int teamSize;  public:  Manager(const std::string& name, int teamSize)  : name(name), teamSize(teamSize) {}  std::string getDetails() const override {  return "Name: " + name + ", Position: Manager, Team Size: " + std::to\_string(teamSize);  }  };  // Класс Developer  class Developer : public EmployeeInterface {  private:  std::string name;  std::string programmingLanguage;  public:  Developer(const std::string& name, const std::string& programmingLanguage)  : name(name), programmingLanguage(programmingLanguage) {}  std::string getDetails() const override {  return "Name: " + name + ", Position: Developer, Programming Language: " + programmingLanguage;  }  };  // Функция, принимающая объект EmployeeInterface  void printEmployeeDetails(const EmployeeInterface& employee) {  std::cout << employee.getDetails() << std::endl;  }  // Пример использования  int main() {  Manager manager("Alice", 5);  Developer developer("Bob", "C++");  printEmployeeDetails(manager);  printEmployeeDetails(developer);  return 0;  } |
| C# |
| using System;  namespace EmployeeExample  {  // Определяем интерфейс  public interface IEmployee  {  string GetDetails();  }  // Класс Manager  public class Manager : IEmployee  {  private string name;  private int teamSize;  public Manager(string name, int teamSize)  {  this.name = name;  this.teamSize = teamSize;  }  public string GetDetails()  {  return $"Name: {name}, Position: Manager, Team Size: {teamSize}";  }  }  // Класс Developer  public class Developer : IEmployee  {  private string name;  private string programmingLanguage;  public Developer(string name, string programmingLanguage)  {  this.name = name;  this.programmingLanguage = programmingLanguage;  }  public string GetDetails()  {  return $"Name: {name}, Position: Developer, Programming Language: {programmingLanguage}";  }  }  // Функция, принимающая объект IEmployee  public static void PrintEmployeeDetails(IEmployee employee)  {  Console.WriteLine(employee.GetDetails());  }  // Пример использования  class Program  {  static void Main(string[] args)  {  IEmployee manager = new Manager("Alice", 5);  IEmployee developer = new Developer("Bob", "C#");  PrintEmployeeDetails(manager);  PrintEmployeeDetails(developer);  }  }  } |
| Java |
| // Определяем интерфейс  interface Employee {  String getDetails();  }  // Класс Manager  class Manager implements Employee {  private String name;  private int teamSize;  public Manager(String name, int teamSize) {  this.name = name;  this.teamSize = teamSize;  }  @Override  public String getDetails() {  return "Name: " + name + ", Position: Manager, Team Size: " + teamSize;  }  }  // Класс Developer  class Developer implements Employee {  private String name;  private String programmingLanguage;  public Developer(String name, String programmingLanguage) {  this.name = name;  this.programmingLanguage = programmingLanguage;  }  @Override  public String getDetails() {  return "Name: " + name + ", Position: Developer, Programming Language: " + programmingLanguage;  }  }  // Функция, принимающая объект Employee  public class Main {  public static void printEmployeeDetails(Employee employee) {  System.out.println(employee.getDetails());  }  // Пример использования  public static void main(String[] args) {  Employee manager = new Manager("Alice", 5);  Employee developer = new Developer("Bob", "Java");  printEmployeeDetails(manager);  printEmployeeDetails(developer);  }  } |
| Python |
| from abc import ABC, abstractmethod  # Определяем интерфейс  class EmployeeInterface(ABC):  @abstractmethod  def get\_details(self):  pass  # Класс Manager  class Manager(EmployeeInterface):  def \_\_init\_\_(self, name, team\_size):  self.name = name  self.team\_size = team\_size  def get\_details(self):  return f'Name: {self.name}, Position: Manager, Team Size: {self.team\_size}'  # Класс Developer  class Developer(EmployeeInterface):  def \_\_init\_\_(self, name, programming\_language):  self.name = name  self.programming\_language = programming\_language  def get\_details(self):  return f'Name: {self.name}, Position: Developer, Programming Language: {self.programming\_language}'  # Функция, принимающая объект EmployeeInterface  def print\_employee\_details(employee: EmployeeInterface):  print(employee.get\_details())  # Пример использования  if \_\_name\_\_ == "\_\_main\_\_":  manager = Manager("Alice", 5)  developer = Developer("Bob", "Python")  print\_employee\_details(manager)  print\_employee\_details(developer) |

### Задачи для самостоятельной работы

Задача 2. Определите метод getDescription() в классе Recipe, который будет возвращать описание рецепта. Создайте подклассы, такие как VeganRecipe, DessertRecipe, и переопределите метод getDescription() для каждого подкласса, чтобы выводить различные типы описаний.

Задача 3. В классе CartItem реализуйте метод calculateTotalPrice(), который вычисляет общую стоимость товара в корзине. Создайте подклассы DiscountedItem и PremiumItem, которые будут переопределять этот метод для применения скидок или надбавок.

Задача 4. Определите метод getSpecification() в классе Gadget, который возвращает спецификации устройства. Создайте подклассы, такие как Smartphone и Laptop, и переопределите метод getSpecification() для каждого из них.

Задача 5. Определите метод addElement() в классе Queue, который добавляет элемент в очередь. Создайте подклассы PriorityQueue и CircularQueue, которые будут переопределять этот метод для различных стратегий добавления элементов.

Задача 6. Реализуйте метод getShapeType() в классе Circle, который возвращает тип фигуры. Создайте несколько подклассов, таких как FilledCircle и OutlinedCircle, которые будут переопределять этот метод для возвращения различной информации.

Задача 7. Определите метод draw() в классе Rectangle, который рисует прямоугольник. Создайте подклассы ColoredRectangle и DashedRectangle, которые будут переопределять этот метод для различной визуализации прямоугольников.

Задача 8. Определите метод drive() в классе Car, который управляет движением автомобиля. Создайте подклассы ElectricCar и GasolineCar, которые будут переопределять этот метод для управления различными типами автомобилей.

Задача 9. Определите метод ride() в классе Bicycle, который реализует процесс езды на велосипеде. Создайте подклассы MountainBike и RoadBike, которые будут переопределять этот метод для различных типов езды.

Задача 10. В классе Course определите метод calculateFinalGrade(), который вычисляет итоговую оценку. Создайте подклассы OnlineCourse и OfflineCourse, которые будут переопределять этот метод для разных типов курсов.

Задача 11. Определите метод approve() в классе LeaveRequest, который будет утверждать заявку на отпуск. Создайте подклассы PaidLeaveRequest и UnpaidLeaveRequest, которые будут переопределять этот метод в зависимости от типа отпуска.

Задача 12. Определите метод makePurchase() в классе Client, который будет осуществлять покупку. Создайте подклассы VIPClient и RegularClient, которые будут переопределять этот метод для различных условий покупки.

Задача 13. Определите метод sendNotification() в классе Contact, который будет отправлять уведомление контакту. Создайте подклассы EmailContact и PhoneContact, которые будут переопределять этот метод для отправки разных типов уведомлений.

Задача 14. В классе Project определите метод calculateBudget(), который вычисляет бюджет проекта. Создайте подклассы SmallProject и LargeProject, которые будут переопределять этот метод в зависимости от масштаба проекта.

Задача 15. Определите метод bootUp() в классе Computer, который будет запускать компьютер. Создайте подклассы WindowsComputer и MacComputer, которые будут переопределять этот метод для запуска различных операционных систем.

Задача 16. Определите метод getGenre() в классе Book, который будет возвращать жанр книги. Создайте подклассы FictionBook и NonFictionBook, которые будут переопределять этот метод для различных жанров.

Задача 17. Определите метод convertCurrency() в классе CurrencyRate, который будет выполнять конвертацию валюты. Создайте подклассы CryptoCurrencyRate и FiatCurrencyRate, которые будут переопределять этот метод для различных типов валют.

Задача 18. Определите метод transferFunds() в классе Account, который будет переводить средства между счетами. Создайте подклассы SavingsAccount и CheckingAccount, которые будут переопределять этот метод для выполнения специфических типов переводов.

Задача 19. Определите метод generateReport() в классе Company, который будет генерировать отчет о деятельности компании. Создайте подклассы TechCompany и ManufacturingCompany, которые будут переопределять этот метод для создания различных типов отчетов.

Задача 20. Определите метод calculateBonus() в классе Employee, который будет вычислять бонус для работника. Создайте подклассы FullTimeEmployee и PartTimeEmployee, которые будут переопределять этот метод в зависимости от типа занятости.

Задача 21. Определите метод play() в классе Movie, который будет запускать фильм. Создайте подклассы ActionMovie и DocumentaryMovie, которые будут переопределять этот метод для различных типов фильмов.

Задача 22. Определите метод addStudent() в классе University, который будет добавлять студента в университет. Создайте подклассы PublicUniversity и PrivateUniversity, которые будут переопределять этот метод для добавления студентов с разными условиями.

Задача 23. Определите метод processPayment() в классе Payment, который будет обрабатывать платеж. Создайте подклассы CreditCardPayment и PayPalPayment, которые будут переопределять этот метод для различных способов оплаты.

Задача 24. Определите метод sendInvoice() в классе Invoice, который будет отправлять счет на оплату. Создайте подклассы EmailInvoice и PaperInvoice, которые будут переопределять этот метод для различных способов отправки счетов.

Задача 25. Определите метод prepare() в классе Recipe, который будет описывать процесс приготовления рецепта. Создайте подклассы SimpleRecipe и ComplexRecipe, которые будут переопределять этот метод для различной сложности приготовления.

Задача 26. Определите метод calculateWarranty() в классе Gadget, который будет вычислять срок гарантии. Создайте подклассы Smartphone и Tablet, которые будут переопределять этот метод для различных типов гаджетов.

Задача 27. Определите метод execute() в классе Transaction, который будет выполнять финансовую операцию. Создайте подклассы Deposit и Withdrawal, которые будут переопределять этот метод для выполнения специфических операций.

Задача 28. Определите метод calculateValue() в классе House, который будет вычислять стоимость дома. Создайте подклассы LuxuryHouse и EconomyHouse, которые будут переопределять этот метод для различных типов домов.

Задача 29. Определите метод evaluateRisk() в классе Project, который будет оценивать риски проекта. Создайте подклассы HighRiskProject и LowRiskProject, которые будут переопределять этот метод для различных типов проектов.

Задача 30. Определите метод calculateMaintenanceCost() в классе Building, который будет вычислять затраты на обслуживание здания. Создайте подклассы Skyscraper и Bungalow, которые будут переопределять этот метод для различных типов зданий.

Задача 31. Определите метод enroll() в классе TrainingCourse, который будет записывать студента на курс. Создайте подклассы OnlineTrainingCourse и InPersonTrainingCourse, которые будут переопределять этот метод для различных форматов курсов.

Задача 32. Определите метод update() в классе System, который будет обновлять систему. Создайте подклассы OperatingSystem и ApplicationSystem, которые будут переопределять этот метод для различных типов систем.

Задача 33. Определите метод calculateRent() в классе Rental, который будет вычислять стоимость аренды. Создайте подклассы ShortTermRental и LongTermRental, которые будут переопределять этот метод для различных типов аренды.

Задача 34. Определите метод display() в классе Map, который будет отображать карту. Создайте подклассы PhysicalMap и DigitalMap, которые будут переопределять этот метод для различных типов карт.

Задача 35. Определите метод allocateFunds() в классе FinancialProject, который будет распределять средства по проекту. Создайте подклассы StartupProject и CorporateProject, которые будут переопределять этот метод для различных типов проектов.

# ГЛАВА 2. ПРИНЦИПЫ ООП

## 1. Dependency Injection

Пример решений задач

Задача 1. Определите метод getDetails() в классе Employee, который будет выводить информацию о работнике. Создайте несколько подклассов, таких как Manager, Developer, и переопределите метод getDetails() для каждого подкласса, чтобы выводить специфическую информацию.

|  |
| --- |
| С++ |
| Вариант 1 |
| #include <iostream>  #include <memory>  class IEmployee {  public:  virtual void getDetails() = 0;  };  class Manager : public IEmployee {  public:  void getDetails() override {  std::cout << "I am a manager." << std::endl;  }  };  class Developer : public IEmployee {  public:  void getDetails() override {  std::cout << "I am a developer." << std::endl;  }  };  class EmployeeService {  private:  IEmployee\* employee; // Указатель на IEmployee  public:  EmployeeService(IEmployee\* emp) : employee(emp) {}  void showDetails() {  employee->getDetails();  }  };  int main() {  Manager mgr;  Developer dev;  EmployeeService service1(&mgr);  EmployeeService service2(&dev);  service1.showDetails(); // Вывод: I am a manager.  service2.showDetails(); // Вывод: I am a developer.  return 0;  } |
| Вариант 2 |
| #include <iostream>  #include <memory>  #include <boost/di.hpp>  class IEmployee {  public:  virtual void getDetails() = 0;  };  class Manager : public IEmployee {  public:  void getDetails() override {  std::cout << "I am a manager." << std::endl;  }  };  class EmployeeService {  public:  EmployeeService(std::shared\_ptr<IEmployee> employee) : employee\_(employee) {}  void showDetails() {  employee\_->getDetails();  }  private:  std::shared\_ptr<IEmployee> employee\_;  };  namespace di = boost::di;  int main() {  auto injector = di::make\_injector(  di::bind<IEmployee>().to<Manager>()  );  auto service = injector.create<std::shared\_ptr<EmployeeService>>();  service->showDetails(); // Вывод: I am a manager.  return 0;  } |
| C# |
| Вариант 1 |
| using System;  interface IEmployee {  void GetDetails();  }  class Manager : IEmployee {  public void GetDetails() {  Console.WriteLine("I am a manager.");  }  }  class Developer : IEmployee {  public void GetDetails() {  Console.WriteLine("I am a developer.");  }  }  class EmployeeService {  private IEmployee \_employee;  public EmployeeService(IEmployee employee) {  \_employee = employee;  }  public void ShowDetails() {  \_employee.GetDetails();  }  }  class Program {  static void Main() {  IEmployee mgr = new Manager();  IEmployee dev = new Developer();  EmployeeService service1 = new EmployeeService(mgr);  EmployeeService service2 = new EmployeeService(dev);  service1.ShowDetails(); // Вывод: I am a manager.  service2.ShowDetails(); // Вывод: I am a developer.  }  } |
| Вариант 2 |
| using System;  using Microsoft.Extensions.DependencyInjection;  interface IEmployee {  void GetDetails();  }  class Manager : IEmployee {  public void GetDetails() {  Console.WriteLine("I am a manager.");  }  }  class EmployeeService {  private readonly IEmployee \_employee;  public EmployeeService(IEmployee employee) {  \_employee = employee;  }  public void ShowDetails() {  \_employee.GetDetails();  }  }  class Program {  static void Main() {  var serviceProvider = new ServiceCollection()  .AddSingleton<IEmployee, Manager>()  .AddTransient<EmployeeService>()  .BuildServiceProvider();  var service = serviceProvider.GetService<EmployeeService>();  service.ShowDetails(); // Вывод: I am a manager.  }  } |
| Java |
| Вариант 1 |
| interface Employee {  void getDetails();  }  class Manager implements Employee {  public void getDetails() {  System.out.println("I am a manager.");  }  }  class Developer implements Employee {  public void getDetails() {  System.out.println("I am a developer.");  }  }  class EmployeeService {  private Employee employee;  public EmployeeService(Employee employee) {  this.employee = employee;  }  public void showDetails() {  employee.getDetails();  }  }  public class Main {  public static void main(String[] args) {  Employee mgr = new Manager();  Employee dev = new Developer();  EmployeeService service1 = new EmployeeService(mgr);  EmployeeService service2 = new EmployeeService(dev);  service1.showDetails(); // Вывод: I am a manager.  service2.showDetails(); // Вывод: I am a developer.  }  } |
| Вариант 2 |
| import org.springframework.context.annotation.Bean;  import org.springframework.context.annotation.ComponentScan;  import org.springframework.context.annotation.Configuration;  import org.springframework.context.annotation.AnnotationConfigApplicationContext;  interface Employee {  void getDetails();  }  class Manager implements Employee {  public void getDetails() {  System.out.println("I am a manager.");  }  }  class EmployeeService {  private final Employee employee;  public EmployeeService(Employee employee) {  this.employee = employee;  }  public void showDetails() {  employee.getDetails();  }  }  @Configuration  @ComponentScan  class AppConfig {  @Bean  public Employee employee() {  return new Manager();  }  @Bean  public EmployeeService employeeService() {  return new EmployeeService(employee());  }  }  public class Main {  public static void main(String[] args) {  AnnotationConfigApplicationContext context = new AnnotationConfigApplicationContext(AppConfig.class);  EmployeeService service = context.getBean(EmployeeService.class);  service.showDetails(); // Вывод: I am a manager.  context.close();  }  } |
| Python |
| Вариант 1 |
| class IEmployee:  def get\_details(self):  pass  class Manager(IEmployee):  def get\_details(self):  print("I am a manager.")  class Developer(IEmployee):  def get\_details(self):  print("I am a developer.")  class EmployeeService:  def \_\_init\_\_(self, employee: IEmployee):  self.employee = employee  def show\_details(self):  self.employee.get\_details()  if \_\_name\_\_ == "\_\_main\_\_":  mgr = Manager()  dev = Developer()  service1 = EmployeeService(mgr)  service2 = EmployeeService(dev)  service1.show\_details() # Вывод: I am a manager.  service2.show\_details() # Вывод: I am a developer. |
| Вариант 2 |
| !pip install dependency\_injector  from dependency\_injector import containers, providers  class IEmployee:  def get\_details(self):  pass  class Manager(IEmployee):  def get\_details(self):  print("I am a manager.")  class EmployeeService:  def \_\_init\_\_(self, employee: IEmployee):  self.employee = employee  def show\_details(self):  self.employee.get\_details()  class Container(containers.DeclarativeContainer):  employee = providers.Singleton(Manager) # Регистрация зависимости  employee\_service = providers.Factory(EmployeeService, employee=employee)  if \_\_name\_\_ == "\_\_main\_\_":  service = Container.employee\_service()  service.show\_details() # Вывод: I am a manager. |

### Задачи для самостоятельной работы

Задача 2. Определите метод getDescription() в классе Recipe, который будет возвращать описание рецепта. Создайте подклассы, такие как VeganRecipe, DessertRecipe, и переопределите метод getDescription() для каждого подкласса, чтобы выводить различные типы описаний.

Задача 3. В классе CartItem реализуйте метод calculateTotalPrice(), который вычисляет общую стоимость товара в корзине. Создайте подклассы DiscountedItem и PremiumItem, которые будут переопределять этот метод для применения скидок или надбавок.

Задача 4. Определите метод getSpecification() в классе Gadget, который возвращает спецификации устройства. Создайте подклассы, такие как Smartphone и Laptop, и переопределите метод getSpecification() для каждого из них.

Задача 5. Определите метод addElement() в классе Queue, который добавляет элемент в очередь. Создайте подклассы PriorityQueue и CircularQueue, которые будут переопределять этот метод для различных стратегий добавления элементов.

Задача 6. Реализуйте метод getShapeType() в классе Circle, который возвращает тип фигуры. Создайте несколько подклассов, таких как FilledCircle и OutlinedCircle, которые будут переопределять этот метод для возвращения различной информации.

Задача 7. Определите метод draw() в классе Rectangle, который рисует прямоугольник. Создайте подклассы ColoredRectangle и DashedRectangle, которые будут переопределять этот метод для различной визуализации прямоугольников.

Задача 8. Определите метод drive() в классе Car, который управляет движением автомобиля. Создайте подклассы ElectricCar и GasCar, которые будут переопределять этот метод для различных типов автомобилей.

Задача 9. Определите класс Event, который использует интерфейс INotifier для отправки уведомлений о событии. Реализуйте конструктор Event с внедрением зависимости INotifier и методы для отправки уведомлений.

Задача 10. Создайте класс Course, который зависит от интерфейса IContentProvider для предоставления информации о курсе. Реализуйте конструктор Course с внедрением зависимости IContentProvider и методы для получения информации о курсе через этот провайдер.

Задача 11. Определите класс LeaveRequest, который использует интерфейс IApprovalService для обработки заявок на отпуск. Реализуйте конструктор LeaveRequest с внедрением зависимости IApprovalService и методы для отправки заявки на одобрение.

Задача 12. Создайте класс Profile, который зависит от интерфейса IProfileStorage для сохранения и загрузки профиля пользователя. Реализуйте конструктор Profile с внедрением зависимости IProfileStorage и методы для обновления и получения профиля.

Задача 13. Определите класс Contact, который использует интерфейс IContactManager для управления контактами. Реализуйте конструктор Contact с внедрением зависимости IContactManager и методы для добавления и удаления контактов.

Задача 14. Создайте класс Task, который зависит от интерфейса ITaskScheduler для планирования задач. Реализуйте конструктор Task с внедрением зависимости ITaskScheduler и методы для планирования и выполнения задач.

Задача 15. Определите класс Computer, который использует интерфейс IHardwareProvider для получения информации о компонентах. Реализуйте конструктор Computer с внедрением зависимости IHardwareProvider и методы для отображения информации о компонентах.

Задача 16. Создайте класс Book, который зависит от интерфейса IReviewService для получения и отображения рецензий. Реализуйте конструктор Book с внедрением зависимости IReviewService и методы для добавления и получения рецензий.

Задача 17. Определите класс CurrencyRate, который использует интерфейс IRateProvider для получения актуальных обменных курсов. Реализуйте конструктор CurrencyRate с внедрением зависимости IRateProvider и методы для получения и обновления обменного курса.

Задача 18. Создайте класс Client, который зависит от интерфейса IClientService для управления клиентской информацией. Реализуйте конструктор Client с внедрением зависимости IClientService и методы для обновления и получения информации о клиенте.

Задача 19. Определите класс Account, который использует интерфейс ITransactionProcessor для обработки транзакций. Реализуйте конструктор Account с внедрением зависимости ITransactionProcessor и методы для пополнения и списания средств.

Задача 20. Создайте класс Company, который зависит от интерфейса IEmployeeManager для управления сотрудниками. Реализуйте конструктор Company с внедрением зависимости IEmployeeManager и методы для добавления и удаления сотрудников.

Задача 21. Определите класс Employee, который использует интерфейс ITrainingProvider для получения информации о тренингах. Реализуйте конструктор Employee с внедрением зависимости ITrainingProvider и методы для записи на тренинги.

Задача 22. Создайте класс Movie, который зависит от интерфейса IReviewProvider для получения рецензий на фильмы. Реализуйте конструктор Movie с внедрением зависимости IReviewProvider и методы для отображения рецензий.

Задача 23. Определите класс University, который использует интерфейс ICourseManager для управления курсами. Реализуйте конструктор University с внедрением зависимости ICourseManager и методы для добавления и удаления курсов.

Задача 24. Создайте класс Payment, который зависит от интерфейса IPaymentProcessor для обработки платежей. Реализуйте конструктор Payment с внедрением зависимости IPaymentProcessor и методы для выполнения и отмены платежей.

Задача 25. Определите класс Invoice, который использует интерфейс ITaxCalculator для расчета налога на выставленный счёт. Реализуйте конструктор Invoice с внедрением зависимости ITaxCalculator и методы для расчета и отображения суммы с налогом.

Задача 26. Создайте класс Recipe, который зависит от интерфейса INutritionalInfoProvider для получения информации о питательных веществах. Реализуйте конструктор Recipe с внедрением зависимости INutritionalInfoProvider и методы для получения и отображения информации о составе.

Задача 27. Определите класс Gadget, который использует интерфейс IWarrantyProvider для проверки гарантии. Реализуйте конструктор Gadget с внедрением зависимости IWarrantyProvider и методы для проверки и продления гарантии.

Задача 28. Создайте класс Transaction, который зависит от интерфейса IFraudDetector для проверки на мошенничество. Реализуйте конструктор Transaction с внедрением зависимости IFraudDetector и методы для проверки и обработки транзакций.

Задача 29. Определите класс House, который использует интерфейс IEnergyCalculator для расчета энергопотребления. Реализуйте конструктор House с внедрением зависимости IEnergyCalculator и методы для расчета и отображения энергопотребления.

Задача 30. Создайте класс Project, который зависит от интерфейса IResourceManager для управления ресурсами проекта. Реализуйте конструктор Project с внедрением зависимости IResourceManager и методы для добавления и удаления ресурсов.

Задача 31. Определите класс Building, который использует интерфейс IMaintenanceService для управления техническим обслуживанием. Реализуйте конструктор Building с внедрением зависимости IMaintenanceService и методы для планирования и выполнения технического обслуживания.

Задача 32. Создайте класс TrainingCourse, который зависит от интерфейса ICourseMaterialProvider для предоставления учебных материалов. Реализуйте конструктор TrainingCourse с внедрением зависимости ICourseMaterialProvider и методы для получения и отображения материалов курса.

Задача 33. Определите класс System, который использует интерфейс IUpdateManager для управления обновлениями системы. Реализуйте конструктор System с внедрением зависимости IUpdateManager и методы для проверки наличия обновлений и их установки.

Задача 34. Создайте класс Rental, который зависит от интерфейса IReservationService для управления бронированиями аренды. Реализуйте конструктор Rental с внедрением зависимости IReservationService и методы для создания и отмены бронирований.

Задача 35. Определите класс Map, который использует интерфейс IMapRenderer для отображения карты. Реализуйте конструктор Map с внедрением зависимости IMapRenderer и методы для отрисовки и обновления карты.

## 2. Маппинг данных

Пример решений задач

Задача 1. Создать два класса UserDTO и UserEntity. Реализовать функцию для маппинга данных из объекта UserDTO в объект UserEntity.

|  |
| --- |
| С++ |
| #include <iostream>  #include <string>  // Класс UserDTO  class UserDTO {  public:  std::string name;  int age;  UserDTO(std::string n, int a) : name(n), age(a) {}  };  // Класс UserEntity  class UserEntity {  public:  std::string fullName;  int years;  UserEntity(std::string fn, int y) : fullName(fn), years(y) {}  };  // Функция для маппинга данных  UserEntity mapUserDTOToEntity(const UserDTO& userDTO) {  return UserEntity(userDTO.name, userDTO.age);  }  int main() {  UserDTO userDTO("Alice", 30);  UserEntity userEntity = mapUserDTOToEntity(userDTO);  std::cout << "Mapped UserEntity: " << userEntity.fullName << ", " << userEntity.years << std::endl;  return 0;  } |
| C# |
| using System;  // Класс UserDTO  public class UserDTO {  public string Name { get; set; }  public int Age { get; set; }  public UserDTO(string name, int age) {  Name = name;  Age = age;  }  }  // Класс UserEntity  public class UserEntity {  public string FullName { get; set; }  public int Years { get; set; }  public UserEntity(string fullName, int years) {  FullName = fullName;  Years = years;  }  }  // Функция для маппинга данных  public static UserEntity MapUserDTOToEntity(UserDTO userDTO) {  return new UserEntity(userDTO.Name, userDTO.Age);  }  public class Program {  public static void Main() {  UserDTO userDTO = new UserDTO("Alice", 30);  UserEntity userEntity = MapUserDTOToEntity(userDTO);  Console.WriteLine($"Mapped UserEntity: {userEntity.FullName}, {userEntity.Years}");  }  } |
| Java |
| // Класс UserDTO  class UserDTO {  String name;  int age;  UserDTO(String name, int age) {  this.name = name;  this.age = age;  }  }  // Класс UserEntity  class UserEntity {  String fullName;  int years;  UserEntity(String fullName, int years) {  this.fullName = fullName;  this.years = years;  }  }  // Функция для маппинга данных  public class Mapper {  public static UserEntity mapUserDTOToEntity(UserDTO userDTO) {  return new UserEntity(userDTO.name, userDTO.age);  }  public static void main(String[] args) {  UserDTO userDTO = new UserDTO("Alice", 30);  UserEntity userEntity = mapUserDTOToEntity(userDTO);  System.out.println("Mapped UserEntity: " + userEntity.fullName + ", " + userEntity.years);  }  } |
| Python |
| # Класс UserDTO  class UserDTO:  def \_\_init\_\_(self, name, age):  self.name = name  self.age = age  # Класс UserEntity  class UserEntity:  def \_\_init\_\_(self, full\_name, years):  self.full\_name = full\_name  self.years = years  # Функция для маппинга данных  def map\_user\_dto\_to\_entity(user\_dto):  return UserEntity(user\_dto.name, user\_dto.age)  # Пример использования  user\_dto = UserDTO("Alice", 30)  user\_entity = map\_user\_dto\_to\_entity(user\_dto)  print(f"Mapped UserEntity: {user\_entity.full\_name}, {user\_entity.years}") |

### Задачи для самостоятельной работы

Задача 2. Определить класс Product с полями name, price и quantity. Реализовать функцию, которая маппирует JSON-данные в объект класса Product.

Задача 3. Создать класс Order с полями id, customerName и totalAmount. Реализовать функцию для маппинга данных из XML-формата в объект класса Order.

Задача 4. Определить два класса EmployeeDTO и EmployeeEntity, где поля имеют разные имена (например, fullName и name). Реализовать маппинг между этими полями.

Задача 5. Создать два класса CompanyDTO и CompanyEntity, в которых одно из полей — это объект другого класса (например, Address). Реализовать маппинг между этими классами с учетом вложенной структуры.

Задача 6. Определить два класса OrderDTO и OrderEntity, где одно из полей — это список товаров. Реализовать маппинг между этими классами с учетом коллекций.

Задача 7. Создать два класса PaymentDTO и PaymentEntity, в которых одно из полей — это enum статус оплаты. Реализовать маппинг между этими классами, учитывая преобразование enum.

Задача 8. Определить два класса UserForm и UserEntity, где в одном классе поле имеет тип String, а в другом — Date. Реализовать маппинг между этими классами с преобразованием типов данных.

Задача 9. Создать два класса InvoiceDTO и InvoiceEntity, в которых одно из полей — это список объектов другого класса. Реализовать маппинг между этими списками, учитывая преобразование типов объектов.

Задача 10. Определить два класса ProductDTO и ProductEntity. Реализовать маппинг между этими классами так, чтобы поле discountedPrice заполнялось только если discount больше 0.

Задача 11. Используя библиотеку маппинга (например, MapStruct или AutoMapper), создать маппинг между двумя классами UserProfileDTO и UserProfileEntity.

Задача 12. Создать класс Customer, представляющий клиента. Реализовать маппинг данных из результата SQL-запроса в объект этого класса.

Задача 13. Определить класс WeatherInfo и реализовать маппинг данных из ответа API о погоде в объект этого класса.

Задача 14. Создать два класса DynamicForm и FormEntity, где один из классов имеет набор полей, определяемых в рантайме. Реализовать маппинг между этими классами.

Задача 15. Определить классы FlatOrderDTO и NestedOrderEntity, где в одном классе данные представлены в плоской структуре, а в другом — во вложенной. Реализовать маппинг с преобразованием структуры.

Задача 16. Создать класс Invoice, представляющий счет. Реализовать маппинг данных из объекта этого класса в JSON.

Задача 17. Определить класс Transaction с полями id, amount и date. Реализовать маппинг данных из CSV-файла в объекты этого класса.

Задача 18. Создать два класса ProductForm и ProductEntity. Реализовать кастомный маппер для маппинга данных между этими классами.

Задача 19. Определить два класса AccountDTO и AccountEntity. Реализовать маппинг между ними так, чтобы в случае ошибки при маппинге выбрасывалось исключение с подробным сообщением.

Задача 20. Создать класс PersonEntity с конструктором, принимающим все поля класса. Реализовать маппинг данных из другого класса PersonDTO с использованием этого конструктора.

Задача 21. Определить два класса BuilderProductDTO и BuilderProductEntity, где один из классов использует паттерн "строитель". Реализовать маппинг данных между этими классами.

Задача 22. Создать базовый класс Person и подклассы Student и Teacher. Реализовать маппинг между этими классами и их DTO.

Задача 23. Создать класс Company с вложенным объектом Address. Реализовать маппинг данных из JSON в объект класса Company, учитывая вложенные структуры.

Задача 24. Определить класс ConsolidatedData и реализовать маппинг данных в этот класс из нескольких источников (например, из нескольких JSON-файлов).

Задача 25. Создать два класса OrderDTO и OrderEntity. Реализовать маппинг с фильтрацией полей на основе определенного условия (например, только активные заказы).

Задача 26. Определить два класса SourceA и SourceB, а также класс Target. Реализовать маппинг, который объединяет данные из SourceA и SourceB в Target.

Задача 27. Создать класс Invoice, представляющий счет. Реализовать маппинг данных из объекта этого класса в SQL-запрос для вставки данных в базу.

Задача 28. Определить класс Product и реализовать маппинг данных из JSON в объект этого класса с использованием внешнего API для получения дополнительных данных.

Задача 29. Создать два класса ClientDTO и ClientEntity. Реализовать маппинг с использованием кастомных правил преобразования для определенных полей.

Задача 30. Определить два класса TransactionDTO и TransactionEntity. Реализовать маппинг между ними с логированием каждого шага маппинга.

Задача 31. Создать класс VersionedDocument, который поддерживает несколько версий. Реализовать маппинг данных в зависимости от версии объекта.

Задача 32. Определить два класса AsyncDataDTO и AsyncDataEntity. Реализовать асинхронный маппинг данных между этими классами.

Задача 33. Создать два класса LogEntryText и LogEntryJSON. Реализовать маппинг данных из текстового формата в JSON и обратно.

Задача 34. Определить два класса ItemDTO и AggregatedItemsEntity. Реализовать маппинг, который агрегирует список объектов ItemDTO в один объект AggregatedItemsEntity.

Задача 35. Создать два класса LocalizedProductDTO и LocalizedProductEntity. Реализовать маппинг данных с учетом языковой локализации полей (например, перевод названий продуктов).

## 3. Принципы SOLID

Пример решений задач

Задача 1. Создайте систему для управления отчетами. Реализуйте класс Report, который отвечает за хранение данных отчета. Создайте интерфейс IReportFormatter с методами для форматирования отчетов. Реализуйте несколько форматтеров (например, PDFReportFormatter, HTMLReportFormatter). Класс ReportPrinter использует IReportFormatter для печати отчетов. Обеспечьте, чтобы Report и ReportPrinter могли работать с любыми форматтерами, не изменяя их код.

|  |
| --- |
| С++ |
| #include <iostream>  #include <string>  // Интерфейс для форматирования отчетов  class IReportFormatter {  public:  virtual std::string Format(const std::string& title, const std::string& content) = 0;  virtual ~IReportFormatter() = default; // Виртуальный деструктор для правильного удаления производных классов  };  // Класс, представляющий отчет  class Report {  public:  std::string Title;  std::string Content;  Report(const std::string& title, const std::string& content)  : Title(title), Content(content) {}  };  // Форматтер для PDF-отчетов  class PDFReportFormatter : public IReportFormatter {  public:  std::string Format(const std::string& title, const std::string& content) override {  // Логика форматирования в PDF  return "PDF Report: " + title + "\n" + content;  }  };  // Форматтер для HTML-отчетов  class HTMLReportFormatter : public IReportFormatter {  public:  std::string Format(const std::string& title, const std::string& content) override {  // Логика форматирования в HTML  return "<h1>" + title + "</h1><p>" + content + "</p>";  }  };  // Класс для печати отчетов  class ReportPrinter {  private:  IReportFormatter\* formatter;  public:  // Принцип инверсии зависимостей (DIP): ReportPrinter зависит от абстракции IReportFormatter  ReportPrinter(IReportFormatter\* formatter) : formatter(formatter) {}  void Print(const Report& report) {  std::string formattedReport = formatter->Format(report.Title, report.Content);  std::cout << formattedReport << std::endl;  }  };  // Пример использования  int main() {  Report report("Отчет о продажах", "Данные о продажах за месяц.");  // Используем PDF форматтер  PDFReportFormatter pdfFormatter;  ReportPrinter pdfPrinter(&pdfFormatter);  pdfPrinter.Print(report);  // Используем HTML форматтер  HTMLReportFormatter htmlFormatter;  ReportPrinter htmlPrinter(&htmlFormatter);  htmlPrinter.Print(report);  return 0;  } |
| C# |
| using System;  // Интерфейс для форматирования отчетов  public interface IReportFormatter  {  string Format(Report report);  }  // Класс, представляющий отчет  public class Report  {  public string Title { get; set; }  public string Content { get; set; }  public Report(string title, string content)  {  Title = title;  Content = content;  }  }  // Форматтер для PDF-отчетов  public class PDFReportFormatter : IReportFormatter  {  public string Format(Report report)  {  // Логика форматирования в PDF  return $"PDF Report: {report.Title}\n{report.Content}";  }  }  // Форматтер для HTML-отчетов  public class HTMLReportFormatter : IReportFormatter  {  public string Format(Report report)  {  // Логика форматирования в HTML  return $"<h1>{report.Title}</h1><p>{report.Content}</p>";  }  }  // Класс для печати отчетов  public class ReportPrinter  {  private readonly IReportFormatter \_formatter;  // Принцип инверсии зависимостей (DIP): ReportPrinter зависит от абстракции IReportFormatter  public ReportPrinter(IReportFormatter formatter)  {  \_formatter = formatter;  }  public void Print(Report report)  {  string formattedReport = \_formatter.Format(report);  Console.WriteLine(formattedReport);  }  }  // Пример использования  public class Program  {  public static void Main(string[] args)  {  Report report = new Report("Отчет о продажах", "Данные о продажах за месяц.");  // Используем PDF форматтер  IReportFormatter pdfFormatter = new PDFReportFormatter();  ReportPrinter pdfPrinter = new ReportPrinter(pdfFormatter);  pdfPrinter.Print(report);  // Используем HTML форматтер  IReportFormatter htmlFormatter = new HTMLReportFormatter();  ReportPrinter htmlPrinter = new ReportPrinter(htmlFormatter);  htmlPrinter.Print(report);  }  }  // Принципы SOLID:  // 1. Принцип единственной ответственности (SRP):  // Каждый класс имеет свою ответственность. Report хранит данные отчета,  // IReportFormatter отвечает за форматирование, а ReportPrinter — за печать.  // 2. Принцип открытости/закрытости (OCP):  // Классы форматтеров могут быть добавлены без изменения кода Report и ReportPrinter.  // Если мы захотим добавить новый форматтер, например, XMLReportFormatter,  // мы просто создадим новый класс, реализующий IReportFormatter.  // 3. Принцип подстановки Лисков (LSP):  // Все форматтеры могут быть использованы в ReportPrinter, не нарушая его работу.  // Например, если мы создадим новый форматтер, он будет работать с ReportPrinter так же, как и существующие.  // 4. Принцип разделения интерфейса (ISP):  // Интерфейс IReportFormatter является узкоспециализированным и не содержит ненужных методов.  // Он предоставляет только один метод Format, который необходим для форматирования отчетов.  // 5. Принцип инверсии зависимостей (DIP):  // ReportPrinter зависит от абстракции IReportFormatter, а не от конкретных реализаций.  // Это позволяет легко заменять форматтеры, не изменяя код ReportPrinter. |
| Java |
| // Интерфейс для форматирования отчетов  public interface IReportFormatter {  String format(Report report);  }  // Класс, представляющий отчет  public class Report {  private String title;  private String content;  public Report(String title, String content) {  this.title = title;  this.content = content;  }  public String getTitle() {  return title;  }  public String getContent() {  return content;  }  }  // Форматтер для PDF-отчетов  public class PDFReportFormatter implements IReportFormatter {  @Override  public String format(Report report) {  // Логика форматирования в PDF  return "PDF Report: " + report.getTitle() + "\n" + report.getContent();  }  }  // Форматтер для HTML-отчетов  public class HTMLReportFormatter implements IReportFormatter {  @Override  public String format(Report report) {  // Логика форматирования в HTML  return "<h1>" + report.getTitle() + "</h1><p>" + report.getContent() + "</p>";  }  }  // Класс для печати отчетов  public class ReportPrinter {  private final IReportFormatter formatter;  // Принцип инверсии зависимостей (DIP): ReportPrinter зависит от абстракции IReportFormatter  public ReportPrinter(IReportFormatter formatter) {  this.formatter = formatter;  }  public void print(Report report) {  String formattedReport = formatter.format(report);  System.out.println(formattedReport);  }  }  // Пример использования  public class Main {  public static void main(String[] args) {  Report report = new Report("Отчет о продажах", "Данные о продажах за месяц.");  // Используем PDF форматтер  IReportFormatter pdfFormatter = new PDFReportFormatter();  ReportPrinter pdfPrinter = new ReportPrinter(pdfFormatter);  pdfPrinter.print(report);  // Используем HTML форматтер  IReportFormatter htmlFormatter = new HTMLReportFormatter();  ReportPrinter htmlPrinter = new ReportPrinter(htmlFormatter);  htmlPrinter.print(report);  }  }  // Принципы SOLID:  // 1. Принцип единственной ответственности (SRP):  // Каждый класс имеет свою ответственность. Report хранит данные отчета,  // IReportFormatter отвечает за форматирование, а ReportPrinter — за печать.  // 2. Принцип открытости/закрытости (OCP):  // Классы форматтеров могут быть добавлены без изменения кода Report и ReportPrinter.  // Если мы захотим добавить новый форматтер, например, XMLReportFormatter,  // мы просто создадим новый класс, реализующий IReportFormatter.  // 3. Принцип подстановки Лисков (LSP):  // Все форматтеры могут быть использованы в ReportPrinter, не нарушая его работу.  // Например, если мы создадим новый форматтер, он будет работать с ReportPrinter так же, как и существующие.  // 4. Принцип разделения интерфейса (ISP):  // Интерфейс IReportFormatter является узкоспециализированным и не содержит ненужных методов.  // Он предоставляет только один метод format, который необходим для форматирования отчетов.  // 5. Принцип инверсии зависимостей (DIP):  // ReportPrinter зависит от абстракции IReportFormatter, а не от конкретных реализаций.  // Это позволяет легко заменять форматтеры, не изменяя код ReportPrinter. |
| Python |
| # Интерфейс для форматирования отчетов  from abc import ABC, abstractmethod  class IReportFormatter(ABC):  @abstractmethod  def format(self, report):  pass  # Класс, представляющий отчет  class Report:  def \_\_init\_\_(self, title, content):  self.title = title  self.content = content  # Форматтер для PDF-отчетов  class PDFReportFormatter(IReportFormatter):  def format(self, report):  # Логика форматирования в PDF  return f"PDF Report: {report.title}\n{report.content}"  # Форматтер для HTML-отчетов  class HTMLReportFormatter(IReportFormatter):  def format(self, report):  # Логика форматирования в HTML  return f"<h1>{report.title}</h1><p>{report.content}</p>"  # Класс для печати отчетов  class ReportPrinter:  def \_\_init\_\_(self, formatter):  # Принцип инверсии зависимостей (DIP): ReportPrinter зависит от абстракции IReportFormatter  self.formatter = formatter  def print(self, report):  formatted\_report = self.formatter.format(report)  print(formatted\_report)  # Пример использования  if \_\_name\_\_ == "\_\_main\_\_":  report = Report("Отчет о продажах", "Данные о продажах за месяц.")  # Используем PDF форматтер  pdf\_formatter = PDFReportFormatter()  pdf\_printer = ReportPrinter(pdf\_formatter)  pdf\_printer.print(report)  # Используем HTML форматтер  html\_formatter = HTMLReportFormatter()  html\_printer = ReportPrinter(html\_formatter)  html\_printer.print(report)  # Принципы SOLID:  # 1. Принцип единственной ответственности (SRP):  # Каждый класс имеет свою ответственность. Report хранит данные отчета,  # IReportFormatter отвечает за форматирование, а ReportPrinter — за печать.  # 2. Принцип открытости/закрытости (OCP):  # Классы форматтеров могут быть добавлены без изменения кода Report и ReportPrinter.  # Если мы захотим добавить новый форматтер, например, XMLReportFormatter,  # мы просто создадим новый класс, реализующий IReportFormatter.  # 3. Принцип подстановки Лисков (LSP):  # Все форматтеры могут быть использованы в ReportPrinter, не нарушая его работу.  # Например, если мы создадим новый форматтер, он будет работать с ReportPrinter так же, как и существующие.  # 4. Принцип разделения интерфейса (ISP):  # Интерфейс IReportFormatter является узкоспециализированным и не содержит ненужных методов.  # Он предоставляет только один метод format, который необходим для форматирования отчетов.  # 5. Принцип инверсии зависимостей (DIP):  # ReportPrinter зависит от абстракции IReportFormatter, а не от конкретных реализаций.  # Это позволяет легко заменять форматтеры, не изменяя код ReportPrinter. |

### Задачи для самостоятельной работы

Задача 2. Разработайте систему платежей. Создайте интерфейс IPaymentMethod с методом processPayment. Реализуйте несколько классов, таких как CreditCardPayment и PayPalPayment, которые реализуют этот интерфейс. Класс PaymentProcessor использует IPaymentMethod для обработки платежей. Убедитесь, что PaymentProcessor не зависит от конкретных реализаций и может быть расширен новыми методами оплаты.

Задача 3. Реализуйте систему пользовательской авторизации. Создайте класс User с методами для управления данными пользователя. Разделите его на интерфейсы IUserData и IUserAuthentication. Реализуйте несколько классов для аутентификации, таких как PasswordAuthentication и OAuthAuthentication. Класс UserManager зависит от IUserAuthentication для аутентификации пользователей и от IUserData для управления данными.

Задача 4. Создайте систему уведомлений. Реализуйте интерфейс INotificationSender с методами для отправки различных типов уведомлений (например, sendEmail, sendSMS). Создайте несколько классов, таких как EmailNotificationSender и SMSNotificationSender, которые реализуют этот интерфейс. Класс NotificationService использует INotificationSender для отправки уведомлений и может поддерживать новые типы уведомлений без изменения своего кода.

Задача 5. Разработайте планировщик задач. Создайте интерфейс ITask с методами execute и getSchedule. Реализуйте несколько классов, таких как DailyTask и WeeklyTask, которые реализуют этот интерфейс. Класс TaskScheduler использует ITask для планирования и выполнения задач. Убедитесь, что TaskScheduler может работать с любыми задачами, не изменяя их код.

Задача 6. Реализуйте систему обработки файлов. Создайте интерфейс IFileHandler с методами readFile и writeFile. Реализуйте классы TextFileHandler и BinaryFileHandler, которые реализуют этот интерфейс. Класс FileProcessor использует IFileHandler для обработки файлов и может работать с различными форматами файлов без изменения своего кода.

Задача 7. Создайте систему управления заказами. Реализуйте класс Order с методами для добавления и удаления товаров. Создайте интерфейс IOrderProcessor с методом processOrder. Реализуйте несколько классов, таких как StandardOrderProcessor и ExpressOrderProcessor, которые реализуют этот интерфейс. Класс OrderManager использует IOrderProcessor для обработки заказов и может поддерживать новые типы обработки заказов без изменения своего кода.

Задача 8. Реализуйте систему стратегий скидок. Создайте интерфейс IDiscountStrategy с методом applyDiscount. Реализуйте несколько стратегий, таких как PercentageDiscount и FixedAmountDiscount, которые реализуют этот интерфейс. Класс DiscountService использует IDiscountStrategy для применения скидок и может поддерживать новые стратегии без изменения своего кода.

Задача 9. Создайте систему отчетов для бизнеса. Реализуйте интерфейс IReportGenerator с методами generate и format. Реализуйте несколько классов, таких как SalesReportGenerator и InventoryReportGenerator, которые реализуют этот интерфейс. Класс ReportManager использует IReportGenerator для генерации отчетов и может поддерживать новые типы отчетов без изменения своего кода.

Задача 10. Разработайте систему управления ресурсами. Реализуйте интерфейс IResource с методами allocate и release. Реализуйте несколько классов, таких как MemoryResource и NetworkResource, которые реализуют этот интерфейс. Класс ResourceManager использует IResource для управления ресурсами и может поддерживать новые типы ресурсов без изменения своего кода.

Задача 11. Реализуйте систему для управления игровыми объектами. Создайте интерфейс IGameObject с методами update и render. Реализуйте несколько классов, таких как Player и Enemy, которые реализуют этот интерфейс. Класс GameEngine использует IGameObject для обновления и рендеринга игровых объектов и может поддерживать новые типы объектов без изменения своего кода.

Задача 12. Создайте систему для управления плагинами. Реализуйте интерфейс IPlugin с методами initialize и execute. Реализуйте несколько плагинов, таких как LoggingPlugin и AuthenticationPlugin, которые реализуют этот интерфейс. Класс PluginManager использует IPlugin для управления плагинами и может поддерживать новые плагины без изменения своего кода.

Задача 13. Реализуйте систему управления учетными записями. Создайте интерфейс IAccount с методами createAccount и deleteAccount. Реализуйте несколько типов учетных записей, таких как StandardAccount и PremiumAccount, которые реализуют этот интерфейс. Класс AccountManager использует IAccount для управления учетными записями и может поддерживать новые типы учетных записей без изменения своего кода.

Задача 14. Разработайте систему отчетности для приложения. Реализуйте интерфейс IReport с методами generate и save. Реализуйте несколько типов отчетов, таких как PDFReport и CSVReport, которые реализуют этот интерфейс. Класс ReportService использует IReport для генерации и сохранения отчетов и может поддерживать новые типы отчетов без изменения своего кода.

Задача 15. Создайте систему для обработки событий. Реализуйте интерфейс IEvent с методами trigger и subscribe. Реализуйте несколько типов событий, таких как UserLoginEvent и OrderPlacedEvent, которые реализуют этот интерфейс. Класс EventManager использует IEvent для управления событиями и может поддерживать новые типы событий без изменения своего кода.

Задача 16: Реализуйте систему управления конфигурацией. Создайте интерфейс IConfigurationProvider с методами для получения и установки конфигурационных параметров. Реализуйте несколько провайдеров, таких как JsonConfigurationProvider и XmlConfigurationProvider, которые реализуют этот интерфейс. Класс ConfigurationManager использует IConfigurationProvider для управления конфигурацией и может поддерживать новые форматы конфигураций без изменения своего кода.

Задача 17: Разработайте систему обработки запросов. Создайте интерфейс IRequestHandler с методом handleRequest. Реализуйте несколько обработчиков, таких как GetRequestHandler и PostRequestHandler, которые реализуют этот интерфейс. Класс RequestProcessor использует IRequestHandler для обработки запросов и может поддерживать новые типы запросов без изменения своего кода.

Задача 18: Создайте систему логирования. Реализуйте интерфейс ILogger с методами для записи логов (например, logInfo, logError). Создайте несколько логеров, таких как FileLogger и ConsoleLogger, которые реализуют этот интерфейс. Класс LoggingService использует ILogger для записи логов и может поддерживать новые типы логеров без изменения своего кода.

Задача 19: Реализуйте систему фильтрации данных. Создайте интерфейс IDataFilter с методом applyFilter. Реализуйте несколько фильтров, таких как DateRangeFilter и CategoryFilter, которые реализуют этот интерфейс. Класс DataProcessor использует IDataFilter для фильтрации данных и может поддерживать новые фильтры без изменения своего кода.

Задача 20: Разработайте систему обработки изображений. Создайте интерфейс IImageProcessor с методами applyFilter и resize. Реализуйте несколько процессоров, таких как GrayscaleImageProcessor и ResizeImageProcessor, которые реализуют этот интерфейс. Класс ImageEditor использует IImageProcessor для обработки изображений и может поддерживать новые типы процессоров без изменения своего кода.

Задача 21: Создайте систему задач для веб-приложений. Реализуйте интерфейс IWebTask с методами execute и schedule. Создайте несколько задач, таких как EmailTask и DatabaseBackupTask, которые реализуют этот интерфейс. Класс WebTaskScheduler использует IWebTask для планирования и выполнения задач и может поддерживать новые типы задач без изменения своего кода.

Задача 22: Реализуйте систему аутентификации устройств. Создайте интерфейс IDeviceAuthenticator с методом authenticateDevice. Реализуйте несколько аутентификаторов, таких как BiometricAuthenticator и PasswordAuthenticator, которые реализуют этот интерфейс. Класс DeviceManager использует IDeviceAuthenticator для аутентификации устройств и может поддерживать новые типы аутентификаторов без изменения своего кода.

Задача 23: Разработайте систему уведомлений по событиям. Реализуйте интерфейс IEventNotifier с методами notifyOnEvent. Реализуйте несколько типов уведомлений, таких как PushNotificationNotifier и WebhookNotifier, которые реализуют этот интерфейс. Класс EventNotificationService использует IEventNotifier для отправки уведомлений по событиям и может поддерживать новые типы уведомлений без изменения своего кода.

Задача 24: Создайте систему генерации уникальных идентификаторов. Реализуйте интерфейс IIdGenerator с методом generateId. Реализуйте несколько генераторов, таких как UuidGenerator и SequentialIdGenerator, которые реализуют этот интерфейс. Класс IdService использует IIdGenerator для генерации идентификаторов и может поддерживать новые генераторы без изменения своего кода.

Задача 25: Реализуйте систему кэширования данных. Создайте интерфейс ICache с методами get и put. Реализуйте несколько реализаций кэша, таких как InMemoryCache и RedisCache, которые реализуют этот интерфейс. Класс CacheManager использует ICache для управления кэшированием данных и может поддерживать новые типы кэшей без изменения своего кода.

Задача 26: Разработайте систему маршрутизации. Создайте интерфейс IRoute с методами getPath и getHandler. Реализуйте несколько маршрутов, таких как StaticRoute и DynamicRoute, которые реализуют этот интерфейс. Класс Router использует IRoute для маршрутизации запросов и может поддерживать новые маршруты без изменения своего кода.

Задача 27: Создайте систему анализа данных. Реализуйте интерфейс IDataAnalyzer с методами analyze и report. Реализуйте несколько анализаторов, таких как SalesDataAnalyzer и UserBehaviorAnalyzer, которые реализуют этот интерфейс. Класс DataAnalysisService использует IDataAnalyzer для анализа данных и может поддерживать новые типы анализаторов без изменения своего кода.

Задача 28: Реализуйте систему отправки сообщений. Создайте интерфейс IMessageSender с методами для отправки сообщений (например, sendMessage). Реализуйте несколько классов, таких как EmailMessageSender и SMSMessageSender, которые реализуют этот интерфейс. Класс MessageService использует IMessageSender для отправки сообщений и может поддерживать новые типы отправки без изменения своего кода.

Задача 29: Разработайте систему управления задачами. Создайте интерфейс ITaskManager с методами для добавления и удаления задач. Реализуйте несколько классов, таких как SimpleTaskManager и AdvancedTaskManager, которые реализуют этот интерфейс. Класс TaskScheduler использует ITaskManager для управления задачами и может поддерживать новые типы управления задачами без изменения своего кода.

Задача 30: Создайте систему управления пользователями. Реализуйте интерфейс IUserRepository с методами для получения и сохранения пользователей. Реализуйте несколько классов, таких как InMemoryUserRepository и DatabaseUserRepository, которые реализуют этот интерфейс. Класс UserService использует IUserRepository для управления пользователями и может поддерживать новые способы хранения данных без изменения своего кода.

Задача 31: Реализуйте систему формирования отчетов по данным. Создайте интерфейс IDataReportGenerator с методами для генерации и форматирования отчетов. Реализуйте несколько классов, таких как SalesDataReportGenerator и InventoryDataReportGenerator, которые реализуют этот интерфейс. Класс ReportService использует IDataReportGenerator для создания отчетов и может поддерживать новые типы генераторов отчетов без изменения своего кода.

Задача 32: Разработайте систему обработки платежных транзакций. Создайте интерфейс ITransactionProcessor с методами для обработки транзакций (например, processTransaction). Реализуйте несколько классов, таких как CreditCardTransactionProcessor и BankTransferTransactionProcessor, которые реализуют этот интерфейс. Класс TransactionService использует ITransactionProcessor для обработки транзакций и может поддерживать новые способы обработки без изменения своего кода.

Задача 33: Создайте систему управления рабочими процессами. Реализуйте интерфейс IWorkflow с методами для запуска и завершения рабочего процесса. Реализуйте несколько классов, таких как SimpleWorkflow и ComplexWorkflow, которые реализуют этот интерфейс. Класс WorkflowManager использует IWorkflow для управления процессами и может поддерживать новые типы процессов без изменения своего кода.

Задача 34: Реализуйте систему создания графиков. Создайте интерфейс IChartCreator с методами для создания и настройки графиков. Реализуйте несколько классов, таких как BarChartCreator и LineChartCreator, которые реализуют этот интерфейс. Класс ChartService использует IChartCreator для создания графиков и может поддерживать новые типы графиков без изменения своего кода.

Задача 35: Разработайте систему обработки запросов к API. Создайте интерфейс IApiRequestHandler с методами для отправки и получения данных от API. Реализуйте несколько классов, таких как RestApiRequestHandler и GraphQlApiRequestHandler, которые реализуют этот интерфейс. Класс ApiRequestService использует IApiRequestHandler для обработки запросов и может поддерживать новые API без изменения своего кода.

# ГЛАВА 3. ПАТТЕРНЫ СОЗДАНИЯ

## 1. Паттерн Singleton

Пример решений задач

Задача 1. Создать класс PersonSingleton, который может иметь только один экземпляр. Реализовать методы для отображения информации о человеке и изменения возраста. Также необходимо продемонстрировать попытку создать еще один объект этого класса и проверить, что оба объекта ссылаются на один и тот же экземпляр.

|  |
| --- |
| С++ |
| #include <iostream>  #include <string>  class PersonSingleton {  private:  static PersonSingleton\* instance; // Указатель на единственный экземпляр  std::string name;  int age;  // Конструктор и деструктор объявлены как private  PersonSingleton(const std::string& name, int age) : name(name), age(age) {}  ~PersonSingleton() {}  public:  // Метод для получения экземпляра класса  static PersonSingleton\* getInstance(const std::string& name = "", int age = 0) {  if (instance == nullptr) {  instance = new PersonSingleton(name, age);  }  return instance;  }  // Метод для отображения информации о человеке  void displayInfo() const {  std::cout << "Name: " << name << ", Age: " << age << std::endl;  }  // Метод для изменения возраста  void setAge(int newAge) {  age = newAge;  }  };  // Инициализация статического члена  PersonSingleton\* PersonSingleton::instance = nullptr;  int main() {  // Создание первого экземпляра  PersonSingleton\* person1 = PersonSingleton::getInstance("Alice", 30);  person1->displayInfo();  // Попытка создать второй экземпляр  PersonSingleton\* person2 = PersonSingleton::getInstance("Bob", 25);  person2->displayInfo();  // Изменение возраста через первый экземпляр  person1->setAge(35);  person1->displayInfo();  person2->displayInfo(); // Проверяем, что второй экземпляр также изменился  // Проверка, что оба указателя указывают на один и тот же экземпляр  if (person1 == person2) {  std::cout << "Both pointers point to the same instance." << std::endl;  } else {  std::cout << "Different instances." << std::endl;  }  return 0;  } |
| C# |
| using System;  class PersonSingleton  {  private static PersonSingleton instance; // Статический экземпляр  private string name;  private int age;  // Приватный конструктор  private PersonSingleton(string name, int age)  {  this.name = name;  this.age = age;  }  // Метод для получения экземпляра класса  public static PersonSingleton GetInstance(string name = "", int age = 0)  {  if (instance == null)  {  instance = new PersonSingleton(name, age);  }  return instance;  }  // Метод для отображения информации о человеке  public void DisplayInfo()  {  Console.WriteLine($"Name: {name}, Age: {age}");  }  // Метод для изменения возраста  public void SetAge(int newAge)  {  age = newAge;  }  }  class Program  {  static void Main(string[] args)  {  // Создание первого экземпляра  PersonSingleton person1 = PersonSingleton.GetInstance("Alice", 30);  person1.DisplayInfo();  // Попытка создать второй экземпляр  PersonSingleton person2 = PersonSingleton.GetInstance("Bob", 25);  person2.DisplayInfo();  // Изменение возраста через первый экземпляр  person1.SetAge(35);  person1.DisplayInfo();  person2.DisplayInfo(); // Проверяем, что второй экземпляр также изменился  // Проверка, что оба указателя указывают на один и тот же экземпляр  if (person1 == person2)  {  Console.WriteLine("Both pointers point to the same instance.");  }  else  {  Console.WriteLine("Different instances.");  }  }  } |
| Java |
| class PersonSingleton {  private static PersonSingleton instance; // Статический экземпляр  private String name;  private int age;  // Приватный конструктор  private PersonSingleton(String name, int age) {  this.name = name;  this.age = age;  }  // Метод для получения экземпляра класса  public static PersonSingleton getInstance(String name, int age) {  if (instance == null) {  instance = new PersonSingleton(name, age);  }  return instance;  }  // Метод для отображения информации о человеке  public void displayInfo() {  System.out.println("Name: " + name + ", Age: " + age);  }  // Метод для изменения возраста  public void setAge(int newAge) {  age = newAge;  }  }  public class Main {  public static void main(String[] args) {  // Создание первого экземпляра  PersonSingleton person1 = PersonSingleton.getInstance("Alice", 30);  person1.displayInfo();  // Попытка создать второй экземпляр  PersonSingleton person2 = PersonSingleton.getInstance("Bob", 25);  person2.displayInfo();  // Изменение возраста через первый экземпляр  person1.setAge(35);  person1.displayInfo();  person2.displayInfo(); // Проверяем, что второй экземпляр также изменился  // Проверка, что оба указателя указывают на один и тот же экземпляр  if (person1 == person2) {  System.out.println("Both pointers point to the same instance.");  } else {  System.out.println("Different instances.");  }  }  } |
| Python |
| class PersonSingleton:  \_instance = None # Статический экземпляр  def \_\_new\_\_(cls, name="", age=0):  if cls.\_instance is None:  cls.\_instance = super(PersonSingleton, cls).\_\_new\_\_(cls)  cls.\_instance.name = name  cls.\_instance.age = age  return cls.\_instance  def display\_info(self):  print(f"Name: {self.name}, Age: {self.age}")  def set\_age(self, new\_age):  self.age = new\_age  # Демонстрация работы класса  if \_\_name\_\_ == "\_\_main\_\_":  # Создание первого экземпляра  person1 = PersonSingleton("Alice", 30)  person1.display\_info()  # Попытка создать второй экземпляр  person2 = PersonSingleton("Bob", 25)  person2.display\_info()  # Изменение возраста через первый экземпляр  person1.set\_age(35)  person1.display\_info()  person2.display\_info() # Проверяем, что второй экземпляр также изменился  # Проверка, что оба указателя указывают на один и тот же экземпляр  if person1 is person2:  print("Both pointers point to the same instance.")  else:  print("Different instances.") |

### Задачи для самостоятельной работы

Задача 2. Определить класс DocumentSingleton, представляющий документ, который может быть только один. Реализовать методы для обновления содержимого и отображения полной информации о документе.

Задача 3. Создать класс ProductSingleton, который представляет товар на складе. Реализовать методы для вычисления общей стоимости товара на складе и обновления цены. Экземпляр класса должен быть только один.

Задача 4. Определить класс CartItemSingleton, который представляет продукт в корзине и позволяет изменять количество и вычислять общую стоимость. Создать возможность иметь только один экземпляр этого класса.

Задача 5. Реализовать класс StackSingleton, представляющий стек с методами push, pop, peek и isEmpty. Убедиться, что существует только один экземпляр этого стека.

Задача 6. Создать класс QueueSingleton, реализующий очередь с методами enqueue, dequeue, peek и isEmpty. Гарантировать, что существует только один экземпляр этой очереди.

Задача 7. Определить класс ShapeSingleton, который может быть только один и представляет геометрическую фигуру. Создать методы для вычисления площади и возможности изменения типа фигуры.

Задача 8. Реализовать класс VehicleSingleton, который может иметь только один экземпляр и представляет транспортное средство. Добавить методы для изменения марки и модели.

Задача 9. Определить класс EventSingleton, который представляет одно уникальное событие. Реализовать методы для изменения места проведения и отображения информации о событии.

Задача 10. Создать класс CourseSingleton, который может иметь только один экземпляр и представляет учебный курс. Реализовать методы для изменения количества кредитов и отображения информации о курсе.

Задача 11. Определить класс LeaveRequestSingleton, представляющий единственную заявку на отпуск. Реализовать методы для изменения дат отпуска и отображения информации о заявке.

Задача 12. Создать класс ProfileSingleton, представляющий единственный профиль пользователя. Реализовать методы для изменения электронной почты и отображения информации о профиле.

Задача 13. Определить класс ContactSingleton, представляющий единственный контакт. Реализовать методы для изменения номера телефона и отображения информации о контакте.

Задача 14. Создать класс TaskSingleton, который представляет единственную задачу. Реализовать методы для изменения описания и отображения информации о задаче.

Задача 15. Определить класс ComputerSingleton, представляющий единственный компьютер. Реализовать методы для изменения модели и отображения информации о компьютере.

Задача 16. Создать класс BookSingleton, представляющий одну книгу. Реализовать методы для изменения автора и отображения информации о книге.

Задача 17. Реализовать класс CurrencyRateSingleton, который будет представлять собой единственный экземпляр курса валюты. Класс должен включать методы для изменения курса валюты и отображения информации о текущем курсе.

Задача 18. Создать класс ClientSingleton, представляющий единственного клиента. Реализовать методы для изменения баланса и отображения информации о клиенте.

Задача 19. Определить класс AccountSingleton, представляющий единственный банковский счёт. Реализовать методы для пополнения счёта, списания средств и отображения информации о счёте.

Задача 20. Создать класс CompanySingleton, представляющий единственную фирму. Реализовать методы для изменения адреса и отображения информации о фирме.

Задача 21. Определить класс EmployeeSingleton, представляющий единственного работника. Реализовать методы для изменения должности и отображения информации о работнике.

Задача 22. Создать класс MovieSingleton, представляющий единственный фильм. Реализовать методы для изменения режиссёра и отображения информации о фильме.

Задача 23. Определить класс UniversitySingleton, представляющий единственный университет. Реализовать методы для добавления студентов и отображения информации о университете.

Задача 24. Создать класс PaymentSingleton, представляющий единственный платёж. Реализовать методы для изменения суммы и отображения информации о платеже.

Задача 25. Определить класс InvoiceSingleton, представляющий единственную оплату. Реализовать методы для изменения суммы и отображения информации о платеже.

Задача 26. Создать класс RecipeSingleton, представляющий единственный рецепт. Реализовать методы для изменения времени приготовления и отображения информации о рецепте.

Задача 27. Определить класс GadgetSingleton, представляющий единственный гаджет. Реализовать методы для изменения цены и отображения информации о гаджете.

Задача 28. Создать класс TransactionSingleton, представляющий единственную финансовую операцию. Реализовать методы для изменения суммы и отображения информации о финансовой операции.

Задача 29. Определить класс HouseSingleton, представляющий единственный дом. Реализовать методы для изменения площади и отображения информации о доме.

Задача 30. Создать класс ProjectSingleton, представляющий единственный проект. Реализовать методы для изменения описания и отображения информации о проекте.

Задача 31. Определить класс BuildingSingleton, представляющий единственное здание. Реализовать методы для изменения этажности и отображения информации о здании.

Задача 32. Создать класс TrainingCourseSingleton, представляющий единственный курс обучения. Реализовать методы для изменения стоимости и отображения информации о курсе.

Задача 33. Определить класс SystemSingleton, представляющий единственную систему. Реализовать методы для изменения версии и отображения информации о системе.

Задача 34. Создать класс RentalSingleton, представляющий единственный объект аренды. Реализовать методы для изменения дат аренды и отображения информации о аренде.

Задача 35. Определить класс MapSingleton, представляющий единственную карту. Реализовать методы для изменения масштаба и отображения информации о карте.

## 2. Factory Method

Пример решений задач

Задача 1. Создать абстрактный класс PersonCreator с методом createPerson(), который должен создавать объекты типа Person. Реализовать классы AdultCreator и ChildCreator, создающие взрослых и детей соответственно.

|  |
| --- |
| С++ |
| #include <iostream>  #include <string>  // Абстрактный класс Person  class Person {  public:  virtual std::string getType() = 0; // Чисто виртуальный метод  };  // Конкретный класс Adult  class Adult : public Person {  public:  std::string getType() override {  return "Взрослый";  }  };  // Конкретный класс Child  class Child : public Person {  public:  std::string getType() override {  return "Ребёнок";  }  };  // Абстрактный класс PersonCreator  class PersonCreator {  public:  virtual Person\* createPerson() = 0; // Чисто виртуальный метод  };  // Конкретный класс AdultCreator  class AdultCreator : public PersonCreator {  public:  Person\* createPerson() override {  return new Adult(); // Создание объекта Adult  }  };  // Конкретный класс ChildCreator  class ChildCreator : public PersonCreator {  public:  Person\* createPerson() override {  return new Child(); // Создание объекта Child  }  };  int main() {  PersonCreator\* adultCreator = new AdultCreator();  Person\* adult = adultCreator->createPerson();  std::cout << adult->getType() << std::endl; // Вывод: Взрослый  PersonCreator\* childCreator = new ChildCreator();  Person\* child = childCreator->createPerson();  std::cout << child->getType() << std::endl; // Вывод: Ребёнок  delete adult;  delete child;  delete adultCreator;  delete childCreator;  return 0;  } |
| C# |
| using System;  // Абстрактный класс Person  abstract class Person  {  public abstract string GetType(); // Абстрактный метод  }  // Конкретный класс Adult  class Adult : Person  {  public override string GetType()  {  return "Взрослый";  }  }  // Конкретный класс Child  class Child : Person  {  public override string GetType()  {  return "Ребёнок";  }  }  // Абстрактный класс PersonCreator  abstract class PersonCreator  {  public abstract Person CreatePerson(); // Абстрактный метод  }  // Конкретный класс AdultCreator  class AdultCreator : PersonCreator  {  public override Person CreatePerson()  {  return new Adult(); // Создание объекта Adult  }  }  // Конкретный класс ChildCreator  class ChildCreator : PersonCreator  {  public override Person CreatePerson()  {  return new Child(); // Создание объекта Child  }  }  class Program  {  static void Main()  {  PersonCreator adultCreator = new AdultCreator();  Person adult = adultCreator.CreatePerson();  Console.WriteLine(adult.GetType()); // Вывод: Взрослый  PersonCreator childCreator = new ChildCreator();  Person child = childCreator.CreatePerson();  Console.WriteLine(child.GetType()); // Вывод: Ребёнок  }  } |
| Java |
| // Абстрактный класс Person  abstract class Person {  public abstract String getType(); // Абстрактный метод  }  // Конкретный класс Adult  class Adult extends Person {  @Override  public String getType() {  return "Взрослый";  }  }  // Конкретный класс Child  class Child extends Person {  @Override  public String getType() {  return "Ребёнок";  }  }  // Абстрактный класс PersonCreator  abstract class PersonCreator {  public abstract Person createPerson(); // Абстрактный метод  }  // Конкретный класс AdultCreator  class AdultCreator extends PersonCreator {  @Override  public Person createPerson() {  return new Adult(); // Создание объекта Adult  }  }  // Конкретный класс ChildCreator  class ChildCreator extends PersonCreator {  @Override  public Person createPerson() {  return new Child(); // Создание объекта Child  }  }  public class Main {  public static void main(String[] args) {  PersonCreator adultCreator = new AdultCreator();  Person adult = adultCreator.createPerson();  System.out.println(adult.getType()); // Вывод: Взрослый  PersonCreator childCreator = new ChildCreator();  Person child = childCreator.createPerson();  System.out.println(child.getType()); // Вывод: Ребёнок  }  } |
| Python |
| # Абстрактный класс Person  from abc import ABC, abstractmethod  class Person(ABC):  @abstractmethod  def get\_type(self):  pass  # Конкретный класс Adult  class Adult(Person):  def get\_type(self):  return "Взрослый"  # Конкретный класс Child  class Child(Person):  def get\_type(self):  return "Ребёнок"  # Абстрактный класс PersonCreator  class PersonCreator(ABC):  @abstractmethod  def create\_person(self):  pass  # Конкретный класс AdultCreator  class AdultCreator(PersonCreator):  def create\_person(self):  return Adult() # Создание объекта Adult  # Конкретный класс ChildCreator  class ChildCreator(PersonCreator):  def create\_person(self):  return Child() # Создание объекта Child  # Пример использования  if \_\_name\_\_ == "\_\_main\_\_":  adult\_creator = AdultCreator()  adult = adult\_creator.create\_person()  print(adult.get\_type()) # Вывод: Взрослый  child\_creator = ChildCreator()  child = child\_creator.create\_person()  print(child.get\_type()) # Вывод: Ребёнок |

### Задачи для самостоятельной работы

Задача 2. Определить абстрактный класс DocumentCreator с методом createDocument(). Реализовать подклассы WordDocumentCreator и PDFDocumentCreator для создания документов разных типов.

Задача 3. Создать абстрактный класс ProductCreator с методом createProduct(). Реализовать подклассы ElectronicsCreator и FurnitureCreator, создающие электронику и мебель.

Задача 4. Определить абстрактный класс CartItemCreator с методом createCartItem(). Реализовать классы FoodItemCreator и ClothingItemCreator, создающие продукты питания и одежду.

Задача 5. Реализовать абстрактный класс StackCreator с методом createStack(). Создать подклассы ArrayStackCreator и LinkedListStackCreator, создающие стек на основе массива и связанного списка.

Задача 6. Создать абстрактный класс QueueCreator с методом createQueue(). Реализовать классы ArrayQueueCreator и LinkedListQueueCreator, создающие очередь на основе массива и связанного списка.

Задача 7. Определить абстрактный класс ShapeCreator с методом createShape(). Реализовать классы CircleCreator и RectangleCreator, создающие круги и прямоугольники.

Задача 8. Создать абстрактный класс VehicleCreator с методом createVehicle(). Реализовать подклассы CarCreator и BicycleCreator, создающие автомобили и велосипеды.

Задача 9. Определить абстрактный класс EventCreator с методом createEvent(). Реализовать классы ConferenceCreator и ConcertCreator, создающие конференции и концерты.

Задача 10. Создать абстрактный класс CourseCreator с методом createCourse(). Реализовать классы MathCourseCreator и HistoryCourseCreator, создающие курсы по математике и истории.

Задача 11. Определить абстрактный класс LeaveRequestCreator с методом createLeaveRequest(). Реализовать подклассы AnnualLeaveCreator и SickLeaveCreator, создающие заявки на ежегодный и больничный отпуск.

Задача 12. Создать абстрактный класс ProfileCreator с методом createProfile(). Реализовать классы UserProfileCreator и AdminProfileCreator, создающие профили пользователей и администраторов.

Задача 13. Определить абстрактный класс ContactCreator с методом createContact(). Реализовать классы PersonalContactCreator и BusinessContactCreator, создающие личные и деловые контакты.

Задача 14. Создать абстрактный класс TaskCreator с методом createTask(). Реализовать подклассы SimpleTaskCreator и ComplexTaskCreator, создающие простые и сложные задачи.

Задача 15. Определить абстрактный класс ComputerCreator с методом createComputer(). Реализовать классы DesktopCreator и LaptopCreator, создающие настольные и портативные компьютеры.

Задача 16. Создать абстрактный класс BookCreator с методом createBook(). Реализовать классы FictionBookCreator и NonFictionBookCreator, создающие художественные и научно-популярные книги.

Задача 17. Определить абстрактный класс CurrencyRateCreator с методом createCurrencyRate(). Реализовать классы USDCreator и EURCreator, создающие курсы доллара и евро.

Задача 18. Создать абстрактный класс ClientCreator с методом createClient(). Реализовать классы RetailClientCreator и CorporateClientCreator, создающие розничных и корпоративных клиентов.

Задача 19. Определить абстрактный класс AccountCreator с методом createAccount(). Реализовать подклассы SavingsAccountCreator и CheckingAccountCreator, создающие сберегательные и текущие счета.

Задача 20. Создать абстрактный класс CompanyCreator с методом createCompany(). Реализовать классы TechCompanyCreator и RetailCompanyCreator, создающие технологические и розничные компании.

Задача 21. Определить абстрактный класс EmployeeCreator с методом createEmployee(). Реализовать классы ManagerCreator и EngineerCreator, создающие менеджеров и инженеров.

Задача 22. Создать абстрактный класс MovieCreator с методом createMovie(). Реализовать подклассы ActionMovieCreator и ComedyMovieCreator, создающие фильмы боевиков и комедии.

Задача 23. Определить абстрактный класс UniversityCreator с методом createUniversity(). Реализовать классы PublicUniversityCreator и PrivateUniversityCreator, создающие государственные и частные университеты.

Задача 24. Создать абстрактный класс PaymentCreator с методом createPayment(). Реализовать классы CreditCardPaymentCreator и PayPalPaymentCreator, создающие платежи через кредитную карту и PayPal.

Задача 25. Определить абстрактный класс InvoiceCreator с методом createInvoice(). Реализовать подклассы UtilityInvoiceCreator и MedicalInvoiceCreator, создающие счета за коммунальные услуги и медицинские счета.

Задача 26. Создать абстрактный класс RecipeCreator с методом createRecipe(). Реализовать классы VegetarianRecipeCreator и NonVegetarianRecipeCreator, создающие вегетарианские и мясные рецепты.

Задача 27. Определить абстрактный класс GadgetCreator с методом createGadget(). Реализовать классы SmartphoneCreator и TabletCreator, создающие смартфоны и планшеты.

Задача 28. Создать абстрактный класс TransactionCreator с методом createTransaction(). Реализовать подклассы DepositTransactionCreator и WithdrawalTransactionCreator, создающие операции депозита и снятия.

Задача 29. Определить абстрактный класс HouseCreator с методом createHouse(). Реализовать классы UrbanHouseCreator и RuralHouseCreator, создающие дома в городской и сельской местности.

Задача 30. Создать абстрактный класс ProjectCreator с методом createProject(). Реализовать классы ITProjectCreator и ConstructionProjectCreator, создающие ИТ-проекты и строительные проекты.

Задача 31. Определить абстрактный класс BuildingCreator с методом createBuilding(). Реализовать классы ResidentialBuildingCreator и CommercialBuildingCreator, создающие жилые и коммерческие здания.

Задача 32. Создать абстрактный класс TrainingCourseCreator с методом createTrainingCourse(). Реализовать классы OnlineCourseCreator и OfflineCourseCreator, создающие онлайн и офлайн курсы.

Задача 33. Определить абстрактный класс SystemCreator с методом createSystem(). Реализовать классы OperatingSystemCreator и DatabaseSystemCreator, создающие операционные системы и системы баз данных.

Задача 34. Создать абстрактный класс RentalCreator с методом createRental(). Реализовать классы CarRentalCreator и ApartmentRentalCreator, создающие аренду автомобилей и квартир.

Задача 35. Определить абстрактный класс MapCreator с методом createMap(). Реализовать классы RoadMapCreator и TopographicMapCreator, создающие дорожные и топографические карты.

## 3. Abstract Factory

Пример решений задач

Задача 1. Создать абстрактную фабрику PersonFactory, которая будет создавать объекты Person и соответствующие документы (например, паспорт). Реализовать фабрики для взрослых и детей.

|  |
| --- |
| С++ |
| #include <iostream>  #include <string>  // Абстрактный класс для документа  class Document {  public:  virtual std::string getType() = 0; // Чтение типа документа  };  // Конкретный класс для паспорта  class Passport : public Document {  public:  std::string getType() override {  return "Passport";  }  };  // Конкретный класс для свидетельства о рождении  class BirthCertificate : public Document {  public:  std::string getType() override {  return "Birth Certificate";  }  };  // Абстрактный класс для человека  class Person {  public:  virtual std::string getName() = 0; // Чтение имени человека  virtual Document\* createDocument() = 0; // Создание документа  };  // Конкретный класс для взрослого  class Adult : public Person {  public:  std::string getName() override {  return "Adult";  }    Document\* createDocument() override {  return new Passport();  }  };  // Конкретный класс для ребенка  class Child : public Person {  public:  std::string getName() override {  return "Child";  }    Document\* createDocument() override {  return new BirthCertificate();  }  };  // Абстрактная фабрика  class PersonFactory {  public:  virtual Person\* createPerson() = 0; // Создание человека  };  // Фабрика для взрослых  class AdultFactory : public PersonFactory {  public:  Person\* createPerson() override {  return new Adult();  }  };  // Фабрика для детей  class ChildFactory : public PersonFactory {  public:  Person\* createPerson() override {  return new Child();  }  };  int main() {  PersonFactory\* factory = new AdultFactory();  Person\* adult = factory->createPerson();  Document\* adultDoc = adult->createDocument();    std::cout << adult->getName() << " has a " << adultDoc->getType() << std::endl;  delete adultDoc;  delete adult;  delete factory;  factory = new ChildFactory();  Person\* child = factory->createPerson();  Document\* childDoc = child->createDocument();    std::cout << child->getName() << " has a " << childDoc->getType() << std::endl;  delete childDoc;  delete child;  delete factory;  return 0;  } |
| C# |
| using System;  // Абстрактный класс для документа  abstract class Document  {  public abstract string GetType(); // Чтение типа документа  }  // Конкретный класс для паспорта  class Passport : Document  {  public override string GetType() => "Passport";  }  // Конкретный класс для свидетельства о рождении  class BirthCertificate : Document  {  public override string GetType() => "Birth Certificate";  }  // Абстрактный класс для человека  abstract class Person  {  public abstract string GetName(); // Чтение имени человека  public abstract Document CreateDocument(); // Создание документа  }  // Конкретный класс для взрослого  class Adult : Person  {  public override string GetName() => "Adult";    public override Document CreateDocument() => new Passport();  }  // Конкретный класс для ребенка  class Child : Person  {  public override string GetName() => "Child";    public override Document CreateDocument() => new BirthCertificate();  }  // Абстрактная фабрика  abstract class PersonFactory  {  public abstract Person CreatePerson(); // Создание человека  }  // Фабрика для взрослых  class AdultFactory : PersonFactory  {  public override Person CreatePerson() => new Adult();  }  // Фабрика для детей  class ChildFactory : PersonFactory  {  public override Person CreatePerson() => new Child();  }  class Program  {  static void Main()  {  PersonFactory factory = new AdultFactory();  Person adult = factory.CreatePerson();  Document adultDoc = adult.CreateDocument();    Console.WriteLine($"{adult.GetName()} has a {adultDoc.GetType()}");  factory = new ChildFactory();  Person child = factory.CreatePerson();  Document childDoc = child.CreateDocument();    Console.WriteLine($"{child.GetName()} has a {childDoc.GetType()}");  }  } |
| Java |
| // Абстрактный класс для документа  abstract class Document {  public abstract String getType(); // Чтение типа документа  }  // Конкретный класс для паспорта  class Passport extends Document {  public String getType() {  return "Passport";  }  }  // Конкретный класс для свидетельства о рождении  class BirthCertificate extends Document {  public String getType() {  return "Birth Certificate";  }  }  // Абстрактный класс для человека  abstract class Person {  public abstract String getName(); // Чтение имени человека  public abstract Document createDocument(); // Создание документа  }  // Конкретный класс для взрослого  class Adult extends Person {  public String getName() {  return "Adult";  }    public Document createDocument() {  return new Passport();  }  }  // Конкретный класс для ребенка  class Child extends Person {  public String getName() {  return "Child";  }    public Document createDocument() {  return new BirthCertificate();  }  }  // Абстрактная фабрика  abstract class PersonFactory {  public abstract Person createPerson(); // Создание человека  }  // Фабрика для взрослых  class AdultFactory extends PersonFactory {  public Person createPerson() {  return new Adult();  }  }  // Фабрика для детей  class ChildFactory extends PersonFactory {  public Person createPerson() {  return new Child();  }  }  public class Main {  public static void main(String[] args) {  PersonFactory factory = new AdultFactory();  Person adult = factory.createPerson();  Document adultDoc = adult.createDocument();    System.out.println(adult.getName() + " has a " + adultDoc.getType());  factory = new ChildFactory();  Person child = factory.createPerson();  Document childDoc = child.createDocument();    System.out.println(child.getName() + " has a " + childDoc.getType());  }  } |
| Python |
| # Абстрактный класс для документа  class Document:  def get\_type(self):  raise NotImplementedError  # Конкретный класс для паспорта  class Passport(Document):  def get\_type(self):  return "Passport"  # Конкретный класс для свидетельства о рождении  class BirthCertificate(Document):  def get\_type(self):  return "Birth Certificate"  # Абстрактный класс для человека  class Person:  def get\_name(self):  raise NotImplementedError  def create\_document(self):  raise NotImplementedError  # Конкретный класс для взрослого  class Adult(Person):  def get\_name(self):  return "Adult"    def create\_document(self):  return Passport()  # Конкретный класс для ребенка  class Child(Person):  def get\_name(self):  return "Child"    def create\_document(self):  return BirthCertificate()  # Абстрактная фабрика  class PersonFactory:  def create\_person(self):  raise NotImplementedError  # Фабрика для взрослых  class AdultFactory(PersonFactory):  def create\_person(self):  return Adult()  # Фабрика для детей  class ChildFactory(PersonFactory):  def create\_person(self):  return Child()  if \_\_name\_\_ == "\_\_main\_\_":  factory = AdultFactory()  adult = factory.create\_person()  adult\_doc = adult.create\_document()    print(f"{adult.get\_name()} has a {adult\_doc.get\_type()}")  factory = ChildFactory()  child = factory.create\_person()  child\_doc = child.create\_document()    print(f"{child.get\_name()} has a {child\_doc.get\_type()}") |

### Задачи для самостоятельной работы

Задача 2. Определить абстрактную фабрику DocumentFactory, которая будет создавать текстовые и графические документы. Реализовать фабрики для документов Word и PDF.

Задача 3. Создать абстрактную фабрику ProductFactory, которая будет создавать объекты Product и их упаковку. Реализовать фабрики для электроники и мебели.

Задача 4. Определить абстрактную фабрику CartFactory, которая будет создавать объекты CartItem и соответствующие скидки. Реализовать фабрики для продуктов питания и одежды.

Задача 5. Реализовать абстрактную фабрику StackFactory, которая будет создавать стеки и методы их инициализации. Создать фабрики для стеков на основе массива и связанного списка.

Задача 6. Создать абстрактную фабрику QueueFactory, которая будет создавать очереди и их обработчики. Реализовать фабрики для очередей на основе массива и связанного списка.

Задача 7. Определить абстрактную фабрику ShapeFactory, которая будет создавать объекты Shape и их текстуры. Реализовать фабрики для создания кругов и прямоугольников.

Задача 8. Создать абстрактную фабрику VehicleFactory, которая будет создавать объекты Vehicle и их двигатели. Реализовать фабрики для автомобилей и велосипедов.

Задача 9. Определить абстрактную фабрику EventFactory, которая будет создавать объекты Event и их расписание. Реализовать фабрики для создания конференций и концертов.

Задача 10. Создать абстрактную фабрику CourseFactory, которая будет создавать учебные курсы и учебные материалы. Реализовать фабрики для курсов по математике и истории.

Задача 11. Определить абстрактную фабрику LeaveRequestFactory, которая будет создавать заявки на отпуск и их подтверждения. Реализовать фабрики для ежегодного и больничного отпуска.

Задача 12. Создать абстрактную фабрику ProfileFactory, которая будет создавать профили пользователей и их настройки. Реализовать фабрики для пользователей и администраторов.

Задача 13. Определить абстрактную фабрику ContactFactory, которая будет создавать контакты и их группы. Реализовать фабрики для личных и деловых контактов.

Задача 14. Создать абстрактную фабрику TaskFactory, которая будет создавать задачи и их расписания. Реализовать фабрики для простых и сложных задач.

Задача 15. Определить абстрактную фабрику ComputerFactory, которая будет создавать компьютеры и их операционные системы. Реализовать фабрики для настольных и портативных компьютеров.

Задача 16. Создать абстрактную фабрику BookFactory, которая будет создавать книги и их обложки. Реализовать фабрики для художественных и научно-популярных книг.

Задача 17. Определить абстрактную фабрику CurrencyRateFactory, которая будет создавать курсы валют и соответствующие аналитические отчеты. Реализовать фабрики для доллара и евро.

Задача 18. Создать абстрактную фабрику ClientFactory, которая будет создавать клиентов и их учетные записи. Реализовать фабрики для розничных и корпоративных клиентов.

Задача 19. Определить абстрактную фабрику AccountFactory, которая будет создавать банковские счета и их истории операций. Реализовать фабрики для сберегательных и текущих счетов.

Задача 20. Создать абстрактную фабрику CompanyFactory, которая будет создавать компании и их корпоративные документы. Реализовать фабрики для технологических и розничных компаний.

Задача 21. Определить абстрактную фабрику EmployeeFactory, которая будет создавать работников и их трудовые договоры. Реализовать фабрики для менеджеров и инженеров.

Задача 22. Создать абстрактную фабрику MovieFactory, которая будет создавать фильмы и их постеры. Реализовать фабрики для боевиков и комедий.

Задача 23. Определить абстрактную фабрику UniversityFactory, которая будет создавать университеты и их факультеты. Реализовать фабрики для государственных и частных университетов.

Задача 24. Создать абстрактную фабрику PaymentFactory, которая будет создавать платежи и их подтверждения. Реализовать фабрики для кредитных карт и PayPal.

Задача 25. Определить абстрактную фабрику InvoiceFactory, которая будет создавать счета и их квитанции. Реализовать фабрики для коммунальных услуг и медицинских счетов.

Задача 26. Создать абстрактную фабрику RecipeFactory, которая будет создавать рецепты и их ингредиенты. Реализовать фабрики для вегетарианских и мясных рецептов.

Задача 27. Определить абстрактную фабрику GadgetFactory, которая будет создавать гаджеты и их аксессуары. Реализовать фабрики для смартфонов и планшетов.

Задача 28. Создать абстрактную фабрику TransactionFactory, которая будет создавать финансовые операции и их квитанции. Реализовать фабрики для депозитов и снятий.

Задача 29. Определить абстрактную фабрику HouseFactory, которая будет создавать дома и их планы. Реализовать фабрики для городских и сельских домов.

Задача 30. Создать абстрактную фабрику ProjectFactory, которая будет создавать проекты и их документацию. Реализовать фабрики для ИТ-проектов и строительных проектов.

Задача 31. Определите абстрактную фабрику ProjectFactory, которая будет создавать проекты и их бюджеты. Реализуйте фабрики для ИТ-проектов и строительных проектов.

Задача 32. Создайте абстрактную фабрику EmployeeFactory, которая будет создавать сотрудников и их контракты. Реализуйте фабрики для технических специалистов и административных сотрудников.

Задача 33. Определите абстрактную фабрику CourseFactory, которая будет создавать курсы обучения и их материалы. Реализуйте фабрики для курсов по программированию и курсов по управлению.

Задача 34. Создайте абстрактную фабрику CarFactory, которая будет создавать автомобили и их технические характеристики. Реализуйте фабрики для спортивных автомобилей и семейных автомобилей.

Задача 35. Определите абстрактную фабрику GadgetFactory, которая будет создавать электронные гаджеты и их аксессуары. Реализуйте фабрики для смартфонов и умных часов.

## 4. Паттерн Builder

Пример решений задач

Задача 1. Создать класс PersonBuilder для создания объекта Person. Класс Person должен включать поля: имя, возраст и адрес. Реализовать методы в PersonBuilder для установки этих значений и метод build() для создания объекта Person.

|  |
| --- |
| С++ |
| #include <iostream>  #include <string>  // Класс Person  class Person {  public:  std::string name;  int age;  std::string address;  void display() {  std::cout << "Name: " << name << ", Age: " << age << ", Address: " << address << std::endl;  }  };  // Класс PersonBuilder  class PersonBuilder {  private:  Person person;  public:  PersonBuilder& setName(const std::string& name) {  person.name = name;  return \*this;  }  PersonBuilder& setAge(int age) {  person.age = age;  return \*this;  }  PersonBuilder& setAddress(const std::string& address) {  person.address = address;  return \*this;  }  Person build() {  return person;  }  };  int main() {  PersonBuilder builder;  Person person = builder.setName("John Doe").setAge(30).setAddress("123 Main St").build();  person.display();  return 0;  } |
| C# |
| using System;  public class Person  {  public string Name { get; set; }  public int Age { get; set; }  public string Address { get; set; }  public void Display()  {  Console.WriteLine($"Name: {Name}, Age: {Age}, Address: {Address}");  }  }  public class PersonBuilder  {  private Person person = new Person();  public PersonBuilder SetName(string name)  {  person.Name = name;  return this;  }  public PersonBuilder SetAge(int age)  {  person.Age = age;  return this;  }  public PersonBuilder SetAddress(string address)  {  person.Address = address;  return this;  }  public Person Build()  {  return person;  }  }  class Program  {  static void Main()  {  PersonBuilder builder = new PersonBuilder();  Person person = builder.SetName("John Doe").SetAge(30).SetAddress("123 Main St").Build();  person.Display();  }  } |
| Java |
| class Person {  private String name;  private int age;  private String address;  public void display() {  System.out.println("Name: " + name + ", Age: " + age + ", Address: " + address);  }  public static class Builder {  private String name;  private int age;  private String address;  public Builder setName(String name) {  this.name = name;  return this;  }  public Builder setAge(int age) {  this.age = age;  return this;  }  public Builder setAddress(String address) {  this.address = address;  return this;  }  public Person build() {  Person person = new Person();  person.name = this.name;  person.age = this.age;  person.address = this.address;  return person;  }  }  }  public class Main {  public static void main(String[] args) {  Person person = new Person.Builder()  .setName("John Doe")  .setAge(30)  .setAddress("123 Main St")  .build();  person.display();  }  } |
| Python |
| class Person:  def \_\_init\_\_(self, name, age, address):  self.name = name  self.age = age  self.address = address  def display(self):  print(f"Name: {self.name}, Age: {self.age}, Address: {self.address}")  class PersonBuilder:  def \_\_init\_\_(self):  self.name = None  self.age = None  self.address = None  def set\_name(self, name):  self.name = name  return self  def set\_age(self, age):  self.age = age  return self  def set\_address(self, address):  self.address = address  return self  def build(self):  return Person(self.name, self.age, self.address)  if \_\_name\_\_ == "\_\_main\_\_":  builder = PersonBuilder()  person = builder.set\_name("John Doe").set\_age(30).set\_address("123 Main St").build()  person.display() |

### Задачи для самостоятельной работы

Задача 2. Создать класс DocumentBuilder, который строит объект Document с полями название, автор и содержание. Реализовать методы для установки каждого поля и метод build() для завершения процесса создания документа.

Задача 3. Создать класс ProductBuilder для создания объекта Product с полями название, цена и количество на складе. Реализовать методы для установки этих значений и метод build() для создания объекта Product.

Задача 4. Создать класс CartItemBuilder для построения объекта CartItem с полями название продукта, количество и цена. Реализовать методы для установки каждого поля и метод build() для завершения создания объекта CartItem.

Задача 5. Определить класс ShapeBuilder для построения объектов Circle и Rectangle. Реализовать методы для установки соответствующих полей и метод build() для завершения создания объекта.

Задача 6. Создать класс VehicleBuilder для создания объекта Vehicle с полями марка и модель. Реализовать методы для установки каждого поля и метод build() для создания объекта Vehicle.

Задача 7. Создать класс EventBuilder для создания объекта Event с полями название, дата и место проведения. Реализовать методы для установки каждого поля и метод build() для завершения создания объекта Event.

Задача 8. Создать класс CourseBuilder для построения объекта Course с полями название курса, код курса и количество кредитов. Реализовать методы для установки этих значений и метод build() для создания объекта Course.

Задача 9. Создать класс LeaveRequestBuilder для создания объекта LeaveRequest с полями имя сотрудника, начальная дата и конечная дата. Реализовать методы для установки каждого поля и метод build() для завершения создания объекта LeaveRequest.

Задача 10. Создать класс ProfileBuilder для построения объекта Profile с полями имя пользователя, электронная почта и дата рождения. Реализовать методы для установки каждого поля и метод build() для создания объекта Profile.

Задача 11. Создать класс ContactBuilder для создания объекта Contact с полями имя, номер телефона и электронная почта. Реализовать методы для установки каждого поля и метод build() для завершения создания объекта Contact.

Задача 12. Создать класс TaskBuilder для построения объекта Task с полями заголовок задачи, описание и срок выполнения. Реализовать методы для установки этих значений и метод build() для создания объекта Task.

Задача 13. Создать класс ComputerBuilder для создания объекта Computer с полями марка, модель и год выпуска. Реализовать методы для установки каждого поля и метод build() для завершения создания объекта Computer.

Задача 14. Создать класс BookBuilder для построения объекта Book с полями название, автор и ISBN. Реализовать методы для установки этих значений и метод build() для создания объекта Book.

Задача 15. Создать класс CurrencyRateBuilder для создания объекта CurrencyRate с полями название валюты, код валюты и обменный курс. Реализовать методы для установки каждого поля и метод build() для завершения создания объекта CurrencyRate.

Задача 16. Создать класс ClientBuilder для построения объекта Client с полями имя, идентификационный номер и баланс. Реализовать методы для установки этих значений и метод build() для создания объекта Client.

Задача 17. Создать класс AccountBuilder для создания объекта Account с полями номер счёта и баланс. Реализовать методы для установки этих значений и метод build() для завершения создания объекта Account.

Задача 18. Создать класс CompanyBuilder для построения объекта Company с полями название и адрес. Реализовать методы для установки каждого поля и метод build() для создания объекта Company.

Задача 19. Создать класс EmployeeBuilder для создания объекта Employee с полями имя, должность и зарплата. Реализовать методы для установки этих значений и метод build() для завершения создания объекта Employee.

Задача 20. Создать класс MovieBuilder для построения объекта Movie с полями название, режиссёр и год выпуска. Реализовать методы для установки этих значений и метод build() для создания объекта Movie.

Задача 21. Создать класс UniversityBuilder для создания объекта University с полями название и количество студентов. Реализовать методы для установки этих значений и метод build() для завершения создания объекта University.

Задача 22. Создать класс PaymentBuilder для построения объекта Payment с полями сумма и дата. Реализовать методы для установки этих значений и метод build() для создания объекта Payment.

Задача 23. Создать класс InvoiceBuilder для создания объекта Invoice с полями номер счёта и сумма. Реализовать методы для установки этих значений и метод build() для завершения создания объекта Invoice.

Задача 24. Создать класс RecipeBuilder для построения объекта Recipe с полями название, ингредиенты и время приготовления. Реализовать методы для установки этих значений и метод build() для создания объекта Recipe.

Задача 25. Создать класс GadgetBuilder для создания объекта Gadget с полями название, тип и цена. Реализовать методы для установки этих значений и метод build() для завершения создания объекта Gadget.

Задача 26. Создать класс TransactionBuilder для построения объекта Transaction с полями тип операции, сумма и дата. Реализовать методы для установки этих значений и метод build() для создания объекта Transaction.

Задача 27. Создать класс HouseBuilder для создания объекта House с полями адрес, количество комнат и площадь. Реализовать методы для установки этих значений и метод build() для завершения создания объекта House.

Задача 28. Создать класс ProjectBuilder для построения объекта Project с полями название, описание и дата начала. Реализовать методы для установки этих значений и метод build() для создания объекта Project.

Задача 29. Создать класс BuildingBuilder для создания объекта Building с полями адрес, этажи и год постройки. Реализовать методы для установки этих значений и метод build() для завершения создания объекта Building.

Задача 30. Создать класс TrainingCourseBuilder для построения объекта TrainingCourse с полями название, продолжительность и стоимость. Реализовать методы для установки этих значений и метод build() для создания объекта TrainingCourse.

Задача 31. Создать класс SystemBuilder для создания объекта System с полями название и версия. Реализовать методы для установки этих значений и метод build() для завершения создания объекта System.

Задача 32. Создать класс RentalBuilder для построения объекта Rental с полями объект аренды, дата начала и дата окончания. Реализовать методы для установки этих значений и метод build() для создания объекта Rental.

Задача 33. Создать класс MapBuilder для создания объекта Map с полями название и масштаб. Реализовать методы для установки этих значений и метод build() для завершения создания объекта Map.

Задача 34. Создать класс DeliveryBuilder для построения объекта Delivery с полями адрес доставки, дата и время доставки. Реализовать методы для установки этих значений и метод build() для создания объекта Delivery.

Задача 35. Создать класс OrderBuilder для создания объекта Order с полями номер заказа, дата и список товаров. Реализовать методы для установки этих значений и метод build() для создания объекта Order.

## 5. Паттерн Prototype

Пример решений задач

Задача 1. Создать класс ClonedPerson, который наследует класс Person и использует паттерн "Prototype" для клонирования объектов. Реализовать метод clone() для создания копии человека с возможностью изменения возраста.

|  |
| --- |
| С++ |
| #include <iostream>  #include <string>  #include <memory>  class Person {  protected:  std::string name;  int age;  public:  Person(const std::string& name, int age) : name(name), age(age) {}  virtual std::unique\_ptr<Person> clone() const = 0; // Абстрактный метод клонирования  bool operator==(const Person& other) const {  return age == other.age && name == other.name;  }  virtual ~Person() = default; // Виртуальный деструктор  };  class ClonedPerson : public Person {  public:  ClonedPerson(const std::string& name, int age) : Person(name, age) {}  std::unique\_ptr<Person> clone() const override {  return std::make\_unique<ClonedPerson>(name, age); // Клонирование текущего объекта  }  void setAge(int newAge) {  age = newAge; // Изменение возраста  }  };  int main() {  auto original = std::make\_unique<ClonedPerson>("John", 30);  auto copy = original->clone();  std::cout << "Равенство объектов по полям: " << (\*copy == \*original) << std::endl;  std::cout << "Равенство объектов по ссылкам: " << (copy.get() == original.get()) << std::endl;  copy->setAge(35);  std::cout << "Равенство объектов по полям после изменения возраста: " << (\*copy == \*original) << std::endl;  std::cout << "Original: " << original->name << ", Age: " << original->age << std::endl;  std::cout << "Copy: " << copy->name << ", Age: " << copy->age << std::endl;  return 0;  } |
| C# |
| using System;  abstract class Person  {  protected string name;  protected int age;  protected Person(string name, int age)  {  this.name = name;  this.age = age;  }  public abstract Person Clone(); // Абстрактный метод клонирования  public override bool Equals(object obj)  {  if (obj is Person person)  {  return age == person.age && name == person.name;  }  return false;  }  public override int GetHashCode()  {  return HashCode.Combine(name, age);  }  }  class ClonedPerson : Person  {  public ClonedPerson(string name, int age) : base(name, age) { }  public override Person Clone()  {  return new ClonedPerson(name, age); // Клонирование текущего объекта  }  public void SetAge(int newAge)  {  age = newAge; // Изменение возраста  }  }  class Program  {  static void Main(string[] args)  {  ClonedPerson original = new ClonedPerson("John", 30);  ClonedPerson copy = (ClonedPerson)original.Clone();  Console.WriteLine("Равенство объектов по полям: " + copy.Equals(original));  Console.WriteLine("Равенство объектов по ссылкам: " + (copy == original));  copy.SetAge(35);  Console.WriteLine("Равенство объектов по полям после изменения возраста: " + copy.Equals(original));  Console.WriteLine($"Original: {original.name}, Age: {original.age}");  Console.WriteLine($"Copy: {copy.name}, Age: {copy.age}");  }  } |
| Java |
| import java.util.Objects;  abstract class Person {  String name;  int age;  Person(String name, int age) {  this.name = name;  this.age = age;  }  public abstract Person clone(); // Абстрактный метод клонирования  @Override  public boolean equals(Object o) {  if (!(o instanceof Person person)) return false;  return age == person.age && Objects.equals(name, person.name);  }  @Override  public int hashCode() {  return Objects.hash(name, age);  }  }  class ClonedPerson extends Person {  ClonedPerson(String name, int age) {  super(name, age);  }  @Override  public Person clone() {  return new ClonedPerson(name, age); // Клонирование текущего объекта  }  void setAge(int newAge) {  age = newAge; // Изменение возраста  }  }  public class Main {  public static void main(String[] args) {  ClonedPerson original = new ClonedPerson("John", 30);  ClonedPerson copy = (ClonedPerson) original.clone();    System.out.println("Равенство объектов по полям: " + copy.equals(original));  System.out.println("Равенство объектов по ссылкам: " + (copy == original));    copy.setAge(35);    System.out.println("Равенство объектов по полям после изменения возраста: " + copy.equals(original));    System.out.println("Original: " + original.name + ", Age: " + original.age);  System.out.println("Copy: " + copy.name + ", Age: " + copy.age);  }  } |
| Python |
| class Person:  def \_\_init\_\_(self, name, age):  self.name = name  self.age = age  def clone(self):  raise NotImplementedError("Метод 'clone' должен быть переопределен в подклассе")  def \_\_eq\_\_(self, other):  if not isinstance(other, Person):  return False  return self.age == other.age and self.name == other.name  def \_\_hash\_\_(self):  return hash((self.name, self.age))  class ClonedPerson(Person):  def \_\_init\_\_(self, name, age):  super().\_\_init\_\_(name, age)  def clone(self):  return ClonedPerson(self.name, self.age) # Клонирование текущего объекта  def set\_age(self, new\_age):  self.age = new\_age # Изменение возраста  if \_\_name\_\_ == "\_\_main\_\_":  original = ClonedPerson("John", 30)  copy = original.clone()  print("Равенство объектов по полям:", copy == original)  print("Равенство объектов по ссылкам:", copy is original)  copy.set\_age(35)  print("Равенство объектов по полям после изменения возраста:", copy == original)  print("Original: {}, Age: {}".format(original.name, original.age))  print("Copy: {}, Age: {}".format(copy.name, copy.age)) |

### Задачи для самостоятельной работы

Задача 2. Создать класс ClonedDocument, который наследует класс Document и поддерживает клонирование объектов. Реализовать метод clone() для создания копии документа с возможностью изменения содержимого.

Задача 3. Создать класс ClonedProduct, который наследует класс Product и реализует клонирование объектов. Реализовать метод clone() для создания копии товара с возможностью изменения цены и количества.

Задача 4. Создать класс ClonedCartItem, который наследует класс CartItem и поддерживает клонирование. Реализовать метод clone() для создания копии продукта в корзине с возможностью изменения количества.

Задача 5. Создать класс ClonedStack, который наследует класс Stack и реализует клонирование объектов. Реализовать метод clone() для создания копии стека с сохранением текущего состояния элементов.

Задача 6. Создать класс ClonedQueue, который наследует класс Queue и поддерживает клонирование объектов. Реализовать метод clone() для создания копии очереди с текущими элементами.

Задача 7. Создать классы ClonedCircle и ClonedRectangle, которые наследуют классы Circle и Rectangle, соответственно. Реализовать метод clone() в каждом классе для создания копий фигур с возможностью изменения параметров.

Задача 8. Создать классы ClonedCar и ClonedBicycle, которые наследуют классы Car и Bicycle, соответственно. Реализовать метод clone() для создания копий транспортных средств с возможностью изменения модели.

Задача 9. Создать класс ClonedEvent, который наследует класс Event и поддерживает клонирование объектов. Реализовать метод clone() для создания копии события с возможностью изменения даты или места проведения.

Задача 10. Создать класс ClonedCourse, который наследует класс Course и реализует клонирование объектов. Реализовать метод clone() для создания копии курса с возможностью изменения количества кредитов.

Задача 11. Создать класс ClonedLeaveRequest, который наследует класс LeaveRequest и поддерживает клонирование объектов. Реализовать метод clone() для создания копии заявки на отпуск с возможностью изменения дат.

Задача 12. Создать класс ClonedProfile, который наследует класс Profile и реализует клонирование объектов. Реализовать метод clone() для создания копии профиля с возможностью изменения электронной почты.

Задача 13. Создать класс ClonedContact, который наследует класс Contact и поддерживает клонирование объектов. Реализовать метод clone() для создания копии контакта с возможностью изменения номера телефона.

Задача 14. Создать класс ClonedTask, который наследует класс Task и реализует клонирование объектов. Реализовать метод clone() для создания копии задачи с возможностью изменения описания.

Задача 15. Создать класс ClonedComputer, который наследует класс Computer и поддерживает клонирование объектов. Реализовать метод clone() для создания копии компьютера с возможностью изменения модели.

Задача 16. Создать класс ClonedBook, который наследует класс Book и реализует клонирование объектов. Реализовать метод clone() для создания копии книги с возможностью изменения автора.

Задача 17. Создать класс ClonedCurrencyRate, который наследует класс CurrencyRate и поддерживает клонирование объектов. Реализовать метод clone() для создания копии курса валют с возможностью изменения обменного курса.

Задача 18. Создать класс ClonedClient, который наследует класс Client и реализует клонирование объектов. Реализовать метод clone() для создания копии клиента с возможностью изменения баланса.

Задача 19. Создать класс ClonedAccount, который наследует класс Account и поддерживает клонирование объектов. Реализовать метод clone() для создания копии счёта с возможностью изменения баланса.

Задача 20. Создать класс ClonedCompany, который наследует класс Company и реализует клонирование объектов. Реализовать метод clone() для создания копии фирмы с возможностью изменения адреса.

Задача 21. Создать класс ClonedEmployee, который наследует класс Employee и поддерживает клонирование объектов. Реализовать метод clone() для создания копии работника с возможностью изменения должности.

Задача 22. Создать класс ClonedMovie, который наследует класс Movie и реализует клонирование объектов. Реализовать метод clone() для создания копии фильма с возможностью изменения режиссёра.

Задача 23. Создать класс ClonedUniversity, который наследует класс University и поддерживает клонирование объектов. Реализовать метод clone() для создания копии университета с возможностью изменения количества студентов.

Задача 24. Создать класс ClonedPayment, который наследует класс Payment и реализует клонирование объектов. Реализовать метод clone() для создания копии платежа с возможностью изменения суммы.

Задача 25. Создать класс ClonedInvoice, который наследует класс Invoice и поддерживает клонирование объектов. Реализовать метод clone() для создания копии счёта с возможностью изменения суммы.

Задача 26. Создать класс ClonedRecipe, который наследует класс Recipe и реализует клонирование объектов. Реализовать метод clone() для создания копии рецепта с возможностью изменения времени приготовления.

Задача 27. Создать класс ClonedGadget, который наследует класс Gadget и поддерживает клонирование объектов. Реализовать метод clone() для создания копии гаджета с возможностью изменения цены.

Задача 28. Создать класс ClonedTransaction, который наследует класс Transaction и реализует клонирование объектов. Реализовать метод clone() для создания копии финансовой операции с возможностью изменения суммы.

Задача 29. Создать класс ClonedHouse, который наследует класс House и поддерживает клонирование объектов. Реализовать метод clone() для создания копии дома с возможностью изменения площади.

Задача 30. Создать класс ClonedProject, который наследует класс Project и реализует клонирование объектов. Реализовать метод clone() для создания копии проекта с возможностью изменения описания.

Задача 31. Создать класс ClonedBuilding, который наследует класс Building и поддерживает клонирование объектов. Реализовать метод clone() для создания копии здания с возможностью изменения этажности.

Задача 32. Создать класс ClonedTrainingCourse, который наследует класс TrainingCourse и реализует клонирование объектов. Реализовать метод clone() для создания копии курса обучения с возможностью изменения стоимости.

Задача 33. Создать класс ClonedSystem, который наследует класс System и поддерживает клонирование объектов. Реализовать метод clone() для создания копии системы с возможностью изменения версии.

Задача 34. Создать класс ClonedRental, который наследует класс Rental и реализует клонирование объектов. Реализовать метод clone() для создания копии аренды с возможностью изменения даты.

Задача 35. Создать класс ClonedMap, который наследует класс Map и поддерживает клонирование объектов. Реализовать метод clone() для создания копии карты с возможностью изменения масштаба.

# ГЛАВА 4. СТРУКТУРНЫЕ ПАТТЕРНЫ

## 1. Паттерн Adapter

Пример решений задач

Задача 1. Создать класс Person, представляющий человека. Определить интерфейс Employee с методами getJobTitle() и getSalary(). Реализовать класс PersonAdapter, который адаптирует Person к интерфейсу Employee.

|  |
| --- |
| С++ |
| #include <iostream>  #include <string>  // Класс Person, представляющий человека  class Person {  public:  Person(const std::string& name, double salary) : name(name), salary(salary) {}  std::string getName() const {  return name;  }  double getSalary() const {  return salary;  }  private:  std::string name;  double salary;  };  // Интерфейс Employee  class Employee {  public:  virtual std::string getJobTitle() const = 0;  virtual double getSalary() const = 0;  };  // Адаптер PersonAdapter  class PersonAdapter : public Employee {  public:  PersonAdapter(const Person& person) : person(person) {}  std::string getJobTitle() const override {  return "Employee: " + person.getName(); // Возвращаем название должности  }  double getSalary() const override {  return person.getSalary(); // Возвращаем зарплату  }  private:  Person person;  };  int main() {  Person person("John Doe", 50000);  PersonAdapter adapter(person);  std::cout << adapter.getJobTitle() << std::endl; // Вывод: Employee: John Doe  std::cout << adapter.getSalary() << std::endl; // Вывод: 50000  return 0;  } |
| C# |
| using System;  // Класс Person, представляющий человека  class Person  {  public string Name { get; }  public double Salary { get; }  public Person(string name, double salary)  {  Name = name;  Salary = salary;  }  }  // Интерфейс Employee  interface IEmployee  {  string GetJobTitle();  double GetSalary();  }  // Адаптер PersonAdapter  class PersonAdapter : IEmployee  {  private readonly Person person;  public PersonAdapter(Person person)  {  this.person = person;  }  public string GetJobTitle()  {  return "Employee: " + person.Name; // Возвращаем название должности  }  public double GetSalary()  {  return person.Salary; // Возвращаем зарплату  }  }  class Program  {  static void Main()  {  Person person = new Person("John Doe", 50000);  IEmployee adapter = new PersonAdapter(person);  Console.WriteLine(adapter.GetJobTitle()); // Вывод: Employee: John Doe  Console.WriteLine(adapter.GetSalary()); // Вывод: 50000  }  } |
| Java |
| // Класс Person, представляющий человека  class Person {  private String name;  private double salary;  public Person(String name, double salary) {  this.name = name;  this.salary = salary;  }  public String getName() {  return name;  }  public double getSalary() {  return salary;  }  }  // Интерфейс Employee  interface Employee {  String getJobTitle();  double getSalary();  }  // Адаптер PersonAdapter  class PersonAdapter implements Employee {  private Person person;  public PersonAdapter(Person person) {  this.person = person;  }  public String getJobTitle() {  return "Employee: " + person.getName(); // Возвращаем название должности  }  public double getSalary() {  return person.getSalary(); // Возвращаем зарплату  }  }  // Главный класс  public class Main {  public static void main(String[] args) {  Person person = new Person("John Doe", 50000);  Employee adapter = new PersonAdapter(person);  System.out.println(adapter.getJobTitle()); // Вывод: Employee: John Doe  System.out.println(adapter.getSalary()); // Вывод: 50000  }  } |
| Python |
| # Класс Person, представляющий человека  class Person:  def \_\_init\_\_(self, name, salary):  self.name = name  self.salary = salary  # Интерфейс Employee  class Employee:  def get\_job\_title(self):  raise NotImplementedError  def get\_salary(self):  raise NotImplementedError  # Адаптер PersonAdapter  class PersonAdapter(Employee):  def \_\_init\_\_(self, person):  self.person = person  def get\_job\_title(self):  return f"Employee: {self.person.name}" # Возвращаем название должности  def get\_salary(self):  return self.person.salary # Возвращаем зарплату  # Пример использования  if \_\_name\_\_ == "\_\_main\_\_":  person = Person("John Doe", 50000)  adapter = PersonAdapter(person)  print(adapter.get\_job\_title()) # Вывод: Employee: John Doe  print(adapter.get\_salary()) # Вывод: 50000 |

### Задачи для самостоятельной работы

Задача 2. Создать класс Document, который содержит название, автора и содержимое. Определить интерфейс File с методами getFileName() и getFileContent(). Реализовать адаптер, позволяющий использовать Document как File.

Задача 3. Создать класс Product с полями название, цена и количество на складе. Определить интерфейс InventoryItem с методами getName(), getPrice(), и getQuantity(). Реализовать адаптер для использования Product в системе инвентаризации.

Задача 4. Создать класс CartItem, представляющий продукт в корзине. Определить интерфейс OrderItem с методами getProductName(), getQuantity(), и getTotalPrice(). Реализовать адаптер, который позволяет использовать CartItem как элемент заказа.

Задача 5. Создать класс Stack, реализующий стек. Определить интерфейс Collection с методами add(), remove(), и get(). Создать адаптер, который позволяет использовать стек как коллекцию данных.

Задача 6. Создать класс Queue, реализующий очередь. Определить интерфейс List с методами add(), remove(), и getFirst(). Реализовать адаптер, который позволяет использовать очередь как список.

Задача 7. Определить абстрактный класс Shape с методом getArea(). Создать интерфейс PlaneObject с методами getX(), getY(), и getArea(). Реализовать адаптеры для подклассов Circle и Rectangle, чтобы использовать их как объекты на плоскости.

Задача 8. Создать класс Vehicle, который является базовым для транспортных средств. Определить интерфейс Car с методами getMake(), getModel(), и drive(). Создать адаптер для использования Vehicle как Car.

Задача 9. Создать класс Event, описывающий событие. Определить интерфейс CalendarEvent с методами getTitle(), getDate(), и getLocation(). Реализовать адаптер для использования Event в календаре.

Задача 10. Создать класс Course, представляющий учебный курс. Определить интерфейс Subject с методами getName(), getCode(), и getCredits(). Реализовать адаптер для использования курса как учебного предмета.

Задача 11. Создать класс LeaveRequest, представляющий заявку на отпуск. Определить интерфейс Request с методами getEmployeeName(), getStartDate(), и getEndDate(). Создать адаптер для использования заявки как запроса.

Задача 12. Создать класс Profile, описывающий профиль пользователя. Определить интерфейс Account с методами getUsername(), getEmail(), и getBirthDate(). Реализовать адаптер для использования профиля как аккаунта.

Задача 13. Создать класс Contact, представляющий контактную информацию. Определить интерфейс User с методами getName(), getPhoneNumber(), и getEmail(). Создать адаптер для использования контакта как пользователя.

Задача 14. Создать класс Task, представляющий задачу. Определить интерфейс Work с методами getTitle(), getDescription(), и getDeadline(). Реализовать адаптер для использования задачи как работы.

Задача 15. Создать класс Computer, представляющий компьютер. Определить интерфейс ElectronicDevice с методами getBrand(), getModel(), и getYear(). Реализовать адаптер для использования компьютера как электронного устройства.

Задача 16. Создать класс Book, описывающий книгу. Определить интерфейс Publication с методами getTitle(), getAuthor(), и getISBN(). Реализовать адаптер для использования книги как издания.

Задача 17. Создать класс CurrencyRate, описывающий курс валют. Определить интерфейс FinancialInstrument с методами getCurrency(), getCode(), и getExchangeRate(). Создать адаптер для использования курса валют как финансового инструмента.

Задача 18. Создать класс Client, представляющий клиента. Определить интерфейс SystemUser с методами getName(), getID(), и getBalance(). Реализовать адаптер для использования клиента как пользователя системы.

Задача 19. Создать класс Account, описывающий банковский счёт. Определить интерфейс BankAccount с методами getAccountNumber(), getBalance(), и deposit(). Реализовать адаптер для использования счёта как банковского аккаунта.

Задача 20. Создать класс Company, представляющий фирму. Определить интерфейс Organization с методами getName(), getAddress(), и updateAddress(). Создать адаптер для использования фирмы как организации.

Задача 21. Создать класс Employee, представляющий работника. Определить интерфейс StaffMember с методами getName(), getPosition(), и getSalary(). Реализовать адаптер для использования работника как сотрудника.

Задача 22. Создать класс Movie, представляющий фильм. Определить интерфейс Media с методами getTitle(), getDirector(), и getReleaseYear(). Реализовать адаптер для использования фильма как медиа.

Задача 23. Создать класс University, представляющий университет. Определить интерфейс EducationalInstitution с методами getName(), getStudentCount(), и addStudents(). Реализовать адаптер для использования университета как учебного заведения.

Задача 24. Создать класс Payment, представляющий платёж. Определить интерфейс Transaction с методами getAmount(), getDate(), и updateAmount(). Реализовать адаптер для использования платежа как транзакции.

Задача 25. Создать класс Invoice, представляющий счёт на оплату. Определить интерфейс Document с методами getInvoiceNumber(), getAmount(), и updateAmount(). Реализовать адаптер для использования счета как документа.

Задача 26. Создать класс Recipe, описывающий кулинарный рецепт. Определить интерфейс CulinaryProcess с методами getRecipeName(), getIngredients(), и getCookingTime(). Реализовать адаптер для использования рецепта как кулинарного процесса.

Задача 27. Создать класс Gadget, представляющий гаджет. Определить интерфейс ElectronicProduct с методами getName(), getType(), и getPrice(). Создать адаптер для использования гаджета как электронного товара.

Задача 28. Создать класс Transaction, описывающий финансовую операцию. Определить интерфейс AccountingEntry с методами getType(), getAmount(), и getDate(). Реализовать адаптер для использования операции как бухгалтерской записи.

Задача 29. Создать класс House, представляющий дом. Определить интерфейс RealEstate с методами getAddress(), getRoomCount(), и getArea(). Реализовать адаптер для использования дома как объекта недвижимости.

Задача 30. Создать класс Project, представляющий проект. Определить интерфейс WorkAssignment с методами getName(), getDescription(), и getStartDate(). Реализовать адаптер для использования проекта как рабочего задания.

Задача 31. Создать класс Building, представляющий здание. Определить интерфейс RealEstate с методами getAddress(), getFloors(), и getYearBuilt(). Создать адаптер для использования здания как объекта недвижимости.

Задача 32. Создать класс TrainingCourse, представляющий курс обучения. Определить интерфейс StudyProgram с методами getTitle(), getDuration(), и getCost(). Реализовать адаптер для использования курса как учебной программы.

Задача 33. Создать класс System, представляющий систему. Определить интерфейс Software с методами getName(), getVersion(), и updateVersion(). Реализовать адаптер для использования системы как программного обеспечения.

Задача 34. Создать класс Rental, описывающий аренду. Определить интерфейс Contract с методами getRentalObject(), getStartDate(), и getEndDate(). Реализовать адаптер для использования аренды как договора.

Задача 35. Создать класс Map, представляющий карту. Определить интерфейс GeographicObject с методами getName(), getScale(), и updateScale(). Реализовать адаптер для использования карты как географического объекта.

## 2. Паттерн Composite

Пример решений задач

Задача 1. Создать класс CompositeDocument, который может содержать в себе другие документы, представляя их как составные части. Реализуйте методы для добавления, удаления документов и отображения полной информации о всех документах внутри CompositeDocument.

|  |
| --- |
| С++ |
| #include <iostream>  #include <vector>  #include <memory>  #include <string>  // Абстрактный класс для документа  class Document {  public:  virtual void display() const = 0; // Чисто виртуальная функция для отображения информации  virtual ~Document() = default; // Виртуальный деструктор  };  // Класс для представления простого документа  class SimpleDocument : public Document {  private:  std::string name;  public:  SimpleDocument(const std::string& name) : name(name) {}  void display() const override {  std::cout << "Документ: " << name << std::endl;  }  };  // Класс для составного документа  class CompositeDocument : public Document {  private:  std::vector<std::shared\_ptr<Document>> documents;  public:  void add(const std::shared\_ptr<Document>& document) {  documents.push\_back(document); // Добавление документа  }  void remove(const std::shared\_ptr<Document>& document) {  documents.erase(std::remove(documents.begin(), documents.end(), document), documents.end()); // Удаление документа  }  void display() const override {  std::cout << "Составной документ содержит:" << std::endl;  for (const auto& doc : documents) {  doc->display(); // Отображение информации о каждом документе  }  }  };  int main() {  auto composite = std::make\_shared<CompositeDocument>();  composite->add(std::make\_shared<SimpleDocument>("Документ 1"));  composite->add(std::make\_shared<SimpleDocument>("Документ 2"));  composite->display(); // Отображение информации о составном документе  return 0;  } |
| C# |
| using System;  using System.Collections.Generic;  // Абстрактный класс для документа  public abstract class Document  {  public abstract void Display(); // Абстрактный метод для отображения информации  }  // Класс для представления простого документа  public class SimpleDocument : Document  {  private string name;  public SimpleDocument(string name)  {  this.name = name;  }  public override void Display()  {  Console.WriteLine($"Документ: {name}");  }  }  // Класс для составного документа  public class CompositeDocument : Document  {  private List<Document> documents = new List<Document>();  public void Add(Document document)  {  documents.Add(document); // Добавление документа  }  public void Remove(Document document)  {  documents.Remove(document); // Удаление документа  }  public override void Display()  {  Console.WriteLine("Составной документ содержит:");  foreach (var doc in documents)  {  doc.Display(); // Отображение информации о каждом документе  }  }  }  class Program  {  static void Main()  {  CompositeDocument composite = new CompositeDocument();  composite.Add(new SimpleDocument("Документ 1"));  composite.Add(new SimpleDocument("Документ 2"));  composite.Display(); // Отображение информации о составном документе  }  } |
| Java |
| import java.util.ArrayList;  import java.util.List;  // Абстрактный класс для документа  abstract class Document {  public abstract void display(); // Абстрактный метод для отображения информации  }  // Класс для представления простого документа  class SimpleDocument extends Document {  private String name;  public SimpleDocument(String name) {  this.name = name;  }  @Override  public void display() {  System.out.println("Документ: " + name);  }  }  // Класс для составного документа  class CompositeDocument extends Document {  private List<Document> documents = new ArrayList<>();  public void add(Document document) {  documents.add(document); // Добавление документа  }  public void remove(Document document) {  documents.remove(document); // Удаление документа  }  @Override  public void display() {  System.out.println("Составной документ содержит:");  for (Document doc : documents) {  doc.display(); // Отображение информации о каждом документе  }  }  }  public class Main {  public static void main(String[] args) {  CompositeDocument composite = new CompositeDocument();  composite.add(new SimpleDocument("Документ 1"));  composite.add(new SimpleDocument("Документ 2"));  composite.display(); // Отображение информации о составном документе  }  } |
| Python |
| from abc import ABC, abstractmethod  # Абстрактный класс для документа  class Document(ABC):  @abstractmethod  def display(self):  pass # Абстрактный метод для отображения информации  # Класс для представления простого документа  class SimpleDocument(Document):  def \_\_init\_\_(self, name):  self.name = name  def display(self):  print(f"Документ: {self.name}")  # Класс для составного документа  class CompositeDocument(Document):  def \_\_init\_\_(self):  self.documents = [] # Список документов  def add(self, document):  self.documents.append(document) # Добавление документа  def remove(self, document):  self.documents.remove(document) # Удаление документа  def display(self):  print("Составной документ содержит:")  for doc in self.documents:  doc.display() # Отображение информации о каждом документе  # Пример использования  if \_\_name\_\_ == "\_\_main\_\_":  composite = CompositeDocument()  composite.add(SimpleDocument("Документ 1"))  composite.add(SimpleDocument("Документ 2"))  composite.display() # Отображение информации о составном документе |

### Задачи для самостоятельной работы

Задача 2. Создать класс Folder, который может содержать как отдельные документы, так и другие папки. Реализуйте методы для добавления, удаления элементов и отображения структуры всех файлов и папок.

Задача 3. Определить класс ProductCatalog, который может содержать в себе отдельные товары (Product) и другие каталоги. Реализуйте методы для добавления, удаления элементов и отображения структуры каталога с общей стоимостью всех товаров.

Задача 4. Создать класс ShoppingCart, который может содержать отдельные товары и другие корзины. Реализуйте методы для добавления, удаления элементов и вычисления общей стоимости всех товаров во всех корзинах.

Задача 5. Определить класс CompositeStack, который может содержать в себе несколько стеков (Stack). Реализуйте методы для добавления, удаления стеков и работы с элементами всех стеков как с единым стеком.

Задача 6. Создать класс CompositeQueue, который может содержать в себе несколько очередей (Queue). Реализуйте методы для добавления, удаления очередей и работы с элементами всех очередей как с единой очередью.

Задача 7. Определить класс CompositeShape, который может содержать в себе несколько геометрических фигур (Shape). Реализуйте методы для добавления, удаления фигур и вычисления общей площади всех фигур в составе CompositeShape.

Задача 8. Создать класс TransportFleet, который может содержать отдельные транспортные средства (Vehicle) и группы транспортных средств. Реализуйте методы для добавления, удаления транспортных средств и отображения информации о парке.

Задача 9. Определить класс CompositeEvent, который может включать в себя несколько событий (Event). Реализуйте методы для добавления, удаления событий и отображения полной информации о всех событиях в составе CompositeEvent.

Задача 10. Создать класс StudyProgram, который может содержать в себе несколько курсов (Course). Реализуйте методы для добавления, удаления курсов и отображения информации о всей учебной программе.

Задача 11. Определить класс CompositeLeaveRequest, который может включать в себя несколько заявок на отпуск (LeaveRequest). Реализуйте методы для добавления, удаления заявок и отображения информации о всех заявках.

Задача 12. Создать класс CompositeProfile, который может содержать в себе несколько профилей (Profile). Реализуйте методы для добавления, удаления профилей и отображения информации о всех профилях.

Задача 13. Определить класс AddressBook, который может содержать в себе отдельные контакты (Contact) и группы контактов. Реализуйте методы для добавления, удаления контактов и отображения всей адресной книги.

Задача 14. Создать класс ProjectBoard, который может содержать в себе отдельные задачи (Task) и группы задач. Реализуйте методы для добавления, удаления задач и отображения всей доски задач.

Задача 15. Определить класс ComputerCluster, который может включать в себя несколько компьютеров (Computer). Реализуйте методы для добавления, удаления компьютеров и отображения информации о всем кластере.

Задача 16. Создать класс Library, который может содержать отдельные книги (Book) и разделы книг. Реализуйте методы для добавления, удаления книг и отображения информации о всех книгах в библиотеке.

Задача 17. Определить класс FinancialPortfolio, который может содержать в себе отдельные курсы валют (CurrencyRate) и портфели с несколькими валютами. Реализуйте методы для добавления, удаления валют и отображения информации о портфеле.

Задача 18. Создать класс ClientList, который может содержать отдельные записи клиентов (Client) и группы клиентов. Реализуйте методы для добавления, удаления клиентов и отображения всей информации о клиентах.

Задача 19. Определить класс CompositeAccount, который может включать в себя несколько счетов (Account). Реализуйте методы для добавления, удаления счетов и отображения общей информации о всех счетах.

Задача 20. Создать класс Corporation, который может содержать в себе отдельные компании (Company) и группы компаний. Реализуйте методы для добавления, удаления компаний и отображения информации о корпорации.

Задача 21. Определить класс Department, который может включать в себя нескольких работников (Employee) и группы работников. Реализуйте методы для добавления, удаления работников и отображения информации о всём отделе.

Задача 22. Создать класс MovieCollection, который может содержать в себе отдельные фильмы (Movie) и подборки фильмов. Реализуйте методы для добавления, удаления фильмов и отображения информации о коллекции.

Задача 23. Определить класс UniversitySystem, который может включать в себя несколько университетов (University). Реализуйте методы для добавления, удаления университетов и отображения информации о всей системе.

Задача 24. Создать класс FinancialReport, который может содержать отдельные платежи (Payment) и группы платежей. Реализуйте методы для добавления, удаления платежей и отображения информации о всём отчёте.

Задача 25. Определить класс CompositeInvoice, который может включать в себя несколько счетов (Invoice). Реализуйте методы для добавления, удаления счетов и отображения общей информации о всех счетах.

Задача 26. Создать класс Cookbook, который может содержать отдельные рецепты (Recipe) и разделы рецептов. Реализуйте методы для добавления, удаления рецептов и отображения информации о всей книге.

Задача 27. Определить класс GadgetCatalog, который может включать в себя отдельные гаджеты (Gadget) и группы гаджетов. Реализуйте методы для добавления, удаления гаджетов и отображения информации о каталоге.

Задача 28. Создать класс FinancialJournal, который может содержать отдельные финансовые операции (Transaction) и разделы операций. Реализуйте методы для добавления, удаления операций и отображения информации о журнале.

Задача 29. Определить класс CompositeHouse, который может включать в себя несколько домов (House). Реализуйте методы для добавления, удаления домов и отображения общей информации о всех домах.

Задача 30. Создать класс CompositeProject, который может содержать в себе несколько проектов (Project). Реализуйте методы для добавления, удаления проектов и отображения информации о всех проектах в составе.

Задача 31. Определить класс BuildingComplex, который может включать в себя несколько зданий (Building). Реализуйте методы для добавления, удаления зданий и отображения информации о комплексе.

Задача 32. Создать класс EducationalProgram, который может содержать отдельные курсы обучения (TrainingCourse) и их группы. Реализуйте методы для добавления, удаления курсов и отображения информации о всей программе.

Задача 33. Определить класс CompositeSystem, который может включать в себя несколько систем (System). Реализуйте методы для добавления, удаления систем и отображения информации о всех системах в составе.

Задача 34. Создать класс RentalEnterprise, который может содержать в себе несколько арендуемых объектов (Rental). Реализуйте методы для добавления, удаления объектов аренды и отображения информации о предприятии.

Задача 35. Определить класс GeographicalAtlas, который может включать в себя несколько карт (Map). Реализуйте методы для добавления, удаления карт и отображения информации о атласе.

## 3. Паттерн Proxy

Пример решений задач

Задача 1. Создать класс Person, представляющий человека, с полями имя и возраст. Реализовать прокси-класс PersonProxy, который контролирует доступ к методу изменения возраста и отображения информации о человеке. Прокси-класс должен разрешать изменение возраста только при наличии специального разрешения.

|  |
| --- |
| С++ |
| #include <iostream>  #include <string>  class Person {  private:  std::string name;  int age;  public:  Person(const std::string& name, int age) : name(name), age(age) {}  void displayInfo() {  std::cout << "Name: " << name << ", Age: " << age << std::endl;  }  void setAge(int newAge) {  age = newAge;  }  };  class PersonProxy {  private:  Person\* person;  bool hasPermission;  public:  PersonProxy(Person\* p, bool permission) : person(p), hasPermission(permission) {}  void displayInfo() {  person->displayInfo();  }  void setAge(int newAge) {  if (hasPermission) {  person->setAge(newAge);  std::cout << "Age updated to: " << newAge << std::endl;  } else {  std::cout << "Permission denied to change age." << std::endl;  }  }  };  int main() {  Person person("Alice", 30);  PersonProxy proxy(&person, false);  proxy.displayInfo(); // Показать информацию  proxy.setAge(35); // Попытка изменить возраст без разрешения  proxy = PersonProxy(&person, true); // Обновляем прокси с разрешением  proxy.setAge(35); // Изменить возраст с разрешением  proxy.displayInfo(); // Показать обновленную информацию  return 0;  } |
| C# |
| using System;  class Person  {  private string name;  private int age;  public Person(string name, int age)  {  this.name = name;  this.age = age;  }  public void DisplayInfo()  {  Console.WriteLine($"Name: {name}, Age: {age}");  }  public void SetAge(int newAge)  {  age = newAge;  }  }  class PersonProxy  {  private Person person;  private bool hasPermission;  public PersonProxy(Person person, bool permission)  {  this.person = person;  this.hasPermission = permission;  }  public void DisplayInfo()  {  person.DisplayInfo();  }  public void SetAge(int newAge)  {  if (hasPermission)  {  person.SetAge(newAge);  Console.WriteLine($"Age updated to: {newAge}");  }  else  {  Console.WriteLine("Permission denied to change age.");  }  }  }  class Program  {  static void Main()  {  Person person = new Person("Alice", 30);  PersonProxy proxy = new PersonProxy(person, false);  proxy.DisplayInfo(); // Показать информацию  proxy.SetAge(35); // Попытка изменить возраст без разрешения  proxy = new PersonProxy(person, true); // Обновляем прокси с разрешением  proxy.SetAge(35); // Изменить возраст с разрешением  proxy.DisplayInfo(); // Показать обновленную информацию  }  } |
| Java |
| class Person {  private String name;  private int age;  public Person(String name, int age) {  this.name = name;  this.age = age;  }  public void displayInfo() {  System.out.println("Name: " + name + ", Age: " + age);  }  public void setAge(int newAge) {  age = newAge;  }  }  class PersonProxy {  private Person person;  private boolean hasPermission;  public PersonProxy(Person person, boolean permission) {  this.person = person;  this.hasPermission = permission;  }  public void displayInfo() {  person.displayInfo();  }  public void setAge(int newAge) {  if (hasPermission) {  person.setAge(newAge);  System.out.println("Age updated to: " + newAge);  } else {  System.out.println("Permission denied to change age.");  }  }  }  public class Main {  public static void main(String[] args) {  Person person = new Person("Alice", 30);  PersonProxy proxy = new PersonProxy(person, false);  proxy.displayInfo(); // Показать информацию  proxy.setAge(35); // Попытка изменить возраст без разрешения  proxy = new PersonProxy(person, true); // Обновляем прокси с разрешением  proxy.setAge(35); // Изменить возраст с разрешением  proxy.displayInfo(); // Показать обновленную информацию  }  } |
| Python |
| class Person:  def \_\_init\_\_(self, name, age):  self.\_name = name  self.\_age = age  def display\_info(self):  print(f"Name: {self.\_name}, Age: {self.\_age}")  def set\_age(self, new\_age):  self.\_age = new\_age  class PersonProxy:  def \_\_init\_\_(self, person, has\_permission):  self.\_person = person  self.\_has\_permission = has\_permission  def display\_info(self):  self.\_person.display\_info()  def set\_age(self, new\_age):  if self.\_has\_permission:  self.\_person.set\_age(new\_age)  print(f"Age updated to: {new\_age}")  else:  print("Permission denied to change age.")  # Пример использования  person = Person("Alice", 30)  proxy = PersonProxy(person, False)  proxy.display\_info() # Показать информацию  proxy.set\_age(35) # Попытка изменить возраст без разрешения  proxy = PersonProxy(person, True) # Обновляем прокси с разрешением  proxy.set\_age(35) # Изменить возраст с разрешением  proxy.display\_info() # Показать обновленную информацию |

### Задачи для самостоятельной работы

Задача 2. Определить класс Document, содержащий название, автора и содержимое. Реализовать прокси-класс DocumentProxy, который ограничивает доступ к методу обновления содержимого, позволяя его изменение только авторизованным пользователям.

Задача 3. Создать класс Product, содержащий поля название, цена и количество на складе. Реализовать прокси-класс ProductProxy, который позволяет обновлять цену и количество только при наличии административных прав.

Задача 4. Определить класс CartItem с полями название продукта, количество и цена. Реализовать прокси-класс CartItemProxy, который ограничивает возможность изменения количества товара только пользователям с правами "менеджера".

Задача 5. Создать класс Stack, реализующий стек. Разработать прокси-класс StackProxy, который ограничивает доступ к методам push и pop, позволяя добавлять и удалять элементы только при определённых условиях (например, для избежания переполнения).

Задача 6. Определить класс Queue, реализующий очередь. Реализовать прокси-класс QueueProxy, который контролирует доступ к методам enqueue и dequeue, разрешая доступ только пользователям с правами на выполнение операций с очередью.

Задача 7. Создать абстрактный класс Shape с абстрактным методом getArea(). Реализовать прокси-класс ShapeProxy, который проверяет права доступа перед вызовом метода getArea() и логирует все вычисления площади.

Задача 8. Определить базовый класс Vehicle с полями марка и модель. Создать прокси-класс VehicleProxy, который ограничивает доступ к методам, связанным с изменением модели и марки транспортного средства, предоставляя доступ только авторизованным пользователям.

Задача 9. Создать класс Event, содержащий название, дату и место проведения. Реализовать прокси-класс EventProxy, который проверяет права пользователя перед изменением места проведения события и логирует все изменения.

Задача 10. Определить класс Course, представляющий курс с полями название курса, код курса и количество кредитов. Реализовать прокси-класс CourseProxy, который ограничивает доступ к методу изменения количества кредитов, предоставляя доступ только администраторам.

Задача 11. Создать класс LeaveRequest с полями имя сотрудника, начальная дата и конечная дата. Реализовать прокси-класс LeaveRequestProxy, который контролирует доступ к методам изменения дат отпуска и предоставляет доступ только сотрудникам с определенными правами.

Задача 12. Определить класс Profile, представляющий профиль пользователя с полями имя, электронная почта и дата рождения. Реализовать прокси-класс ProfileProxy, который контролирует изменение электронной почты и отображение информации, предоставляя доступ только авторизованным пользователям.

Задача 13. Создать класс Contact с полями имя, номер телефона и электронная почта. Реализовать прокси-класс ContactProxy, который позволяет изменять номер телефона и просматривать информацию о контакте только авторизованным пользователям.

Задача 14. Определить класс Task с полями заголовок задачи, описание и срок выполнения. Создать прокси-класс TaskProxy, который позволяет изменять описание задачи и отображать её информацию только пользователям с определенными правами доступа.

Задача 15. Создать класс Computer, представляющий компьютер с полями марка, модель и год выпуска. Реализовать прокси-класс ComputerProxy, который ограничивает доступ к методу изменения модели и предоставления информации о компьютере, предоставляя доступ только администраторам системы.

Задача 16. Определить класс Book, представляющий книгу с полями название, автор и ISBN. Реализовать прокси-класс BookProxy, который контролирует доступ к изменению автора и отображению информации о книге, предоставляя доступ только пользователям с правами "редактора".

Задача 17. Создать класс CurrencyRate, представляющий курс валют с полями название валюты, код валюты и обменный курс. Реализовать прокси-класс CurrencyRateProxy, который ограничивает возможность обновления обменного курса, предоставляя доступ только определённым пользователям.

Задача 18. Определить класс Client, представляющий клиента с полями имя, идентификационный номер и баланс. Реализовать прокси-класс ClientProxy, который контролирует доступ к изменению баланса и отображению информации о клиенте, предоставляя доступ только сотрудникам с определёнными правами.

Задача 19. Создать класс Account, представляющий счёт с полями номер счёта и баланс. Реализовать прокси-класс AccountProxy, который контролирует операции по пополнению и списанию средств, предоставляя доступ только пользователям с правами "кассира".

Задача 20. Определить класс Company, представляющий фирму с полями название и адрес. Реализовать прокси-класс CompanyProxy, который контролирует доступ к изменению адреса фирмы и отображению информации о ней, предоставляя доступ только руководству компании.

Задача 21. Создать класс Employee, представляющий работника с полями имя, должность и зарплата. Реализовать прокси-класс EmployeeProxy, который контролирует изменение должности и отображение информации о зарплате, предоставляя доступ только отделу кадров.

Задача 22. Определить класс Movie, представляющий фильм с полями название, режиссёр и год выпуска. Реализовать прокси-класс MovieProxy, который контролирует доступ к изменению режиссёра и отображению информации о фильме, предоставляя доступ только авторизованным пользователям.

Задача 23. Создать класс University, представляющий университет с полями название и количество студентов. Реализовать прокси-класс UniversityProxy, который контролирует доступ к добавлению студентов и отображению информации о университете, предоставляя доступ только администрации.

Задача 24. Определить класс Payment, представляющий платёж с полями сумма и дата. Реализовать прокси-класс PaymentProxy, который контролирует изменение суммы и отображение информации о платеже, предоставляя доступ только бухгалтерии.

Задача 25. Создать класс Invoice, представляющий счёт на оплату с полями номер счёта и сумма. Реализовать прокси-класс InvoiceProxy, который контролирует доступ к изменению суммы и отображению информации о счёте, предоставляя доступ только пользователям с определёнными правами.

Задача 26. Определить класс Recipe, представляющий рецепт с полями название, ингредиенты и время приготовления. Реализовать прокси-класс RecipeProxy, который контролирует возможность изменения времени приготовления и отображения рецепта, предоставляя доступ только шеф-повару.

Задача 27. Создать класс Gadget, представляющий гаджет с полями название, тип и цена. Реализовать прокси-класс GadgetProxy, который контролирует доступ к изменению цены и отображению информации о гаджете, предоставляя доступ только сотрудникам с правами "менеджера".

Задача 28. Определить класс Transaction, представляющий финансовую операцию с полями тип операции, сумма и дата. Реализовать прокси-класс TransactionProxy, который контролирует доступ к изменению суммы операции и предоставлению информации о ней, предоставляя доступ только финансовому отделу.

Задача 29. Создать класс House, представляющий дом с полями адрес, количество комнат и площадь. Реализовать прокси-класс HouseProxy, который контролирует доступ к изменению площади и отображению информации о доме, предоставляя доступ только владельцу дома.

Задача 30. Определить класс Project, представляющий проект с полями название, описание и дата начала. Реализовать прокси-класс ProjectProxy, который контролирует доступ к изменению описания проекта и отображению его информации, предоставляя доступ только руководителю проекта.

Задача 31. Создать класс Building, представляющий здание с полями адрес, этажи и год постройки. Реализовать прокси-класс BuildingProxy, который контролирует доступ к изменению этажности здания и отображению информации о нём, предоставляя доступ только инженерам.

Задача 32. Определить класс TrainingCourse, представляющий курс обучения с полями название, продолжительность и стоимость. Реализовать прокси-класс TrainingCourseProxy, который контролирует доступ к изменению стоимости курса и предоставлению информации о нём, предоставляя доступ только организаторам курса.

Задача 33. Создать класс System, представляющий систему с полями название и версия. Реализовать прокси-класс SystemProxy, который контролирует доступ к изменению версии системы и отображению информации о ней, предоставляя доступ только администраторам системы.

Задача 34. Определить класс Rental, представляющий аренду с полями объект аренды, дата начала и дата окончания. Реализовать прокси-класс RentalProxy, который контролирует доступ к изменению дат аренды и отображению информации о аренде, предоставляя доступ только арендатору.

Задача 35. Создать класс Map, представляющий карту с полями название и масштаб. Реализовать прокси-класс MapProxy, который контролирует доступ к изменению масштаба карты и отображению информации о ней, предоставляя доступ только пользователям с определёнными правами.

## 4. Паттерн Bridge

Пример решений задач

Задача 1. Создать класс Person, представляющий человека, с абстрактным методом getProfession(). Создайте подклассы Teacher и Doctor, каждый из которых реализует метод getProfession(), возвращая соответствующую профессию. Реализуйте паттерн Bridge для связи профессии с типом занятости (например, полная занятость, частичная занятость).

|  |
| --- |
| С++ |
| #include <iostream>  #include <string>  // Абстрактный класс для типа занятости  class EmploymentType {  public:  virtual std::string getType() = 0; // Чисто виртуальный метод для получения типа занятости  };  // Конкретная реализация для полной занятости  class FullTime : public EmploymentType {  public:  std::string getType() override {  return "Полная занятость"; // Возвращает тип занятости  }  };  // Конкретная реализация для частичной занятости  class PartTime : public EmploymentType {  public:  std::string getType() override {  return "Частичная занятость"; // Возвращает тип занятости  }  };  // Абстрактный класс для человека  class Person {  protected:  EmploymentType\* employmentType; // Указатель на тип занятости  public:  Person(EmploymentType\* empType) : employmentType(empType) {} // Конструктор инициализирует тип занятости  virtual std::string getProfession() = 0; // Чисто виртуальный метод для получения профессии  void showEmployment() {  std::cout << "Тип занятости: " << employmentType->getType() << std::endl; // Вывод типа занятости  }  };  // Конкретная реализация профессии "Учитель"  class Teacher : public Person {  public:  Teacher(EmploymentType\* empType) : Person(empType) {} // Конструктор вызывает базовый  std::string getProfession() override {  return "Учитель"; // Возвращает профессию  }  };  // Конкретная реализация профессии "Доктор"  class Doctor : public Person {  public:  Doctor(EmploymentType\* empType) : Person(empType) {} // Конструктор вызывает базовый  std::string getProfession() override {  return "Доктор"; // Возвращает профессию  }  };  int main() {  EmploymentType\* fullTime = new FullTime(); // Создание объекта полной занятости  EmploymentType\* partTime = new PartTime(); // Создание объекта частичной занятости  Person\* teacher = new Teacher(fullTime); // Создание учителя с полной занятостью  Person\* doctor = new Doctor(partTime); // Создание доктора с частичной занятостью  std::cout << "Профессия: " << teacher->getProfession() << std::endl; // Вывод профессии учителя  teacher->showEmployment(); // Вывод информации о типе занятости  std::cout << "Профессия: " << doctor->getProfession() << std::endl; // Вывод профессии доктора  doctor->showEmployment(); // Вывод информации о типе занятости  // Освобождение памяти  delete teacher;  delete doctor;  delete fullTime;  delete partTime;  return 0; // Завершение программы  } |
| C# |
| using System;  // Абстрактный класс для типа занятости  public abstract class EmploymentType  {  public abstract string GetType(); // Чисто виртуальный метод  }  // Конкретная реализация для полной занятости  public class FullTime : EmploymentType  {  public override string GetType() => "Полная занятость"; // Возвращает тип занятости  }  // Конкретная реализация для частичной занятости  public class PartTime : EmploymentType  {  public override string GetType() => "Частичная занятость"; // Возвращает тип занятости  }  // Абстрактный класс для человека  public abstract class Person  {  protected EmploymentType employmentType; // Связь с типом занятости  protected Person(EmploymentType empType)  {  employmentType = empType; // Инициализация типа занятости  }  public abstract string GetProfession(); // Чисто виртуальный метод для получения профессии  public void ShowEmployment()  {  Console.WriteLine($"Тип занятости: {employmentType.GetType()}"); // Вывод типа занятости  }  }  // Конкретная реализация профессии "Учитель"  public class Teacher : Person  {  public Teacher(EmploymentType empType) : base(empType) { }  public override string GetProfession() => "Учитель"; // Возвращает профессию  }  // Конкретная реализация профессии "Доктор"  public class Doctor : Person  {  public Doctor(EmploymentType empType) : base(empType) { }  public override string GetProfession() => "Доктор"; // Возвращает профессию  }  // Главный класс программы  class Program  {  static void Main()  {  EmploymentType fullTime = new FullTime(); // Создание типа занятости  EmploymentType partTime = new PartTime(); // Создание типа занятости  Person teacher = new Teacher(fullTime); // Создание учителя с полной занятостью  Person doctor = new Doctor(partTime); // Создание доктора с частичной занятостью  Console.WriteLine($"Профессия: {teacher.GetProfession()}");  teacher.ShowEmployment(); // Вывод информации о профессии и типе занятости  Console.WriteLine($"Профессия: {doctor.GetProfession()}");  doctor.ShowEmployment(); // Вывод информации о профессии и типе занятости  }  } |
| Java |
| // Абстрактный класс для типа занятости  abstract class EmploymentType {  public abstract String getType(); // Чисто виртуальный метод  }  // Конкретная реализация для полной занятости  class FullTime extends EmploymentType {  public String getType() {  return "Полная занятость"; // Возвращает тип занятости  }  }  // Конкретная реализация для частичной занятости  class PartTime extends EmploymentType {  public String getType() {  return "Частичная занятость"; // Возвращает тип занятости  }  }  // Абстрактный класс для человека  abstract class Person {  protected EmploymentType employmentType; // Связь с типом занятости  protected Person(EmploymentType empType) {  this.employmentType = empType; // Инициализация типа занятости  }  public abstract String getProfession(); // Чисто виртуальный метод для получения профессии  public void showEmployment() {  System.out.println("Тип занятости: " + employmentType.getType()); // Вывод типа занятости  }  }  // Конкретная реализация профессии "Учитель"  class Teacher extends Person {  public Teacher(EmploymentType empType) {  super(empType); // Инициализация базового класса  }  public String getProfession() {  return "Учитель"; // Возвращает профессию  }  }  // Конкретная реализация профессии "Доктор"  class Doctor extends Person {  public Doctor(EmploymentType empType) {  super(empType); // Инициализация базового класса  }  public String getProfession() {  return "Доктор"; // Возвращает профессию  }  }  // Главный класс программы  public class Main {  public static void main(String[] args) {  EmploymentType fullTime = new FullTime(); // Создание типа занятости  EmploymentType partTime = new PartTime(); // Создание типа занятости  Person teacher = new Teacher(fullTime); // Создание учителя с полной занятостью  Person doctor = new Doctor(partTime); // Создание доктора с частичной занятостью  System.out.println("Профессия: " + teacher.getProfession());  teacher.showEmployment(); // Вывод информации о профессии и типе занятости  System.out.println("Профессия: " + doctor.getProfession());  doctor.showEmployment(); // Вывод информации о профессии и типе занятости  }  } |
| Python |
| # Абстрактный класс для типа занятости  class EmploymentType:  def get\_type(self):  raise NotImplementedError("Метод должен быть реализован в подклассе.") # Исключение для абстрактного метода  # Конкретная реализация для полной занятости  class FullTime(EmploymentType):  def get\_type(self):  return "Полная занятость" # Возвращает тип занятости  # Конкретная реализация для частичной занятости  class PartTime(EmploymentType):  def get\_type(self):  return "Частичная занятость" # Возвращает тип занятости  # Абстрактный класс для человека  class Person:  def \_\_init\_\_(self, employment\_type):  self.employment\_type = employment\_type # Инициализация типа занятости  def get\_profession(self):  raise NotImplementedError("Метод должен быть реализован в подклассе.") # Исключение для абстрактного метода  def show\_employment(self):  print(f"Тип занятости: {self.employment\_type.get\_type()}") # Вывод типа занятости  # Конкретная реализация профессии "Учитель"  class Teacher(Person):  def get\_profession(self):  return "Учитель" # Возвращает профессию  # Конкретная реализация профессии "Доктор"  class Doctor(Person):  def get\_profession(self):  return "Доктор" # Возвращает профессию  # Главная программа  if \_\_name\_\_ == "\_\_main\_\_":  full\_time = FullTime() # Создание типа занятости  part\_time = PartTime() # Создание типа занятости  teacher = Teacher(full\_time) # Создание учителя с полной занятостью  doctor = Doctor(part\_time) # Создание доктора с частичной занятостью  print(f"Профессия: {teacher.get\_profession()}")  teacher.show\_employment() # Вывод информации о профессии и типе занятости  print(f"Профессия: {doctor.get\_profession()}")  doctor.show\_employment() # Вывод информации о профессии и типе занятости |

### Задачи для самостоятельной работы

Задача 2. Создать абстрактный класс Document с методами print() и save(). Реализуйте классы TextDocument и Spreadsheet, которые наследуют Document и реализуют указанные методы. Используйте паттерн Bridge для разделения типа документа и способа его сохранения (например, сохранение в файл или в базу данных).

Задача 3. Определить абстрактный класс Product, который имеет метод getDescription(). Создайте подклассы Electronics и Clothing, которые реализуют метод getDescription(). Используйте паттерн Bridge для связывания класса товара с различными типами упаковки (например, коробка, пакет).

Задача 4. Создать абстрактный класс CartItem с методами calculateTotalPrice() и displayItemDetails(). Реализуйте подклассы PhysicalItem и DigitalItem, которые наследуют CartItem и реализуют указанные методы. Примените паттерн Bridge для отделения способа расчёта общей стоимости от типа товара.

Задача 5. Определить абстрактный класс AbstractStack с методами push(), pop() и isEmpty(). Реализуйте подклассы ArrayStack и LinkedStack, которые наследуют AbstractStack и используют разные структуры данных. Используйте паттерн Bridge для отделения операций над стеком от его внутренней структуры.

Задача 6. Создать абстрактный класс AbstractQueue с методами enqueue(), dequeue() и isEmpty(). Реализуйте подклассы CircularQueue и PriorityQueue, каждый из которых наследует AbstractQueue и использует разные алгоритмы для реализации очереди. Примените паттерн Bridge для разделения способа работы с очередью и её внутренней структуры.

Задача 7. Определить абстрактный класс Shape с методом draw(). Создайте подклассы Circle и Square, которые реализуют метод draw(). Примените паттерн Bridge для разделения способа рисования фигуры и её типа (например, рисование на экране или на бумаге).

Задача 8. Создать абстрактный класс Vehicle с методами startEngine() и stopEngine(). Реализуйте подклассы Car и Motorcycle, каждый из которых наследует Vehicle и реализует указанные методы. Используйте паттерн Bridge для отделения способа запуска двигателя от типа транспортного средства.

Задача 9. Определить абстрактный класс Event с методами schedule() и cancel(). Создайте подклассы Concert и Conference, которые реализуют методы schedule() и cancel(). Используйте паттерн Bridge для отделения типа события и его способа планирования.

Задача 10. Создать абстрактный класс Course с методами enroll() и drop(). Реализуйте подклассы OnlineCourse и OnsiteCourse, которые наследуют Course и реализуют указанные методы. Используйте паттерн Bridge для разделения типа курса и способа его прохождения (онлайн или офлайн).

Задача 11. Определить абстрактный класс LeaveRequest с методами approve() и reject(). Создайте подклассы SickLeave и VacationLeave, которые реализуют методы approve() и reject(). Примените паттерн Bridge для отделения типа заявки на отпуск от способа её обработки.

Задача 12. Создать абстрактный класс Profile с методами updateEmail() и displayInfo(). Реализуйте подклассы UserProfile и AdminProfile, которые наследуют Profile и реализуют указанные методы. Примените паттерн Bridge для разделения типа профиля и способа его обновления.

Задача 13. Определить абстрактный класс Contact с методами call() и sendEmail(). Создайте подклассы PersonalContact и BusinessContact, которые реализуют методы call() и sendEmail(). Примените паттерн Bridge для разделения типа контакта и способа взаимодействия с ним.

Задача 14. Создать абстрактный класс Task с методами complete() и postpone(). Реализуйте подклассы PersonalTask и WorkTask, которые наследуют Task и реализуют указанные методы. Примените паттерн Bridge для отделения типа задачи от способа её выполнения.

Задача 15. Определить абстрактный класс Computer с методами start() и shutdown(). Создайте подклассы Desktop и Laptop, которые наследуют Computer и реализуют указанные методы. Используйте паттерн Bridge для разделения типа компьютера и способа его запуска.

Задача 16. Создать абстрактный класс Book с методами open() и read(). Реализуйте подклассы EBook и PrintedBook, которые наследуют Book и реализуют указанные методы. Используйте паттерн Bridge для отделения типа книги и способа её чтения.

Задача 17. Определить абстрактный класс CurrencyRate с методами convertTo() и updateRate(). Создайте подклассы USDollarRate и EuroRate, которые наследуют CurrencyRate и реализуют указанные методы. Используйте паттерн Bridge для разделения типа валюты и способа конвертации.

Задача 18. Создать абстрактный класс Client с методами makePurchase() и getBalance(). Реализуйте подклассы RegularClient и VIPClient, которые наследуют Client и реализуют указанные методы. Примените паттерн Bridge для отделения типа клиента и способа обслуживания.

Задача 19. Определить абстрактный класс Account с методами deposit() и withdraw(). Создайте подклассы SavingsAccount и CheckingAccount, которые наследуют Account и реализуют указанные методы. Используйте паттерн Bridge для отделения типа счёта и способа управления средствами.

Задача 20. Создать абстрактный класс Company с методами hireEmployee() и fireEmployee(). Реализуйте подклассы TechCompany и ManufacturingCompany, которые наследуют Company и реализуют указанные методы. Используйте паттерн Bridge для отделения типа компании и способа управления персоналом.

Задача 21. Определить абстрактный класс Employee с методами work() и takeLeave(). Создайте подклассы Manager и Engineer, которые наследуют Employee и реализуют указанные методы. Используйте паттерн Bridge для отделения типа работника и способа выполнения его обязанностей.

Задача 22. Создать абстрактный класс Movie с методами play() и pause(). Реализуйте подклассы ActionMovie и DramaMovie, которые наследуют Movie и реализуют указанные методы. Примените паттерн Bridge для отделения типа фильма и способа его воспроизведения.

Задача 23. Определить абстрактный класс University с методами enrollStudent() и graduateStudent(). Создайте подклассы PublicUniversity и PrivateUniversity, которые наследуют University и реализуют указанные методы. Примените паттерн Bridge для отделения типа университета и способа обучения студентов.

Задача 24. Создать абстрактный класс Payment с методами processPayment() и refund(). Реализуйте подклассы CreditCardPayment и PayPalPayment, которые наследуют Payment и реализуют указанные методы. Используйте паттерн Bridge для отделения типа платежа и способа его обработки.

Задача 25. Определить абстрактный класс Invoice с методами generate() и send(). Создайте подклассы ElectronicInvoice и PaperInvoice, которые наследуют Invoice и реализуют указанные методы. Используйте паттерн Bridge для разделения типа счёта и способа его отправки.

Задача 26. Создать абстрактный класс Recipe с методами prepare() и serve(). Реализуйте подклассы VegetarianRecipe и MeatRecipe, которые наследуют Recipe и реализуют указанные методы. Примените паттерн Bridge для отделения типа рецепта и способа его приготовления.

Задача 27. Определить абстрактный класс Gadget с методами turnOn() и turnOff(). Создайте подклассы Smartphone и Smartwatch, которые наследуют Gadget и реализуют указанные методы. Используйте паттерн Bridge для отделения типа гаджета и способа его использования.

Задача 28. Создать абстрактный класс Transaction с методами execute() и cancel(). Реализуйте подклассы BankTransfer и CryptoTransfer, которые наследуют Transaction и реализуют указанные методы. Используйте паттерн Bridge для разделения типа операции и способа её выполнения.

Задача 29. Определить абстрактный класс House с методами build() и renovate(). Создайте подклассы WoodenHouse и BrickHouse, которые наследуют House и реализуют указанные методы. Примените паттерн Bridge для отделения типа дома и способа его строительства.

Задача 30. Создать абстрактный класс Project с методами start() и finish(). Реализуйте подклассы SoftwareProject и ConstructionProject, которые наследуют Project и реализуют указанные методы. Используйте паттерн Bridge для отделения типа проекта и способа его управления.

Задача 31. Определить абстрактный класс Building с методами construct() и demolish(). Создайте подклассы ResidentialBuilding и CommercialBuilding, которые наследуют Building и реализуют указанные методы. Используйте паттерн Bridge для разделения типа здания и способа его строительства.

Задача 32. Создать абстрактный класс TrainingCourse с методами enrollStudent() и completeCourse(). Реализуйте подклассы OnlineTrainingCourse и ClassroomTrainingCourse, которые наследуют TrainingCourse и реализуют указанные методы. Используйте паттерн Bridge для отделения типа курса и способа его проведения.

Задача 33. Определить абстрактный класс System с методами install() и uninstall(). Создайте подклассы OperatingSystem и DatabaseSystem, которые наследуют System и реализуют указанные методы. Примените паттерн Bridge для отделения типа системы и способа её установки.

Задача 34. Создать абстрактный класс Rental с методами startRental() и endRental(). Реализуйте подклассы CarRental и ApartmentRental, которые наследуют Rental и реализуют указанные методы. Используйте паттерн Bridge для отделения типа аренды и способа её управления.

Задача 35. Определить абстрактный класс Map с методами display() и zoom(). Создайте подклассы RoadMap и SatelliteMap, которые наследуют Map и реализуют указанные методы. Используйте паттерн Bridge для отделения типа карты и способа её отображения.

## 5. Паттерн Decorator

Пример решений задач

Задача 1. Определите класс Person с базовыми методами для получения имени и возраста. Создайте декоратор, который добавляет функциональность для изменения адреса человека и отображения этой информации.

|  |
| --- |
| С++ |
| #include <iostream>  #include <string>  // Базовый класс Person  class Person {  public:  virtual std::string getName() const = 0;  virtual int getAge() const = 0;  virtual ~Person() {}  };  // Конкретная реализация класса Person  class ConcretePerson : public Person {  private:  std::string name;  int age;  public:  ConcretePerson(const std::string& name, int age) : name(name), age(age) {}  std::string getName() const override {  return name;  }  int getAge() const override {  return age;  }  };  // Декоратор для добавления функциональности  class AddressDecorator : public Person {  private:  Person\* person;  std::string address;  public:  AddressDecorator(Person\* p, const std::string& addr) : person(p), address(addr) {}  std::string getName() const override {  return person->getName();  }  int getAge() const override {  return person->getAge();  }  void displayAddress() const {  std::cout << "Address: " << address << std::endl;  }  };  int main() {  Person\* person = new ConcretePerson("Alice", 30);  AddressDecorator decoratedPerson(person, "123 Main St");  std::cout << "Name: " << decoratedPerson.getName() << std::endl;  std::cout << "Age: " << decoratedPerson.getAge() << std::endl;  decoratedPerson.displayAddress();  delete person;  return 0;  } |
| C# |
| using System;  // Базовый класс Person  public abstract class Person  {  public abstract string GetName();  public abstract int GetAge();  }  // Конкретная реализация класса Person  public class ConcretePerson : Person  {  private string name;  private int age;  public ConcretePerson(string name, int age)  {  this.name = name;  this.age = age;  }  public override string GetName()  {  return name;  }  public override int GetAge()  {  return age;  }  }  // Декоратор для добавления функциональности  public class AddressDecorator : Person  {  private Person person;  private string address;  public AddressDecorator(Person person, string address)  {  this.person = person;  this.address = address;  }  public override string GetName()  {  return person.GetName();  }  public override int GetAge()  {  return person.GetAge();  }  public void DisplayAddress()  {  Console.WriteLine($"Address: {address}");  }  }  class Program  {  static void Main()  {  Person person = new ConcretePerson("Alice", 30);  AddressDecorator decoratedPerson = new AddressDecorator(person, "123 Main St");  Console.WriteLine($"Name: {decoratedPerson.GetName()}");  Console.WriteLine($"Age: {decoratedPerson.GetAge()}");  decoratedPerson.DisplayAddress();  }  } |
| Java |
| // Базовый класс Person  abstract class Person {  public abstract String getName();  public abstract int getAge();  }  // Конкретная реализация класса Person  class ConcretePerson extends Person {  private String name;  private int age;  public ConcretePerson(String name, int age) {  this.name = name;  this.age = age;  }  @Override  public String getName() {  return name;  }  @Override  public int getAge() {  return age;  }  }  // Декоратор для добавления функциональности  class AddressDecorator extends Person {  private Person person;  private String address;  public AddressDecorator(Person person, String address) {  this.person = person;  this.address = address;  }  @Override  public String getName() {  return person.getName();  }  @Override  public int getAge() {  return person.getAge();  }  public void displayAddress() {  System.out.println("Address: " + address);  }  }  public class Main {  public static void main(String[] args) {  Person person = new ConcretePerson("Alice", 30);  AddressDecorator decoratedPerson = new AddressDecorator(person, "123 Main St");  System.out.println("Name: " + decoratedPerson.getName());  System.out.println("Age: " + decoratedPerson.getAge());  decoratedPerson.displayAddress();  }  } |
| Python |
| # Базовый класс Person  class Person:  def get\_name(self):  raise NotImplementedError  def get\_age(self):  raise NotImplementedError  # Конкретная реализация класса Person  class ConcretePerson(Person):  def \_\_init\_\_(self, name, age):  self.name = name  self.age = age  def get\_name(self):  return self.name  def get\_age(self):  return self.age  # Декоратор для добавления функциональности  class AddressDecorator(Person):  def \_\_init\_\_(self, person, address):  self.person = person  self.address = address  def get\_name(self):  return self.person.get\_name()  def get\_age(self):  return self.person.get\_age()  def display\_address(self):  print(f"Address: {self.address}")  # Пример использования  if \_\_name\_\_ == "\_\_main\_\_":  person = ConcretePerson("Alice", 30)  decorated\_person = AddressDecorator(person, "123 Main St")  print(f"Name: {decorated\_person.get\_name()}")  print(f"Age: {decorated\_person.get\_age()}")  decorated\_person.display\_address() |

### Задачи для самостоятельной работы

Задача 2. Создайте класс Document, который хранит информацию о документе. Реализуйте декоратор, который добавляет метод для подсчета количества слов в содержимом документа.

Задача 3. Определите класс Product с методами для работы с товаром. Создайте декоратор, который добавляет функциональность для вычисления скидки на товар и отображения итоговой цены.

Задача 4. Создайте класс CartItem с методами для работы с товаром в корзине. Реализуйте декоратор, который добавляет метод для расчета стоимости доставки товара.

Задача 5. Определите класс Stack с базовыми методами стека. Создайте декоратор, который добавляет возможность отслеживания истории операций над стеком.

Задача 6. Создайте класс Queue с основными методами для работы с очередью. Реализуйте декоратор, который добавляет возможность приоритетной обработки элементов.

Задача 7. Определите базовый класс Shape с методами для расчета площади. Создайте декоратор, который добавляет метод для вычисления периметра фигуры.

Задача 8. Создайте класс Vehicle и его подклассы. Реализуйте декоратор, который добавляет функциональность для расчета расхода топлива на определенное расстояние.

Задача 9. Определите класс Event с методами для работы с событием. Создайте декоратор, который добавляет функциональность для расчета оставшегося времени до начала события.

Задача 10. Создайте класс Course с методами для работы с учебным курсом. Реализуйте декоратор, который добавляет возможность установки уровня сложности курса.

Задача 11. Определите класс LeaveRequest с методами для работы с заявкой на отпуск. Создайте декоратор, который добавляет функциональность для расчета количества оставшихся дней отпуска.

Задача 12. Создайте класс Profile с методами для работы с профилем пользователя. Реализуйте декоратор, который добавляет возможность смены пароля и отображения этой информации.

Задача 13. Определите класс Contact с методами для работы с контактной информацией. Создайте декоратор, который добавляет возможность отправки уведомлений по email.

Задача 14. Создайте класс Task с методами для работы с задачей. Реализуйте декоратор, который добавляет возможность установки приоритетов для задач.

Задача 15. Определите класс Computer с методами для работы с компьютером. Создайте декоратор, который добавляет функциональность для проверки совместимости с программным обеспечением.

Задача 16. Создайте класс Book с методами для работы с книгой. Реализуйте декоратор, который добавляет возможность подсчета количества страниц в определенной главе.

Задача 17. Определите класс CurrencyRate с методами для работы с валютой. Создайте декоратор, который добавляет возможность конвертации валюты в другие валюты.

Задача 18. Создайте класс Client с методами для работы с клиентом. Реализуйте декоратор, который добавляет возможность начисления бонусных баллов при совершении покупок.

Задача 19. Определите класс Account с методами для работы со счётом. Создайте декоратор, который добавляет функциональность для автоматического расчета процентов на остаток.

Задача 20. Создайте класс Company с методами для работы с фирмой. Реализуйте декоратор, который добавляет возможность отображения информации о филиалах компании.

Задача 21. Определите класс Employee с методами для работы с сотрудником. Создайте декоратор, который добавляет функциональность для отслеживания истории повышения зарплаты.

Задача 22. Создайте класс Movie с методами для работы с фильмом. Реализуйте декоратор, который добавляет возможность отображения списка наград, полученных фильмом.

Задача 23. Определите класс University с методами для работы с университетом. Создайте декоратор, который добавляет возможность отображения рейтинга университета.

Задача 24. Создайте класс Payment с методами для работы с платежом. Реализуйте декоратор, который добавляет возможность автоматического напоминания о предстоящем платеже.

Задача 25. Определите класс Invoice с методами для работы со счётом на оплату. Создайте декоратор, который добавляет возможность автоматического начисления пени за просрочку оплаты.

Задача 26. Создайте класс Recipe с методами для работы с рецептом. Реализуйте декоратор, который добавляет возможность автоматического расчета калорийности блюда.

Задача 27. Определите класс Gadget с методами для работы с гаджетом. Создайте декоратор, который добавляет функциональность для сравнения характеристик с другими гаджетами.

Задача 28. Создайте класс Transaction с методами для работы с финансовыми операциями. Реализуйте декоратор, который добавляет возможность автоматического расчета налогов с транзакции.

Задача 29. Определите класс House с методами для работы с домом. Создайте декоратор, который добавляет возможность расчета стоимости ремонта дома.

Задача 30. Создайте класс Project с методами для работы с проектом. Реализуйте декоратор, который добавляет возможность отображения диаграммы Ганта для проекта.

Задача 31. Определите класс Building с методами для работы с зданием. Создайте декоратор, который добавляет возможность расчета стоимости аренды здания.

Задача 32. Создайте класс TrainingCourse с методами для работы с учебным курсом. Реализуйте декоратор, который добавляет возможность автоматической выдачи сертификатов по окончании курса.

Задача 33. Определите класс System с методами для работы с системой. Создайте декоратор, который добавляет функциональность для автоматического обновления системы.

Задача 34. Создайте класс Rental с методами для работы с арендой. Реализуйте декоратор, который добавляет возможность автоматического продления срока аренды.

Задача 35. Определите класс Map с методами для работы с картой. Создайте декоратор, который добавляет функциональность для отображения маршрутов и точек интереса.

## 6. Паттерн Facade

Пример решений задач

Задача 1: Разработайте фасад PersonFacade, который будет предоставлять упрощённый интерфейс для управления экземплярами класса Person. Фасад должен включать метод для добавления человека, принимающий информацию о новом человеке, такую как имя (FirstName), фамилия (SecondName), пол (Gender) и возраст (Age), и создающий экземпляр класса Person. Также необходимо реализовать метод для отображения полной информации о человеке, включая все указанные поля: имя, фамилия, пол и возраст. Кроме того, должен быть метод, позволяющий изменять возраст человека с учётом различных условий, таких как проверка прав доступа или бизнес-логики. Важно, чтобы в методах фасада была реализована логика, изменяющая последовательность вызова низкоуровневых методов класса Person в зависимости от заданных условий. Например, если у пользователя есть специальные права, он может изменять информацию о человеке; в противном случае доступно только её просмотр.

|  |
| --- |
| С++ |
| #include <iostream>  #include <string>  class Person {  public:  Person(const std::string& firstName, const std::string& secondName, const std::string& gender, int age)  : firstName(firstName), secondName(secondName), gender(gender), age(age) {}  void displayInfo() const {  std::cout << "Имя: " << firstName << ", Фамилия: " << secondName  << ", Пол: " << gender << ", Возраст: " << age << std::endl;  }  void setAge(int newAge) {  age = newAge;  }  private:  std::string firstName;  std::string secondName;  std::string gender;  int age;  };  class PersonFacade {  public:  PersonFacade() : currentUserHasAccess(false) {}  void setUserAccess(bool hasAccess) {  currentUserHasAccess = hasAccess;  }  void addPerson(const std::string& firstName, const std::string& secondName, const std::string& gender, int age) {  person = new Person(firstName, secondName, gender, age);  }  void displayPersonInfo() const {  if (person) {  person->displayInfo();  } else {  std::cout << "Информация о человеке отсутствует." << std::endl;  }  }  void changePersonAge(int newAge) {  if (currentUserHasAccess) {  if (person) {  person->setAge(newAge);  std::cout << "Возраст изменен на: " << newAge << std::endl;  } else {  std::cout << "Информация о человеке отсутствует." << std::endl;  }  } else {  std::cout << "У вас нет прав для изменения возраста." << std::endl;  }  }  private:  Person\* person = nullptr;  bool currentUserHasAccess;  };  int main() {  PersonFacade facade;  facade.addPerson("Иван", "Иванов", "Мужской", 30);    std::cout << "Информация о человеке:" << std::endl;  facade.displayPersonInfo();  // Попробуем изменить возраст без прав доступа  facade.changePersonAge(35);  // Устанавливаем права доступа и изменяем возраст  facade.setUserAccess(true);  facade.changePersonAge(35);    std::cout << "Обновленная информация о человеке:" << std::endl;  facade.displayPersonInfo();  return 0;  } |
| C# |
| using System;  public class Person  {  private string firstName;  private string secondName;  private string gender;  private int age;  public Person(string firstName, string secondName, string gender, int age)  {  this.firstName = firstName;  this.secondName = secondName;  this.gender = gender;  this.age = age;  }  public void DisplayInfo()  {  Console.WriteLine($"Имя: {firstName}, Фамилия: {secondName}, Пол: {gender}, Возраст: {age}");  }  public void SetAge(int newAge)  {  age = newAge;  }  }  public class PersonFacade  {  private Person person;  private bool currentUserHasAccess;  public void SetUserAccess(bool hasAccess)  {  currentUserHasAccess = hasAccess;  }  public void AddPerson(string firstName, string secondName, string gender, int age)  {  person = new Person(firstName, secondName, gender, age);  }  public void DisplayPersonInfo()  {  if (person != null)  {  person.DisplayInfo();  }  else  {  Console.WriteLine("Информация о человеке отсутствует.");  }  }  public void ChangePersonAge(int newAge)  {  if (currentUserHasAccess)  {  if (person != null)  {  person.SetAge(newAge);  Console.WriteLine($"Возраст изменен на: {newAge}");  }  else  {  Console.WriteLine("Информация о человеке отсутствует.");  }  }  else  {  Console.WriteLine("У вас нет прав для изменения возраста.");  }  }  }  class Program  {  static void Main(string[] args)  {  PersonFacade facade = new PersonFacade();  facade.AddPerson("Иван", "Иванов", "Мужской", 30);  Console.WriteLine("Информация о человеке:");  facade.DisplayPersonInfo();  // Попробуем изменить возраст без прав доступа  facade.ChangePersonAge(35);  // Устанавливаем права доступа и изменяем возраст  facade.SetUserAccess(true);  facade.ChangePersonAge(35);  Console.WriteLine("Обновленная информация о человеке:");  facade.DisplayPersonInfo();  }  } |
| Java |
| class Person {  private String firstName;  private String secondName;  private String gender;  private int age;  public Person(String firstName, String secondName, String gender, int age) {  this.firstName = firstName;  this.secondName = secondName;  this.gender = gender;  this.age = age;  }  public void displayInfo() {  System.out.println("Имя: " + firstName + ", Фамилия: " + secondName +  ", Пол: " + gender + ", Возраст: " + age);  }  public void setAge(int newAge) {  age = newAge;  }  }  class PersonFacade {  private Person person;  private boolean currentUserHasAccess;  public void setUserAccess(boolean hasAccess) {  currentUserHasAccess = hasAccess;  }  public void addPerson(String firstName, String secondName, String gender, int age) {  person = new Person(firstName, secondName, gender, age);  }  public void displayPersonInfo() {  if (person != null) {  person.displayInfo();  } else {  System.out.println("Информация о человеке отсутствует.");  }  }  public void changePersonAge(int newAge) {  if (currentUserHasAccess) {  if (person != null) {  person.setAge(newAge);  System.out.println("Возраст изменен на: " + newAge);  } else {  System.out.println("Информация о человеке отсутствует.");  }  } else {  System.out.println("У вас нет прав для изменения возраста.");  }  }  }  public class Main {  public static void main(String[] args) {  PersonFacade facade = new PersonFacade();  facade.addPerson("Иван", "Иванов", "Мужской", 30);  System.out.println("Информация о человеке:");  facade.displayPersonInfo();  // Попробуем изменить возраст без прав доступа  facade.changePersonAge(35);  // Устанавливаем права доступа и изменяем возраст  facade.setUserAccess(true);  facade.changePersonAge(35);  System.out.println("Обновленная информация о человеке:");  facade.displayPersonInfo();  }  } |
| Python |
| class Person:  def \_\_init\_\_(self, first\_name, second\_name, gender, age):  self.first\_name = first\_name  self.second\_name = second\_name  self.gender = gender  self.age = age  def display\_info(self):  print(f"Имя: {self.first\_name}, Фамилия: {self.second\_name}, Пол: {self.gender}, Возраст: {self.age}")  def set\_age(self, new\_age):  self.age = new\_age  class PersonFacade:  def \_\_init\_\_(self):  self.person = None  self.current\_user\_has\_access = False  def set\_user\_access(self, has\_access):  self.current\_user\_has\_access = has\_access  def add\_person(self, first\_name, second\_name, gender, age):  self.person = Person(first\_name, second\_name, gender, age)  def display\_person\_info(self):  if self.person:  self.person.display\_info()  else:  print("Информация о человеке отсутствует.")  def change\_person\_age(self, new\_age):  if self.current\_user\_has\_access:  if self.person:  self.person.set\_age(new\_age)  print(f"Возраст изменен на: {new\_age}")  else:  print("Информация о человеке отсутствует.")  else:  print("У вас нет прав для изменения возраста.")  if \_\_name\_\_ == "\_\_main\_\_":  facade = PersonFacade()  facade.add\_person("Иван", "Иванов", "Мужской", 30)  print("Информация о человеке:")  facade.display\_person\_info()  # Попробуем изменить возраст без прав доступа  facade.change\_person\_age(35)  # Устанавливаем права доступа и изменяем возраст  facade.set\_user\_access(True)  facade.change\_person\_age(35)  print("Обновленная информация о человеке:")  facade.display\_person\_info() |

### Задачи для самостоятельной работы

Задача 2: Определите фасад DocumentFacade, который объединит операции создания, обновления содержимого и отображения информации о документе. Фасад должен обеспечивать упрощённый интерфейс для работы с классом Document.

Задача 3: Реализуйте фасад ProductFacade, который объединит функции работы с товаром. Включите методы для создания товара, вычисления общей стоимости на складе и обновления цены.

Задача 4: Создайте фасад CartFacade, который предоставит методы для управления корзиной товаров. Фасад должен включать методы для добавления товара, обновления количества и вычисления общей стоимости всех товаров в корзине.

Задача 5: Реализуйте фасад StackFacade, который будет предоставлять упрощённый интерфейс для работы с классом Stack. Включите методы для добавления, удаления элементов, просмотра верхнего элемента и проверки на пустоту.

Задача 6: Определите фасад QueueFacade, который обеспечит простой интерфейс для работы с классом Queue. Фасад должен включать методы для добавления элемента, удаления, просмотра первого элемента и проверки на пустоту.

Задача 7: Создайте фасад ShapeFacade, который упростит работу с различными геометрическими фигурами. Фасад должен предоставлять методы для создания фигур и вычисления их площади.

Задача 8: Реализуйте фасад VehicleFacade, который объединит функции работы с транспортными средствами. Включите методы для создания и управления различными типами транспортных средств, такими как автомобили и велосипеды.

Задача 9: Создайте фасад EventFacade, который обеспечит упрощённый доступ к управлению событиями. Фасад должен включать методы для создания событий, обновления места проведения и отображения информации о событии.

Задача 10: Реализуйте фасад CourseFacade, который упростит работу с учебными курсами. Включите методы для создания курса, обновления количества кредитов и отображения информации о курсе.

Задача 11: Определите фасад LeaveRequestFacade, который предоставит удобный интерфейс для управления заявками на отпуск. Фасад должен включать методы для создания заявки, обновления дат отпуска и отображения информации о заявке.

Задача 12: Создайте фасад ProfileFacade, который обеспечит упрощённый доступ к управлению профилями пользователей. Включите методы для создания профиля, обновления электронной почты и отображения информации о профиле.

Задача 13: Реализуйте фасад ContactFacade, который объединит функции работы с контактами. Включите методы для создания контакта, обновления номера телефона и отображения информации о контакте.

Задача 14: Определите фасад TaskFacade, который обеспечит упрощённый интерфейс для управления задачами. Включите методы для создания задачи, обновления её описания и отображения информации о задаче.

Задача 15: Создайте фасад ComputerFacade, который упростит работу с классом Computer. Включите методы для создания компьютера, обновления его модели и отображения информации о компьютере.

Задача 16: Реализуйте фасад BookFacade, который предоставит удобный интерфейс для управления книгами. Включите методы для создания книги, обновления автора и отображения информации о книге.

Задача 17: Определите фасад CurrencyRateFacade, который обеспечит упрощённый доступ к управлению курсами валют. Включите методы для создания курса валют, обновления обменного курса и отображения информации о валюте.

Задача 18: Создайте фасад ClientFacade, который упростит работу с классом Client. Включите методы для создания клиента, обновления баланса и отображения информации о клиенте.

Задача 19: Реализуйте фасад AccountFacade, который предоставит удобный интерфейс для управления банковскими счетами. Включите методы для создания счета, пополнения, списания средств и отображения информации о счете.

Задача 20: Определите фасад CompanyFacade, который упростит работу с классом Company. Включите методы для создания компании, обновления адреса и отображения информации о компании.

Задача 21: Создайте фасад EmployeeFacade, который предоставит упрощённый доступ к управлению сотрудниками. Включите методы для создания сотрудника, обновления его должности и отображения информации о нём.

Задача 22: Реализуйте фасад MovieFacade, который обеспечит удобный интерфейс для управления фильмами. Включите методы для создания фильма, обновления режиссёра и отображения информации о фильме.

Задача 23: Определите фасад UniversityFacade, который упростит работу с университетами. Включите методы для создания университета, добавления студентов и отображения информации о университете.

Задача 24: Создайте фасад PaymentFacade, который объединит функции работы с платежами. Включите методы для создания платежа, обновления суммы и отображения информации о платеже.

Задача 25: Реализуйте фасад InvoiceFacade, который предоставит удобный интерфейс для управления инвойсами. Включите методы для создания инвойса, обновления суммы и отображения информации о счёте.

Задача 26: Определите фасад RecipeFacade, который упростит работу с рецептами. Включите методы для создания рецепта, обновления времени приготовления и отображения информации о рецепте.

Задача 27: Создайте фасад GadgetFacade, который обеспечит упрощённый доступ к управлению гаджетами. Включите методы для создания гаджета, обновления цены и отображения информации о гаджете.

Задача 28: Реализуйте фасад TransactionFacade, который предоставит удобный интерфейс для работы с финансовыми операциями. Включите методы для создания операции, обновления суммы и отображения информации о транзакции.

Задача 29: Создайте фасад HouseFacade, который упростит управление классом House. Фасад должен включать методы для создания дома, обновления площади и отображения информации о доме.

Задача 30: Определите фасад ProjectFacade, который предоставит упрощённый доступ к управлению проектами. Включите методы для создания проекта, обновления его описания и отображения информации о проекте.

Задача 31: Реализуйте фасад BuildingFacade, который объединит функции работы с классом Building. Включите методы для создания здания, обновления количества этажей и отображения информации о здании.

Задача 32: Создайте фасад TrainingCourseFacade, который упростит работу с учебными курсами. Включите методы для создания курса, обновления продолжительности и отображения информации о курсе.

Задача 33: Определите фасад SystemFacade, который обеспечит упрощённый доступ к управлению системой. Включите методы для создания системы, обновления версии и отображения информации о системе.

Задача 34: Реализуйте фасад RentalFacade, который предоставит удобный интерфейс для управления арендой. Включите методы для создания аренды, обновления дат аренды и отображения информации о ней.

Задача 35: Создайте фасад MapFacade, который объединит функции работы с классом Map. Включите методы для создания карты, обновления масштаба и отображения информации о карте.

## 7. Паттерн Flyweight

Пример решений задач

Задача 1. Определите класс Font, который представляет собой шрифт для отображения текста. Класс должен содержать тип шрифта, размер и цвет. Реализуйте методы для изменения размера и цвета шрифта. Исследуйте, как можно оптимизировать использование различных шрифтов в тексте.

|  |
| --- |
| С++ |
| #include <iostream>  #include <map>  #include <memory>  #include <string>  // Flyweight класс, представляющий шрифт  class Font {  public:  // Конструктор для инициализации шрифта  Font(const std::string& type, int size, const std::string& color)  : type\_(type), size\_(size), color\_(color) {}  // Метод для изменения размера шрифта  void setSize(int size) {  size\_ = size;  }  // Метод для изменения цвета шрифта  void setColor(const std::string& color) {  color\_ = color;  }  // Метод для отображения информации о шрифте  void display() const {  std::cout << "Font Type: " << type\_ << ", Size: " << size\_ << ", Color: " << color\_ << std::endl;  }  private:  std::string type\_; // Тип шрифта  int size\_; // Размер шрифта  std::string color\_; // Цвет шрифта  };  // Фабрика для управления созданием шрифтов  class FontFactory {  public:  // Метод для получения шрифта, если он уже существует, или создания нового  std::shared\_ptr<Font> getFont(const std::string& type, int size, const std::string& color) {  std::string key = type + std::to\_string(size) + color; // Уникальный ключ для шрифта  if (fonts\_.find(key) == fonts\_.end()) {  fonts\_[key] = std::make\_shared<Font>(type, size, color); // Создание нового шрифта  }  return fonts\_[key]; // Возврат существующего шрифта  }  private:  std::map<std::string, std::shared\_ptr<Font>> fonts\_; // Хранение шрифтов  };  int main() {  FontFactory factory; // Создание фабрики шрифтов  auto font1 = factory.getFont("Arial", 12, "Red"); // Получение шрифта  font1->display(); // Отображение шрифта  auto font2 = factory.getFont("Arial", 12, "Red"); // Получение того же шрифта  font2->display(); // Отображение шрифта (должен быть тем же экземпляром, что и font1)  return 0;  } |
| C# |
| using System;  using System.Collections.Generic;  // Flyweight класс, представляющий шрифт  class Font  {  private string type; // Тип шрифта  private int size; // Размер шрифта  private string color; // Цвет шрифта  // Конструктор для инициализации шрифта  public Font(string type, int size, string color)  {  this.type = type;  this.size = size;  this.color = color;  }  // Метод для изменения размера шрифта  public void SetSize(int size)  {  this.size = size;  }  // Метод для изменения цвета шрифта  public void SetColor(string color)  {  this.color = color;  }  // Метод для отображения информации о шрифте  public void Display()  {  Console.WriteLine($"Font Type: {type}, Size: {size}, Color: {color}");  }  }  // Фабрика для управления созданием шрифтов  class FontFactory  {  private Dictionary<string, Font> fonts = new Dictionary<string, Font>(); // Хранение шрифтов  // Метод для получения шрифта  public Font GetFont(string type, int size, string color)  {  string key = $"{type}{size}{color}"; // Уникальный ключ для шрифта  if (!fonts.ContainsKey(key))  {  fonts[key] = new Font(type, size, color); // Создание нового шрифта  }  return fonts[key]; // Возврат существующего шрифта  }  }  class Program  {  static void Main()  {  FontFactory factory = new FontFactory(); // Создание фабрики шрифтов  Font font1 = factory.GetFont("Arial", 12, "Red"); // Получение шрифта  font1.Display(); // Отображение шрифта  Font font2 = factory.GetFont("Arial", 12, "Red"); // Получение того же шрифта  font2.Display(); // Отображение шрифта (должен быть тем же экземпляром, что и font1)  }  } |
| Java |
| import java.util.HashMap;  import java.util.Map;  // Flyweight класс, представляющий шрифт  class Font {  private String type; // Тип шрифта  private int size; // Размер шрифта  private String color; // Цвет шрифта  // Конструктор для инициализации шрифта  public Font(String type, int size, String color) {  this.type = type;  this.size = size;  this.color = color;  }  // Метод для изменения размера шрифта  public void setSize(int size) {  this.size = size;  }  // Метод для изменения цвета шрифта  public void setColor(String color) {  this.color = color;  }  // Метод для отображения информации о шрифте  public void display() {  System.out.println("Font Type: " + type + ", Size: " + size + ", Color: " + color);  }  }  // Фабрика для управления созданием шрифтов  class FontFactory {  private Map<String, Font> fonts = new HashMap<>(); // Хранение шрифтов  // Метод для получения шрифта  public Font getFont(String type, int size, String color) {  String key = type + size + color; // Уникальный ключ для шрифта  if (!fonts.containsKey(key)) {  fonts.put(key, new Font(type, size, color)); // Создание нового шрифта  }  return fonts.get(key); // Возврат существующего шрифта  }  }  public class Main {  public static void main(String[] args) {  FontFactory factory = new FontFactory(); // Создание фабрики шрифтов  Font font1 = factory.getFont("Arial", 12, "Red"); // Получение шрифта  font1.display(); // Отображение шрифта  Font font2 = factory.getFont("Arial", 12, "Red"); // Получение того же шрифта  font2.display(); // Отображение шрифта (должен быть тем же экземпляром, что и font1)  }  } |
| Python |
| class Font:  def \_\_init\_\_(self, font\_type, size, color):  self.font\_type = font\_type # Тип шрифта  self.size = size # Размер шрифта  self.color = color # Цвет шрифта  def set\_size(self, size):  self.size = size # Изменение размера шрифта  def set\_color(self, color):  self.color = color # Изменение цвета шрифта  def display(self):  print(f"Font Type: {self.font\_type}, Size: {self.size}, Color: {self.color}") # Отображение информации о шрифте  class FontFactory:  def \_\_init\_\_(self):  self.fonts = {} # Хранение шрифтов  def get\_font(self, font\_type, size, color):  key = (font\_type, size, color) # Уникальный ключ для шрифта  if key not in self.fonts:  self.fonts[key] = Font(font\_type, size, color) # Создание нового шрифта  return self.fonts[key] # Возврат существующего шрифта  if \_\_name\_\_ == "\_\_main\_\_":  factory = FontFactory() # Создание фабрики шрифтов  font1 = factory.get\_font("Arial", 12, "Red") # Получение шрифта  font1.display() # Отображение шрифта  font2 = factory.get\_font("Arial", 12, "Red") # Получение того же шрифта  font2.display() # Отображение шрифта (должен быть тем же экземпляром, что и font1) |

### Задачи для самостоятельной работы

Задача 2. Создайте класс GraphicObject, представляющий графический элемент, например, линию или круг. Реализуйте методы для установки и изменения координат и цвета графического объекта. Рассмотрите, как использовать паттерн Flyweight для оптимизации использования различных графических объектов на экране.

Задача 3. Определите класс TextElement, представляющий текстовый элемент в документе. Реализуйте методы для изменения текста и отображения элемента. Изучите, как можно использовать паттерн Flyweight для оптимизации использования различных текстовых элементов в документе.

Задача 4. Создайте класс Icon, представляющий иконку в пользовательском интерфейсе. Класс должен содержать поле для изображения и методов для отображения и изменения иконки. Исследуйте, как паттерн Flyweight может помочь в оптимизации отображения множества иконок.

Задача 5. Определите класс Background, представляющий фон для различных элементов пользовательского интерфейса. Реализуйте методы для установки цвета и изображения фона. Рассмотрите, как паттерн Flyweight может быть применен для управления фоновыми изображениями.

Задача 6. Создайте класс ButtonIcon, представляющий значок, используемый на кнопке. Реализуйте методы для изменения и отображения значка. Исследуйте возможности применения паттерна Flyweight для уменьшения количества уникальных значков кнопок.

Задача 7. Определите класс Texture, представляющий текстуру для графических объектов. Реализуйте методы для установки и отображения текстуры. Рассмотрите, как можно использовать Flyweight для управления большими наборами текстур в игре или приложении.

Задача 8. Создайте класс Pixel, представляющий отдельный пиксель на экране. Реализуйте методы для установки и получения цвета пикселя. Исследуйте, как можно применить паттерн Flyweight для эффективного управления цветами пикселей на экране.

Задача 9. Определите класс UIElement, представляющий элемент пользовательского интерфейса, такой как кнопка или текстовое поле. Реализуйте методы для установки и изменения свойств элемента. Изучите, как Flyweight может быть использован для управления множеством элементов интерфейса.

Задача 10. Создайте класс Theme, представляющий тему оформления для приложения. Класс должен содержать поля для различных цветовых схем и шрифтов. Реализуйте методы для применения и изменения темы. Исследуйте, как Flyweight может помочь в управлении различными темами оформления.

Задача 11. Определите класс DesignElement, представляющий элемент дизайна, такой как иконка или рамка. Реализуйте методы для изменения и отображения элемента. Рассмотрите, как паттерн Flyweight может быть применен для оптимизации использования различных элементов дизайна.

Задача 12. Создайте класс ImageBackground, представляющий фон для изображений в приложении. Реализуйте методы для установки и изменения фона. Исследуйте, как можно использовать Flyweight для управления фоновыми изображениями.

Задача 13. Определите класс Frame, представляющий рамку для различных элементов. Реализуйте методы для изменения стиля и цвета рамки. Рассмотрите использование Flyweight для оптимизации различных стилей рамок.

Задача 14. Создайте класс LayoutElement, представляющий элемент лейаута, такой как панель или контейнер. Реализуйте методы для изменения и отображения элементов лейаута. Изучите возможности применения паттерна Flyweight для управления множеством элементов лейаута.

Задача 15. Определите класс TextStyle, представляющий стиль текста в документе. Класс должен содержать поля для шрифта, размера и цвета. Реализуйте методы для изменения стиля текста и применения его к различным частям документа. Рассмотрите применение Flyweight для управления стилями текста.

Задача 16. Создайте класс PrintTemplate, представляющий шаблон для печати документов. Реализуйте методы для изменения элементов шаблона и отображения его. Исследуйте, как Flyweight может помочь в управлении множеством шаблонов печати.

Задача 17. Определите класс InterfaceGraphicElement, представляющий графический элемент в интерфейсе пользователя. Реализуйте методы для установки и изменения свойств элемента. Рассмотрите использование Flyweight для оптимизации отображения графических элементов интерфейса.

Задача 18. Создайте класс Drawing, представляющий рисунок, состоящий из различных фигур и линий. Реализуйте методы для добавления и изменения фигур в рисунке. Изучите, как Flyweight может помочь в управлении множеством рисунков.

Задача 19. Определите класс FileIcon, представляющий иконку файла в файловом менеджере. Реализуйте методы для установки и изменения иконки. Рассмотрите использование Flyweight для оптимизации множества иконок файлов.

Задача 20. Создайте класс Symbol, представляющий символ в тексте или графическом интерфейсе. Реализуйте методы для изменения и отображения символа. Исследуйте, как Flyweight может помочь в управлении множеством символов.

Задача 21. Определите класс Form, представляющий форму для ввода данных в приложении. Реализуйте методы для добавления и изменения полей формы. Рассмотрите использование Flyweight для оптимизации различных форм.

Задача 22. Создайте класс Toolbar, представляющий панель инструментов в приложении. Реализуйте методы для добавления и изменения инструментов на панели. Изучите возможности применения Flyweight для управления множеством инструментов.

Задача 23. Определите класс FormTemplate, представляющий шаблон формы для ввода данных. Реализуйте методы для изменения полей и стилей шаблона. Рассмотрите использование Flyweight для управления множеством шаблонов форм.

Задача 24. Создайте класс GraphicsLibrary, представляющий библиотеку графических объектов. Реализуйте методы для добавления и изменения объектов в библиотеке. Исследуйте, как Flyweight может помочь в управлении большим количеством графических объектов.

Задача 25. Определите класс Animation, представляющий анимацию для графических объектов. Реализуйте методы для изменения кадров и продолжительности анимации. Рассмотрите использование Flyweight для оптимизации анимаций.

Задача 26. Создайте класс ReportTemplate, представляющий шаблон для создания отчетов. Реализуйте методы для изменения элементов шаблона и отображения его. Исследуйте, как Flyweight может помочь в управлении множеством шаблонов отчетов.

Задача 27. Определите класс ViewElement, представляющий элемент представления в графическом интерфейсе. Реализуйте методы для изменения и отображения элемента. Рассмотрите использование Flyweight для оптимизации множества элементов представления.

Задача 28. Создайте класс AppIcon, представляющий иконку приложения. Реализуйте методы для изменения и отображения иконки. Изучите возможности применения Flyweight для управления множеством иконок приложений.

Задача 29. Определите класс PageTemplate, представляющий шаблон страницы для веб-сайта или документа. Реализуйте методы для изменения элементов шаблона и отображения его. Рассмотрите использование Flyweight для управления множеством шаблонов страниц.

Задача 30. Создайте класс UIComponent, представляющий элемент интерфейса, такой как поле ввода или кнопка. Реализуйте методы для изменения и отображения элемента. Исследуйте, как Flyweight может помочь в управлении множеством интерфейсных элементов.

Задача 31. Определите класс Scene, представляющий графическую сцену в приложении или игре. Реализуйте методы для добавления и изменения объектов в сцене. Рассмотрите использование Flyweight для оптимизации графических сцен.

Задача 32. Создайте класс ButtonTemplate, представляющий шаблон кнопки с различными стилями и функциями. Реализуйте методы для изменения и применения шаблона. Изучите возможности применения Flyweight для управления множеством кнопок.

Задача 33. Определите класс InterfaceFunction, представляющий функцию в интерфейсе пользователя, такую как обработчик события. Реализуйте методы для изменения и отображения функции. Рассмотрите использование Flyweight для оптимизации множества функций интерфейса.

Задача 34. Создайте класс IconTemplate, представляющий шаблон иконки с различными стилями и размерами. Реализуйте методы для изменения и применения шаблона. Исследуйте, как Flyweight может помочь в управлении множеством иконок.

Задача 35. Определите класс Picture, представляющий изображение, которое может быть использовано в различных частях приложения. Реализуйте методы для изменения и отображения изображения.

# ГЛАВА 5. ПОВЕДЕНЧЕСКИЕ ПАТТЕРНЫ

## 1. Паттерн Observer

Пример решений задач

Задача 1. Создайте класс TemperatureSensor, который сообщает о изменениях температуры. Реализуйте механизм для уведомления подписчиков об изменении температуры.

|  |
| --- |
| С++ |
| #include <iostream>  #include <vector>  #include <algorithm> // Для std::remove  // Интерфейс наблюдателя  class Observer {  public:  virtual void update(float temperature) = 0; // Метод обновления  };  // Класс наблюдаемого объекта  class TemperatureSensor {  private:  std::vector<Observer\*> observers; // Список подписчиков  float temperature; // Текущая температура  public:  // Добавление наблюдателя  void addObserver(Observer\* observer) {  observers.push\_back(observer);  }  // Удаление наблюдателя  void removeObserver(Observer\* observer) {  observers.erase(std::remove(observers.begin(), observers.end(), observer), observers.end());  }  // Установка новой температуры  void setTemperature(float temp) {  temperature = temp;  notifyObservers(); // Уведомление подписчиков  }  // Уведомление всех наблюдателей  void notifyObservers() {  for (Observer\* observer : observers) {  observer->update(temperature);  }  }  };  // Пример наблюдателя  class Display : public Observer {  public:  // Реализация метода обновления  void update(float temperature) override {  std::cout << "Температура обновлена: " << temperature << "°C" << std::endl;  }  };  int main() {  TemperatureSensor sensor; // Создание сенсора  Display display; // Создание дисплея  sensor.addObserver(&display); // Подписка дисплея на обновления  sensor.setTemperature(25.0f); // Установка температуры  sensor.setTemperature(30.0f); // Изменение температуры  return 0;  } |
| C# |
| using System;  using System.Collections.Generic;  // Интерфейс наблюдателя  public interface IObserver {  void Update(float temperature); // Метод обновления  }  // Класс наблюдаемого объекта  public class TemperatureSensor {  private List<IObserver> observers = new List<IObserver>(); // Список подписчиков  private float temperature; // Текущая температура  // Добавление наблюдателя  public void AddObserver(IObserver observer) {  observers.Add(observer);  }  // Удаление наблюдателя  public void RemoveObserver(IObserver observer) {  observers.Remove(observer);  }  // Установка новой температуры  public void SetTemperature(float temp) {  temperature = temp;  NotifyObservers(); // Уведомление подписчиков  }  // Уведомление всех наблюдателей  private void NotifyObservers() {  foreach (var observer in observers) {  observer.Update(temperature);  }  }  }  // Пример наблюдателя  public class Display : IObserver {  // Реализация метода обновления  public void Update(float temperature) {  Console.WriteLine($"Температура обновлена: {temperature}°C");  }  }  class Program {  static void Main() {  TemperatureSensor sensor = new TemperatureSensor(); // Создание сенсора  Display display = new Display(); // Создание дисплея  sensor.AddObserver(display); // Подписка дисплея на обновления  sensor.SetTemperature(25.0f); // Установка температуры  sensor.SetTemperature(30.0f); // Изменение температуры  }  } |
| Java |
| import java.util.ArrayList;  import java.util.List;  // Интерфейс наблюдателя  interface Observer {  void update(float temperature); // Метод обновления  }  // Класс наблюдаемого объекта  class TemperatureSensor {  private List<Observer> observers = new ArrayList<>(); // Список подписчиков  private float temperature; // Текущая температура  // Добавление наблюдателя  public void addObserver(Observer observer) {  observers.add(observer);  }  // Удаление наблюдателя  public void removeObserver(Observer observer) {  observers.remove(observer);  }  // Установка новой температуры  public void setTemperature(float temp) {  temperature = temp;  notifyObservers(); // Уведомление подписчиков  }  // Уведомление всех наблюдателей  private void notifyObservers() {  for (Observer observer : observers) {  observer.update(temperature);  }  }  }  // Пример наблюдателя  class Display implements Observer {  // Реализация метода обновления  public void update(float temperature) {  System.out.println("Температура обновлена: " + temperature + "°C");  }  }  public class Main {  public static void main(String[] args) {  TemperatureSensor sensor = new TemperatureSensor(); // Создание сенсора  Display display = new Display(); // Создание дисплея  sensor.addObserver(display); // Подписка дисплея на обновления  sensor.setTemperature(25.0f); // Установка температуры  sensor.setTemperature(30.0f); // Изменение температуры  }  } |
| Python |
| from abc import ABC, abstractmethod  # Интерфейс наблюдателя  class Observer(ABC):  @abstractmethod  def update(self, temperature):  pass  # Класс наблюдаемого объекта  class TemperatureSensor:  def \_\_init\_\_(self):  self.observers = [] # Список подписчиков  self.temperature = 0.0 # Текущая температура  # Добавление наблюдателя  def add\_observer(self, observer):  self.observers.append(observer)  # Удаление наблюдателя  def remove\_observer(self, observer):  self.observers.remove(observer)  # Установка новой температуры  def set\_temperature(self, temp):  self.temperature = temp  self.notify\_observers() # Уведомление подписчиков  # Уведомление всех наблюдателей  def notify\_observers(self):  for observer in self.observers:  observer.update(self.temperature)  # Пример наблюдателя  class Display(Observer):  # Реализация метода обновления  def update(self, temperature):  print(f"Температура обновлена: {temperature}°C")  # Пример использования  if \_\_name\_\_ == "\_\_main\_\_":  sensor = TemperatureSensor() # Создание сенсора  display = Display() # Создание дисплея  sensor.add\_observer(display) # Подписка дисплея на обновления  sensor.set\_temperature(25.0) # Установка температуры  sensor.set\_temperature(30.0) # Изменение температуры |

**Обратите внимание:** Обратите внимание: можно добавить несколько наблюдений, чтобы лучше понять суть паттерна 'Наблюдатель'. Вот пример на Java:

|  |
| --- |
| import java.util.ArrayList;  import java.util.List;  // Интерфейс наблюдателя  interface Observer {  void update(float temperature); // Метод обновления  }  // Класс наблюдаемого объекта  class TemperatureSensor {  private List<Observer> observers = new ArrayList<>(); // Список подписчиков  private float temperature; // Текущая температура  // Добавление наблюдателя  public void addObserver(Observer observer) {  observers.add(observer);  }  // Удаление наблюдателя  public void removeObserver(Observer observer) {  observers.remove(observer);  }  // Установка новой температуры  public void setTemperature(float temp) {  this.temperature = temp; // Обновление текущей температуры  notifyObservers(); // Уведомление подписчиков  }  // Уведомление всех наблюдателей  private void notifyObservers() {  for (Observer observer : observers) {  observer.update(temperature); // Вызов метода обновления для каждого наблюдателя  }  }  }  // Пример наблюдателя: Дисплей  class Display implements Observer {  // Реализация метода обновления  @Override  public void update(float temperature) {  System.out.println("Температура обновлена: " + temperature + "°C");  }  }  // Пример второго наблюдателя: Сервис для преобразования в Фаренгейты  class ServiceFahrenheit implements Observer {  // Реализация метода обновления  @Override  public void update(float temperature) {  float tempFahr = (temperature \* 9 / 5) + 32; // Преобразование в Фаренгейты  System.out.println("Температура обновлена: " + tempFahr + "°F");  }  }  // Главный класс  public class Main {  public static void main(String[] args) {  // Создание сенсора и наблюдателей  TemperatureSensor sensor = new TemperatureSensor(); // Создание сенсора  Display display = new Display(); // Создание дисплея  ServiceFahrenheit service = new ServiceFahrenheit(); // Создание сервиса  // Подписка дисплея и сервиса на обновления  sensor.addObserver(display);  sensor.addObserver(service);    // Установка и изменение температуры  sensor.setTemperature(25.0f); // Установка начальной температуры  sensor.setTemperature(30.0f); // Изменение температуры  }  } |

### Задачи для самостоятельной работы

Задача 2. Определите класс WeatherStation, который следит за изменениями погоды. Реализуйте систему уведомлений для метеорологов, когда погода меняется.

Задача 3. Создайте класс StockMarket, который отслеживает изменения курса акций. Реализуйте механизм для уведомления инвесторов о колебаниях курса акций.

Задача 4. Определите класс Order, который обновляет статус заказа. Реализуйте систему уведомлений для клиентов, когда статус их заказа меняется.

Задача 5. Создайте класс Product, который уведомляет подписчиков о изменении цены. Реализуйте механизм оповещения покупателей об изменении цен.

Задача 6. Создайте класс FuelGauge, который отслеживает уровень топлива в автомобиле. Реализуйте уведомления для водителей, когда уровень топлива падает ниже определенного порога.

Задача 7. Определите класс BusSchedule, который управляет расписанием автобусов. Реализуйте механизм уведомления пассажиров об изменениях в расписании.

Задача 8. Создайте класс SecuritySystem, который отслеживает состояние системы безопасности. Реализуйте уведомления для охранников и пользователей при изменении состояния системы.

Задача 9. Определите класс Inventory, который отслеживает состояние запасов в магазине. Реализуйте систему уведомлений для менеджеров о необходимости пополнения запасов.

Задача 10. Создайте класс WebsiteActivity, который отслеживает уровень активности пользователей на сайте. Реализуйте механизм уведомлений для администраторов при значительных изменениях активности.

Задача 11. Определите класс UserProfile, который отслеживает изменения в профиле пользователя. Реализуйте систему уведомлений для социальных сетей при изменении информации о пользователе.

Задача 12. Создайте класс Project, который отслеживает изменения в состоянии проекта. Реализуйте механизм уведомлений для команды проекта при изменении статуса или выполнения задач.

Задача 13. Определите класс Subscription, который управляет подписками на услуги. Реализуйте систему уведомлений для пользователей при изменении статуса их подписки.

Задача 14. Создайте класс TaskManager, который управляет задачами в приложении. Реализуйте механизм уведомлений для пользователей при изменении статуса задач.

Задача 15. Определите класс Equipment, который отслеживает состояние промышленного оборудования. Реализуйте уведомления для технического персонала при сбоях или изменениях в состоянии оборудования.

Задача 16. Создайте класс NewsPublisher, который публикует новости. Реализуйте механизм уведомлений для подписчиков при выходе новых новостей.

Задача 17. Определите класс ProjectTask, который отслеживает статус задач в проекте. Реализуйте систему уведомлений для членов команды при изменении статуса задач.

Задача 18. Создайте класс AppStatistics, который отслеживает ключевые метрики приложения. Реализуйте механизм уведомлений для разработчиков при изменении статистики.

Задача 19. Определите класс Blog, который отслеживает изменения в публикациях. Реализуйте уведомления для подписчиков при обновлении или добавлении новых публикаций.

Задача 20. Создайте класс EventSchedule, который управляет расписанием мероприятий. Реализуйте механизм уведомлений для участников при изменении расписания.

Задача 21. Определите класс BatteryMonitor, который отслеживает уровень заряда батареи устройства. Реализуйте систему уведомлений для пользователя при низком уровне заряда.

Задача 22. Создайте класс RestaurantOrder, который отслеживает состояние заказов в ресторане. Реализуйте уведомления для клиентов при изменении статуса их заказа.

Задача 23. Определите класс PromotionManager, который отслеживает активные предложения и акции. Реализуйте уведомления для клиентов о новых или измененных предложениях.

Задача 24. Создайте класс ServiceSubscription, который управляет подписками на различные сервисы. Реализуйте уведомления для пользователей при изменении статуса их подписки.

Задача 25. Определите класс HealthMonitor, который отслеживает состояние здоровья пациента. Реализуйте уведомления для врачей при значительных изменениях в показателях здоровья.

Задача 26. Создайте класс CRM, который управляет данными о клиентах. Реализуйте механизм уведомлений для сотрудников при обновлении информации о клиентах.

Задача 27. Определите класс ServerMonitor, который отслеживает состояние серверов. Реализуйте уведомления для администраторов при сбоях или изменениях в состоянии серверов.

Задача 28. Создайте класс BookingSystem, который управляет системой бронирования. Реализуйте уведомления для пользователей при изменении статуса их бронирования.

Задача 29. Определите класс CompanyStock, который отслеживает состояние акций компании. Реализуйте систему уведомлений для акционеров при изменении акций.

Задача 30. Создайте класс CloudService, который управляет состоянием облачных сервисов. Реализуйте уведомления для администраторов и пользователей при изменении состояния сервисов.

Задача 31. Определите класс ClassSchedule, который управляет учебным расписанием. Реализуйте уведомления для студентов и преподавателей при изменении расписания.

Задача 32. Создайте класс SocialNetwork, который отслеживает изменения в социальной сети (новости, сообщения). Реализуйте механизм уведомлений для пользователей при новых обновлениях.

Задача 33. Определите класс UserManagement, который управляет информацией о пользователях. Реализуйте уведомления для администраторов при изменении данных пользователей.

Задача 34. Создайте класс StockLevel, который отслеживает уровень запаса в различных магазинах. Реализуйте уведомления для управляющих при достижении критических уровней запаса.

Задача 35. Определите класс OrderManagement, который управляет состоянием заказов в системе электронной коммерции. Реализуйте систему уведомлений для клиентов и сотрудников при изменении статуса заказов.

## 2. Паттерн Strategy

Пример решений задач

Задача 1. Создайте интерфейс PaymentStrategy с методом pay(amount). Реализуйте конкретные стратегии CreditCardPayment и PayPalPayment, которые будут осуществлять платеж разными способами. Реализуйте класс PaymentContext, который использует PaymentStrategy для выполнения платежей.

|  |
| --- |
| С++ |
| #include <iostream>  // Интерфейс стратегии платежа  class PaymentStrategy {  public:  virtual void pay(float amount) = 0; // Метод для выполнения платежа  };  // Конкретная стратегия: Платеж через кредитную карту  class CreditCardPayment : public PaymentStrategy {  public:  void pay(float amount) override {  std::cout << "Оплата " << amount << " с помощью кредитной карты." << std::endl;  }  };  // Конкретная стратегия: Платеж через PayPal  class PayPalPayment : public PaymentStrategy {  public:  void pay(float amount) override {  std::cout << "Оплата " << amount << " через PayPal." << std::endl;  }  };  // Контекст для выполнения платежа  class PaymentContext {  private:  PaymentStrategy\* strategy; // Стратегия платежа  public:  PaymentContext(PaymentStrategy\* strategy) : strategy(strategy) {}  void setStrategy(PaymentStrategy\* strategy) {  this->strategy = strategy; // Установка новой стратегии  }  void executePayment(float amount) {  strategy->pay(amount); // Выполнение платежа  }  };  int main() {  PaymentContext context(new CreditCardPayment()); // Создание контекста с кредитной картой  context.executePayment(100.0f); // Выполнение платежа  context.setStrategy(new PayPalPayment()); // Смена стратегии на PayPal  context.executePayment(150.0f); // Выполнение платежа  return 0;  } |
| C# |
| using System;  // Интерфейс стратегии платежа  public interface IPaymentStrategy {  void Pay(float amount); // Метод для выполнения платежа  }  // Конкретная стратегия: Платеж через кредитную карту  public class CreditCardPayment : IPaymentStrategy {  public void Pay(float amount) {  Console.WriteLine($"Оплата {amount} с помощью кредитной карты.");  }  }  // Конкретная стратегия: Платеж через PayPal  public class PayPalPayment : IPaymentStrategy {  public void Pay(float amount) {  Console.WriteLine($"Оплата {amount} через PayPal.");  }  }  // Контекст для выполнения платежа  public class PaymentContext {  private IPaymentStrategy strategy; // Стратегия платежа  public PaymentContext(IPaymentStrategy strategy) {  this.strategy = strategy; // Установка стратегии  }  public void SetStrategy(IPaymentStrategy strategy) {  this.strategy = strategy; // Установка новой стратегии  }  public void ExecutePayment(float amount) {  strategy.Pay(amount); // Выполнение платежа  }  }  class Program {  static void Main() {  PaymentContext context = new PaymentContext(new CreditCardPayment()); // Создание контекста с кредитной картой  context.ExecutePayment(100.0f); // Выполнение платежа  context.SetStrategy(new PayPalPayment()); // Смена стратегии на PayPal  context.ExecutePayment(150.0f); // Выполнение платежа  }  } |
| Java |
| // Интерфейс стратегии платежа  interface PaymentStrategy {  void pay(float amount); // Метод для выполнения платежа  }  // Конкретная стратегия: Платеж через кредитную карту  class CreditCardPayment implements PaymentStrategy {  public void pay(float amount) {  System.out.println("Оплата " + amount + " с помощью кредитной карты.");  }  }  // Конкретная стратегия: Платеж через PayPal  class PayPalPayment implements PaymentStrategy {  public void pay(float amount) {  System.out.println("Оплата " + amount + " через PayPal.");  }  }  // Контекст для выполнения платежа  class PaymentContext {  private PaymentStrategy strategy; // Стратегия платежа  public PaymentContext(PaymentStrategy strategy) {  this.strategy = strategy; // Установка стратегии  }  public void setStrategy(PaymentStrategy strategy) {  this.strategy = strategy; // Установка новой стратегии  }  public void executePayment(float amount) {  strategy.pay(amount); // Выполнение платежа  }  }  public class Main {  public static void main(String[] args) {  PaymentContext context = new PaymentContext(new CreditCardPayment()); // Создание контекста с кредитной картой  context.executePayment(100.0f); // Выполнение платежа  context.setStrategy(new PayPalPayment()); // Смена стратегии на PayPal  context.executePayment(150.0f); // Выполнение платежа  }  } |
| Python |
| from abc import ABC, abstractmethod  # Интерфейс стратегии платежа  class PaymentStrategy(ABC):  @abstractmethod  def pay(self, amount):  pass  # Конкретная стратегия: Платеж через кредитную карту  class CreditCardPayment(PaymentStrategy):  def pay(self, amount):  print(f"Оплата {amount} с помощью кредитной карты.")  # Конкретная стратегия: Платеж через PayPal  class PayPalPayment(PaymentStrategy):  def pay(self, amount):  print(f"Оплата {amount} через PayPal.")  # Контекст для выполнения платежа  class PaymentContext:  def \_\_init\_\_(self, strategy: PaymentStrategy):  self.strategy = strategy # Установка стратегии  def set\_strategy(self, strategy: PaymentStrategy):  self.strategy = strategy # Установка новой стратегии  def execute\_payment(self, amount):  self.strategy.pay(amount) # Выполнение платежа  # Пример использования  if \_\_name\_\_ == "\_\_main\_\_":  context = PaymentContext(CreditCardPayment()) # Создание контекста с кредитной картой  context.execute\_payment(100.0) # Выполнение платежа  context.set\_strategy(PayPalPayment()) # Смена стратегии на PayPal  context.execute\_payment(150.0) # Выполнение платежа |

### Задачи для самостоятельной работы

Задача 2. Определите интерфейс SortingStrategy с методом sort(list). Реализуйте конкретные стратегии BubbleSort, QuickSort и MergeSort. Реализуйте класс Sorter, который использует SortingStrategy для сортировки списков.

Задача 3. Создайте интерфейс RenderingStrategy с методом render(shape). Реализуйте конкретные стратегии VectorRendering и RasterRendering. Реализуйте класс ShapeRenderer, который использует RenderingStrategy для рендеринга графических объектов.

Задача 4. Определите интерфейс RouteStrategy с методом calculateRoute(start, end). Реализуйте конкретные стратегии ShortestRoute и FastestRoute. Реализуйте класс NavigationSystem, который использует RouteStrategy для расчета маршрутов.

Задача 5. Создайте интерфейс DiscountStrategy с методом applyDiscount(price). Реализуйте конкретные стратегии SeasonalDiscount, ClearanceDiscount и VolumeDiscount. Реализуйте класс Pricing, который использует DiscountStrategy для расчета итоговой цены.

Задача 6. Определите интерфейс UserBehaviorStrategy с методом execute(). Реализуйте конкретные стратегии AggressiveBehavior, PassiveBehavior и NeutralBehavior. Реализуйте класс UserContext, который использует UserBehaviorStrategy для выполнения действий пользователей.

Задача 7. Создайте интерфейс TextRecognitionStrategy с методом recognize(text). Реализуйте конкретные стратегии OCRRecognition и NaturalLanguageProcessing. Реализуйте класс TextRecognizer, который использует TextRecognitionStrategy для распознавания текста.

Задача 8. Определите интерфейс StorageStrategy с методом save(data). Реализуйте конкретные стратегии DatabaseStorage, FileStorage и CloudStorage. Реализуйте класс DataManager, который использует StorageStrategy для сохранения данных.

Задача 9. Создайте интерфейс ClassificationStrategy с методом classify(document). Реализуйте конкретные стратегии ManualClassification и AutomatedClassification. Реализуйте класс DocumentClassifier, который использует ClassificationStrategy для классификации документов.

Задача 10. Определите интерфейс CompressionStrategy с методом compress(data). Реализуйте конкретные стратегии ZIPCompression, RarCompression и GZIPCompression. Реализуйте класс Compressor, который использует CompressionStrategy для сжатия данных.

Задача 11. Создайте интерфейс RiskAssessmentStrategy с методом assessRisk(data). Реализуйте конкретные стратегии QuantitativeRiskAssessment и QualitativeRiskAssessment. Реализуйте класс RiskEvaluator, который использует RiskAssessmentStrategy для оценки рисков.

Задача 12. Определите интерфейс ImageProcessingStrategy с методом process(image). Реализуйте конкретные стратегии GrayscaleProcessing и SepiaProcessing. Реализуйте класс ImageProcessor, который использует ImageProcessingStrategy для обработки изображений.

Задача 13. Создайте интерфейс TaskManagementStrategy с методом manageTasks(tasks). Реализуйте конкретные стратегии SimpleTaskManagement и AdvancedTaskManagement. Реализуйте класс TaskManager, который использует TaskManagementStrategy для управления задачами.

Задача 14. Определите интерфейс FileFormatStrategy с методом parseFile(file). Реализуйте конкретные стратегии XMLFileFormat, JSONFileFormat и CSVFileFormat. Реализуйте класс FileParser, который использует FileFormatStrategy для обработки файлов.

Задача 15. Создайте интерфейс QueryOptimizationStrategy с методом optimize(query). Реализуйте конкретные стратегии IndexOptimization, CachingOptimization и QueryRewritingOptimization. Реализуйте класс QueryOptimizer, который использует QueryOptimizationStrategy для оптимизации запросов.

Задача 16. Определите интерфейс PersonalizationStrategy с методом customize(userPreferences). Реализуйте конкретные стратегии ThemePersonalization, LayoutPersonalization и ContentPersonalization. Реализуйте класс InterfaceCustomizer, который использует PersonalizationStrategy для настройки интерфейса.

Задача 17. Создайте интерфейс SearchStrategy с методом search(query). Реализуйте конкретные стратегии LinearSearch и BinarySearch. Реализуйте класс SearchEngine, который использует SearchStrategy для выполнения поиска.

Задача 18. Определите интерфейс TimeManagementStrategy с методом manageTime(schedule). Реализуйте конкретные стратегии PomodoroTechnique и TimeBlockingTechnique. Реализуйте класс TimeManager, который использует TimeManagementStrategy для управления временем.

Задача 19. Создайте интерфейс AlgorithmStrategy с методом executeAlgorithm(data). Реализуйте конкретные стратегии SortingAlgorithm, SearchingAlgorithm и OptimizationAlgorithm. Реализуйте класс AlgorithmExecutor, который использует AlgorithmStrategy для выполнения задач.

Задача 20. Определите интерфейс DataAccessStrategy с методом fetchData(). Реализуйте конкретные стратегии APIAccess, DatabaseAccess и FileAccess. Реализуйте класс DataAccessor, который использует DataAccessStrategy для получения данных.

Задача 21. Создайте интерфейс NotificationStrategy с методом sendNotification(message). Реализуйте конкретные стратегии EmailNotification, SMSNotification и PushNotification. Реализуйте класс NotificationSender, который использует NotificationStrategy для отправки уведомлений.

Задача 22. Определите интерфейс RequestHandlingStrategy с методом handleRequest(request). Реализуйте конкретные стратегии RESTfulRequestHandler и SOAPRequestHandler. Реализуйте класс RequestHandler, который использует RequestHandlingStrategy для обработки запросов.

Задача 23. Создайте интерфейс OrderProcessingStrategy с методом processOrder(order). Реализуйте конкретные стратегии StandardOrderProcessing и ExpressOrderProcessing. Реализуйте класс OrderProcessor, который использует OrderProcessingStrategy для обработки заказов.

Задача 24. Определите интерфейс AccessControlStrategy с методом grantAccess(user). Реализуйте конкретные стратегии RoleBasedAccessControl и AttributeBasedAccessControl. Реализуйте класс AccessControlManager, который использует AccessControlStrategy для управления доступом.

Задача 25. Создайте интерфейс BackupStrategy с методом backup(data). Реализуйте конкретные стратегии FullBackup, IncrementalBackup и DifferentialBackup. Реализуйте класс BackupManager, который использует BackupStrategy для резервного копирования данных.

Задача 26. Определите интерфейс ResourceManagementStrategy с методом manageResources(resources). Реализуйте конкретные стратегии StaticResourceManagement и DynamicResourceManagement. Реализуйте класс ResourceManager, который использует ResourceManagementStrategy для управления ресурсами.

Задача 27. Создайте интерфейс PerformanceEvaluationStrategy с методом evaluatePerformance(metrics). Реализуйте конкретные стратегии BenchmarkingEvaluation и KPIBasedEvaluation. Реализуйте класс PerformanceEvaluator, который использует PerformanceEvaluationStrategy для оценки эффективности.

Задача 28. Определите интерфейс ReportGenerationStrategy с методом generateReport(data). Реализуйте конкретные стратегии PDFReportGeneration и ExcelReportGeneration. Реализуйте класс ReportGenerator, который использует ReportGenerationStrategy для генерации отчетов.

Задача 29. Создайте интерфейс ExceptionHandlingStrategy с методом handleException(exception). Реализуйте конкретные стратегии LogExceptionHandling и NotifyExceptionHandling. Реализуйте класс ExceptionHandler, который использует ExceptionHandlingStrategy для обработки исключений.

Задача 30. Определите интерфейс LoggingStrategy с методом log(message). Реализуйте конкретные стратегии ConsoleLogging и FileLogging. Реализуйте класс Logger, который использует LoggingStrategy для записи логов.

Задача 31. Создайте интерфейс TransactionStrategy с методом processTransaction(transaction). Реализуйте конкретные стратегии CreditTransaction и DebitTransaction. Реализуйте класс TransactionProcessor, который использует TransactionStrategy для обработки транзакций.

Задача 32. Определите интерфейс PriorityStrategy с методом applyPriority(task). Реализуйте конкретные стратегии HighPriority, MediumPriority и LowPriority. Реализуйте класс TaskScheduler, который использует PriorityStrategy для управления задачами.

Задача 33. Создайте интерфейс DataSourceStrategy с методом getData(). Реализуйте конкретные стратегии APIDataSource, DatabaseDataSource и FileDataSource. Реализуйте класс DataSourceManager, который использует DataSourceStrategy для получения данных.

Задача 34. Определите интерфейс PerformanceMeasurementStrategy с методом measurePerformance(). Реализуйте конкретные стратегии TimeBasedMeasurement и ResourceUsageMeasurement. Реализуйте класс PerformanceMeasurer, который использует PerformanceMeasurementStrategy для измерения производительности.

Задача 35. Создайте интерфейс DataFormatStrategy с методом convert(data). Реализуйте конкретные стратегии XMLFormat, JSONFormat и CSVFormat. Реализуйте класс DataConverter, который использует DataFormatStrategy для преобразования данных.

## 3. Паттерн Command

Пример решений задач

Задача 1. Определите команду, которая будет изменять возраст объекта класса Person. Реализуйте методы для установки нового возраста и выполнения команды.

|  |
| --- |
| С++ |
| #include <iostream>  #include <string>  // Класс Person  class Person {  public:  void setAge(int age) {  this->age = age;  }  void display() {  std::cout << "Age: " << age << std::endl;  }  private:  int age;  };  // Интерфейс команды  class Command {  public:  virtual void execute() = 0;  };  // Команда для изменения возраста  class SetAgeCommand : public Command {  public:  SetAgeCommand(Person\* person, int age) : person(person), age(age) {}  void execute() override {  person->setAge(age);  }  private:  Person\* person;  int age;  };  int main() {  Person person;  SetAgeCommand command(&person, 25);  command.execute(); // Устанавливаем возраст  person.display(); // Выводим возраст  return 0;  } |
| C# |
| using System;  // Класс Person  public class Person {  private int age;  public void SetAge(int age) {  this.age = age;  }  public void Display() {  Console.WriteLine($"Age: {age}");  }  }  // Интерфейс команды  public interface ICommand {  void Execute();  }  // Команда для изменения возраста  public class SetAgeCommand : ICommand {  private Person person;  private int age;  public SetAgeCommand(Person person, int age) {  this.person = person;  this.age = age;  }  public void Execute() {  person.SetAge(age);  }  }  public class Program {  public static void Main() {  Person person = new Person();  SetAgeCommand command = new SetAgeCommand(person, 25);  command.Execute(); // Устанавливаем возраст  person.Display(); // Выводим возраст  }  } |
| Java |
| // Класс Person  class Person {  private int age;  public void setAge(int age) {  this.age = age;  }  public void display() {  System.out.println("Age: " + age);  }  }  // Интерфейс команды  interface Command {  void execute();  }  // Команда для изменения возраста  class SetAgeCommand implements Command {  private Person person;  private int age;  public SetAgeCommand(Person person, int age) {  this.person = person;  this.age = age;  }  public void execute() {  person.setAge(age);  }  }  public class Main {  public static void main(String[] args) {  Person person = new Person();  SetAgeCommand command = new SetAgeCommand(person, 25);  command.execute(); // Устанавливаем возраст  person.display(); // Выводим возраст  }  } |
| Python |
| # Класс Person  class Person:  def \_\_init\_\_(self):  self.age = 0  def set\_age(self, age):  self.age = age  def display(self):  print(f"Age: {self.age}")  # Интерфейс команды  class Command:  def execute(self):  pass  # Команда для изменения возраста  class SetAgeCommand(Command):  def \_\_init\_\_(self, person, age):  self.person = person  self.age = age  def execute(self):  self.person.set\_age(self.age)  # Пример использования  if \_\_name\_\_ == "\_\_main\_\_":  person = Person()  command = SetAgeCommand(person, 25)  command.execute() # Устанавливаем возраст  person.display() # Выводим возраст |

### Задачи для самостоятельной работы

Задача 2. Создайте команду, которая обновляет содержимое объекта класса Document. Реализуйте методы для установки нового содержимого и выполнения команды.

Задача 3. Определите команду, которая изменяет цену объекта класса Product. Реализуйте методы для установки новой цены и выполнения команды.

Задача 4. Создайте команду, которая изменяет количество объекта класса CartItem. Реализуйте методы для установки нового количества и выполнения команды.

Задача 5. Определите команду, которая добавляет элемент в объект класса Stack. Реализуйте методы для добавления элемента и выполнения команды.

Задача 6. Создайте команду, которая удаляет элемент из объекта класса Queue. Реализуйте методы для удаления элемента и выполнения команды.

Задача 7. Определите команду, которая вычисляет площадь объекта класса Shape (например, Circle или Rectangle). Реализуйте методы для выполнения команды и получения результата.

Задача 8. Создайте команду, которая обновляет модель объекта класса Vehicle. Реализуйте методы для установки новой модели и выполнения команды.

Задача 9. Определите команду, которая обновляет место проведения объекта класса Event. Реализуйте методы для установки нового места и выполнения команды.

Задача 10. Создайте команду, которая изменяет количество кредитов у объекта класса Course. Реализуйте методы для установки нового количества кредитов и выполнения команды.

Задача 11. Определите команду, которая обновляет начальную и конечную дату у объекта класса LeaveRequest. Реализуйте методы для установки новых дат и выполнения команды.

Задача 12. Создайте команду, которая изменяет электронную почту объекта класса Profile. Реализуйте методы для установки новой электронной почты и выполнения команды.

Задача 13. Определите команду, которая изменяет номер телефона объекта класса Contact. Реализуйте методы для установки нового номера и выполнения команды.

Задача 14. Создайте команду, которая изменяет описание объекта класса Task. Реализуйте методы для установки нового описания и выполнения команды.

Задача 15. Определите команду, которая изменяет модель объекта класса Computer. Реализуйте методы для установки новой модели и выполнения команды.

Задача 16. Создайте команду, которая изменяет автора объекта класса Book. Реализуйте методы для установки нового автора и выполнения команды.

Задача 17. Определите команду, которая изменяет обменный курс объекта класса CurrencyRate. Реализуйте методы для установки нового обменного курса и выполнения команды.

Задача 18. Создайте команду, которая обновляет баланс объекта класса Client. Реализуйте методы для установки нового баланса и выполнения команды.

Задача 19. Определите команду, которая пополняет счёт объекта класса Account. Реализуйте методы для добавления суммы и выполнения команды.

Задача 20. Создайте команду, которая изменяет адрес объекта класса Company. Реализуйте методы для установки нового адреса и выполнения команды.

Задача 21. Определите команду, которая изменяет должность объекта класса Employee. Реализуйте методы для установки новой должности и выполнения команды.

Задача 22. Создайте команду, которая изменяет режиссёра объекта класса Movie. Реализуйте методы для установки нового режиссёра и выполнения команды.

Задача 23. Определите команду, которая увеличивает количество студентов объекта класса University. Реализуйте методы для добавления студентов и выполнения команды.

Задача 24. Создайте команду, которая изменяет сумму объекта класса Payment. Реализуйте методы для установки новой суммы и выполнения команды.

Задача 25. Определите команду, которая изменяет сумму объекта класса Invoice. Реализуйте методы для установки новой суммы и выполнения команды.

Задача 26. Создайте команду, которая изменяет время приготовления объекта класса Recipe. Реализуйте методы для установки нового времени и выполнения команды.

Задача 27. Определите команду, которая изменяет цену объекта класса Gadget. Реализуйте методы для установки новой цены и выполнения команды.

Задача 28. Создайте команду, которая изменяет сумму объекта класса Transaction. Реализуйте методы для установки новой суммы и выполнения команды.

Задача 29. Определите команду, которая изменяет площадь объекта класса House. Реализуйте методы для установки новой площади и выполнения команды.

Задача 30. Создайте команду, которая изменяет описание объекта класса Project. Реализуйте методы для установки нового описания и выполнения команды.

Задача 31. Определите команду, которая изменяет количество этажей объекта класса Building. Реализуйте методы для установки нового количества этажей и выполнения команды.

Задача 32. Создайте команду, которая изменяет стоимость объекта класса TrainingCourse. Реализуйте методы для установки новой стоимости и выполнения команды.

Задача 33. Определите команду, которая изменяет версию объекта класса System. Реализуйте методы для установки новой версии и выполнения команды.

Задача 34. Создайте команду, которая изменяет начальную и конечную даты объекта класса Rental. Реализуйте методы для установки новых дат и выполнения команды.

Задача 35. Определите команду, которая изменяет масштаб объекта класса Map. Реализуйте методы для установки нового масштаба и выполнения команды.

## 4. Паттерн Iterator

Пример решений задач

Задача 1. Создать класс PersonCollection, который хранит объекты класса Person. Реализовать итератор, позволяющий проходить по списку людей и получать информацию о каждом человеке.

|  |
| --- |
| С++ |
| #include <iostream>  #include <vector>  #include <iterator>  // Класс, представляющий человека  class Person {  public:  std::string name; // Имя  int age; // Возраст  // Конструктор  Person(std::string n, int a) : name(n), age(a) {}  };  // Класс для хранения коллекции людей  class PersonCollection {  private:  std::vector<Person> people; // Вектор для хранения объектов Person  public:  // Метод для добавления человека в коллекцию  void addPerson(const Person& person) {  people.push\_back(person);  }  // Вложенный класс итератора  class Iterator {  private:  const PersonCollection& collection; // Ссылка на коллекцию  size\_t index; // Индекс текущего элемента  public:  // Конструктор итератора  Iterator(const PersonCollection& c, size\_t i) : collection(c), index(i) {}  // Метод для проверки наличия следующего элемента  bool hasNext() {  return index < collection.people.size();  }  // Метод для получения следующего элемента  Person next() {  return collection.people[index++];  }  };  // Метод для получения итератора  Iterator getIterator() const {  return Iterator(\*this, 0);  }  };  int main() {  PersonCollection collection; // Создание коллекции  collection.addPerson(Person("Alice", 30)); // Добавление людей  collection.addPerson(Person("Bob", 25));  // Получение итератора и вывод информации о людях  PersonCollection::Iterator it = collection.getIterator();  while (it.hasNext()) {  Person person = it.next();  std::cout << "Name: " << person.name << ", Age: " << person.age << std::endl;  }  return 0;  } |
| C# |
| using System;  using System.Collections.Generic;  // Класс, представляющий человека  class Person {  public string Name { get; set; } // Имя  public int Age { get; set; } // Возраст  // Конструктор  public Person(string name, int age) {  Name = name;  Age = age;  }  }  // Класс для хранения коллекции людей  class PersonCollection {  private List<Person> people = new List<Person>(); // Список для хранения объектов Person  // Метод для добавления человека в коллекцию  public void AddPerson(Person person) {  people.Add(person);  }  // Метод для получения итератора  public Iterator GetIterator() {  return new Iterator(this);  }  // Вложенный класс итератора  public class Iterator {  private PersonCollection collection; // Ссылка на коллекцию  private int index; // Индекс текущего элемента  // Конструктор итератора  public Iterator(PersonCollection collection) {  this.collection = collection;  index = 0;  }  // Метод для проверки наличия следующего элемента  public bool HasNext() {  return index < collection.people.Count;  }  // Метод для получения следующего элемента  public Person Next() {  return collection.people[index++];  }  }  }  class Program {  static void Main() {  PersonCollection collection = new PersonCollection(); // Создание коллекции  collection.AddPerson(new Person("Alice", 30)); // Добавление людей  collection.AddPerson(new Person("Bob", 25));  // Получение итератора и вывод информации о людях  PersonCollection.Iterator it = collection.GetIterator();  while (it.HasNext()) {  Person person = it.Next();  Console.WriteLine($"Name: {person.Name}, Age: {person.Age}");  }  }  } |
| Java |
| import java.util.ArrayList;  import java.util.List;  // Класс, представляющий человека  class Person {  String name; // Имя  int age; // Возраст  // Конструктор  Person(String name, int age) {  this.name = name;  this.age = age;  }  }  // Класс для хранения коллекции людей  class PersonCollection {  private List<Person> people = new ArrayList<>(); // Список для хранения объектов Person  // Метод для добавления человека в коллекцию  public void addPerson(Person person) {  people.add(person);  }  // Метод для получения итератора  public Iterator getIterator() {  return new Iterator(this);  }  // Вложенный класс итератора  public class Iterator {  private PersonCollection collection; // Ссылка на коллекцию  private int index; // Индекс текущего элемента  // Конструктор итератора  public Iterator(PersonCollection collection) {  this.collection = collection;  this.index = 0;  }  // Метод для проверки наличия следующего элемента  public boolean hasNext() {  return index < collection.people.size();  }  // Метод для получения следующего элемента  public Person next() {  return collection.people.get(index++);  }  }  }  // Главный класс  public class Main {  public static void main(String[] args) {  PersonCollection collection = new PersonCollection(); // Создание коллекции  collection.addPerson(new Person("Alice", 30)); // Добавление людей  collection.addPerson(new Person("Bob", 25));  // Получение итератора и вывод информации о людях  PersonCollection.Iterator it = collection.getIterator();  while (it.hasNext()) {  Person person = it.next();  System.out.println("Name: " + person.name + ", Age: " + person.age);  }  }  } |
| Python |
| # Класс, представляющий человека  class Person:  def \_\_init\_\_(self, name, age):  self.name = name # Имя  self.age = age # Возраст  # Класс для хранения коллекции людей  class PersonCollection:  def \_\_init\_\_(self):  self.people = [] # Список для хранения объектов Person  # Метод для добавления человека в коллекцию  def add\_person(self, person):  self.people.append(person)  # Метод для получения итератора  def get\_iterator(self):  return PersonIterator(self)  # Класс итератора  class PersonIterator:  def \_\_init\_\_(self, collection):  self.collection = collection # Ссылка на коллекцию  self.index = 0 # Индекс текущего элемента  # Метод для проверки наличия следующего элемента  def has\_next(self):  return self.index < len(self.collection.people)  # Метод для получения следующего элемента  def next(self):  person = self.collection.people[self.index]  self.index += 1  return person  # Главный код  if \_\_name\_\_ == "\_\_main\_\_":  collection = PersonCollection() # Создание коллекции  collection.add\_person(Person("Alice", 30)) # Добавление людей  collection.add\_person(Person("Bob", 25))  # Получение итератора и вывод информации о людях  iterator = collection.get\_iterator()  while iterator.has\_next():  person = iterator.next()  print(f"Name: {person.name}, Age: {person.age}") |

### Задачи для самостоятельной работы

Задача 2. Создать класс DocumentCollection, содержащий объекты класса Document. Реализовать итератор, который позволит перебирать документы и выводить информацию о каждом документе.

Задача 3. Создать класс Cart, который хранит объекты класса CartItem. Реализовать итератор, позволяющий проходить по товарам в корзине и получать информацию о каждом товаре.

Задача 4. Создать класс StackCollection, который реализует стек объектов. Реализовать итератор, позволяющий проходить по элементам стека, начиная с верхнего.

Задача 5. Создать класс QueueCollection, реализующий очередь. Реализовать итератор, позволяющий перебирать элементы очереди в порядке их добавления.

Задача 6. Создать класс ShapeCollection, который хранит объекты класса Shape. Реализовать итератор, позволяющий проходить по фигурам и вычислять их площади.

Задача 7. Создать класс VehicleCollection, содержащий объекты класса Vehicle. Реализовать итератор для перебора транспортных средств и получения информации о каждом из них.

Задача 8. Создать класс EventCollection, который хранит объекты класса Event. Реализовать итератор для перебора событий и получения информации о каждом событии.

Задача 9. Создать класс CourseCollection, содержащий объекты класса Course. Реализовать итератор, позволяющий проходить по курсам и получать информацию о каждом курсе.

Задача 10. Создать класс LeaveRequestCollection, который хранит объекты класса LeaveRequest. Реализовать итератор, позволяющий проходить по заявкам и получать информацию о каждой из них.

Задача 11. Создать класс ProfileCollection, содержащий объекты класса Profile. Реализовать итератор для перебора профилей и получения информации о каждом пользователе.

Задача 12. Создать класс ContactCollection, который хранит объекты класса Contact. Реализовать итератор для прохода по контактам и получения информации о каждом из них.

Задача 13. Создать класс TaskCollection, содержащий объекты класса Task. Реализовать итератор, позволяющий перебирать задачи и выводить информацию о каждой задаче.

Задача 14. Создать класс ComputerCollection, который хранит объекты класса Computer. Реализовать итератор для перебора компьютеров и получения информации о каждом из них.

Задача 15. Создать класс BookCollection, содержащий объекты класса Book. Реализовать итератор, позволяющий проходить по книгам и получать информацию о каждой книге.

Задача 16. Создать класс CurrencyRateCollection, который хранит объекты класса CurrencyRate. Реализовать итератор для перебора валют и получения информации о каждой валюте.

Задача 17. Создать класс ClientCollection, содержащий объекты класса Client. Реализовать итератор для прохода по клиентам и получения информации о каждом из них.

Задача 18. Создать класс AccountCollection, который хранит объекты класса Account. Реализовать итератор для перебора счетов и получения информации о каждом счете.

Задача 19. Создать класс CompanyCollection, содержащий объекты класса Company. Реализовать итератор для прохода по фирмам и получения информации о каждой фирме.

Задача 20. Создать класс EmployeeCollection, который хранит объекты класса Employee. Реализовать итератор для перебора работников и получения информации о каждом из них.

Задача 21. Создать класс MovieCollection, содержащий объекты класса Movie. Реализовать итератор для прохода по фильмам и получения информации о каждом фильме.

Задача 22. Создать класс UniversityCollection, который хранит объекты класса University. Реализовать итератор для перебора университетов и получения информации о каждом университете.

Задача 23. Создать класс PaymentCollection, содержащий объекты класса Payment. Реализовать итератор, позволяющий проходить по платежам и получать информацию о каждом платеже.

Задача 24. Создать класс InvoiceCollection, который хранит объекты класса Invoice. Реализовать итератор для перебора счетов-фактур и получения информации о каждом счете.

Задача 25. Создать класс RecipeCollection, содержащий объекты класса Recipe. Реализовать итератор для прохода по рецептам и получения информации о каждом рецепте.

Задача 26. Создать класс GadgetCollection, который хранит объекты класса Gadget. Реализовать итератор для перебора гаджетов и получения информации о каждом гаджете.

Задача 27. Создать класс TransactionCollection, содержащий объекты класса Transaction. Реализовать итератор, позволяющий проходить по финансовым операциям и получать информацию о каждой операции.

Задача 28. Создать класс HouseCollection, который хранит объекты класса House. Реализовать итератор для перебора домов и получения информации о каждом доме.

Задача 29. Создать класс ProjectCollection, содержащий объекты класса Project. Реализовать итератор для прохода по проектам и получения информации о каждом проекте.

Задача 30. Создать класс BuildingCollection, который хранит объекты класса Building. Реализовать итератор для перебора зданий и получения информации о каждом здании.

Задача 31. Создать класс TrainingCourseCollection, содержащий объекты класса TrainingCourse. Реализовать итератор для прохода по курсам обучения и получения информации о каждом курсе.

Задача 32. Создать класс SystemCollection, который хранит объекты класса System. Реализовать итератор для перебора систем и получения информации о каждой системе.

Задача 33. Создать класс RentalCollection, содержащий объекты класса Rental. Реализовать итератор для прохода по арендам и получения информации о каждой аренде.

Задача 34. Создать класс MapCollection, который хранит объекты класса Map. Реализовать итератор для перебора карт и получения информации о каждой карте.

Задача 35. Создать класс UserDataCollection, содержащий объекты класса UserData. Реализовать итератор, позволяющий проходить по пользовательским данным и получать информацию о каждом пользователе.

## 5. Паттерн State

Пример решений задач

Задача 1. Реализовать паттерн State для управления состояниями банковского счёта. Счёт может быть в одном из трёх состояний: "активный", "замороженный" и "закрытый". Реализовать методы для пополнения счёта, снятия средств и проверки баланса в зависимости от текущего состояния счёта.

|  |
| --- |
| С++ |
| #include <iostream>  using namespace std;  // Вперёд объявляем класс состояния счёта  class AccountState;  // Класс контекста, представляющий банковский счёт  class BankAccount {  private:  AccountState\* state; // Текущее состояние счёта  double balance; // Баланс счёта  public:  BankAccount() : state(nullptr), balance(0.0) {  state = new ActiveState(this); // Инициализация активным состоянием  }  void setState(AccountState\* newState); // Изменение состояния  void deposit(double amount); // Метод пополнения  void withdraw(double amount); // Метод снятия  void checkBalance(); // Метод проверки баланса  double getBalance() const { return balance; } // Получение текущего баланса  void addBalance(double amount) { balance += amount; } // Добавление к балансу  void subtractBalance(double amount) { balance -= amount; } // Вычитание из баланса  };  // Абстрактный класс состояния  class AccountState {  protected:  BankAccount\* account; // Ссылка на банковский счёт  public:  AccountState(BankAccount\* account) : account(account) {}  virtual void deposit(double amount) = 0; // Абстрактный метод пополнения  virtual void withdraw(double amount) = 0; // Абстрактный метод снятия  virtual void checkBalance() = 0; // Абстрактный метод проверки баланса  };  // Класс активного состояния  class ActiveState : public AccountState {  public:  ActiveState(BankAccount\* account) : AccountState(account) {}  void deposit(double amount) override {  account->addBalance(amount); // Добавление к балансу  cout << "Пополнено: " << amount << endl;  }  void withdraw(double amount) override {  if (amount <= account->getBalance()) {  account->subtractBalance(amount); // Вычитание из баланса  cout << "Снято: " << amount << endl;  } else {  cout << "Недостаточно средств!" << endl; // Недостаточно средств  }  }  void checkBalance() override {  cout << "Баланс: " << account->getBalance() << endl; // Отображение баланса  }  };  // Класс замороженного состояния  class FrozenState : public AccountState {  public:  FrozenState(BankAccount\* account) : AccountState(account) {}  void deposit(double amount) override {  account->addBalance(amount); // Позволяем пополнение даже при заморозке  cout << "Пополнено: " << amount << " (Счёт заморожен)" << endl;  }  void withdraw(double amount) override {  cout << "Счёт заморожен. Снятие невозможно." << endl; // Снятие не разрешено  }  void checkBalance() override {  cout << "Баланс: " << account->getBalance() << " (Счёт заморожен)" << endl; // Отображение баланса  }  };  // Класс закрытого состояния  class ClosedState : public AccountState {  public:  ClosedState(BankAccount\* account) : AccountState(account) {}  void deposit(double amount) override {  cout << "Счёт закрыт. Пополнение невозможно." << endl; // Пополнение не разрешено  }  void withdraw(double amount) override {  cout << "Счёт закрыт. Снятие невозможно." << endl; // Снятие не разрешено  }  void checkBalance() override {  cout << "Счёт закрыт. Баланс недоступен." << endl; // Нет баланса для отображения  }  };  // Реализация метода установки состояния  void BankAccount::setState(AccountState\* newState) {  delete state; // Освобождение текущего состояния  state = newState; // Установка нового состояния  }  // Главная функция  int main() {  BankAccount account; // Создание банковского счёта  account.deposit(100); // Пополнение счёта  account.checkBalance(); // Проверка баланса  account.withdraw(50); // Снятие средств  account.setState(new FrozenState(&account)); // Переход в замороженное состояние  account.deposit(50); // Пополнение при заморозке  account.withdraw(50); // Попытка снятия при заморозке  account.checkBalance(); // Проверка баланса  account.setState(new ClosedState(&account)); // Переход в закрытое состояние  account.deposit(100); // Попытка пополнения при закрытии  account.withdraw(50); // Попытка снятия при закрытии  account.checkBalance(); // Проверка баланса  return 0;  } |
| C# |
| using System;  // Абстрактный класс состояния  public abstract class AccountState {  protected BankAccount account;  protected AccountState(BankAccount account) {  this.account = account;  }  public abstract void Deposit(double amount);  public abstract void Withdraw(double amount);  public abstract void CheckBalance();  }  // Класс контекста, представляющий банковский счёт  public class BankAccount {  private AccountState state;  private double balance;  public BankAccount() {  state = new ActiveState(this); // Инициализация активным состоянием  }  public void SetState(AccountState newState) {  state = newState; // Установка нового состояния  }  public void Deposit(double amount) {  state.Deposit(amount); // Делегирование вызова состояния  }  public void Withdraw(double amount) {  state.Withdraw(amount); // Делегирование вызова состояния  }  public void CheckBalance() {  state.CheckBalance(); // Делегирование вызова состояния  }  public double GetBalance() {  return balance; // Получение текущего баланса  }  public void AddBalance(double amount) {  balance += amount; // Добавление к балансу  }  public void SubtractBalance(double amount) {  balance -= amount; // Вычитание из баланса  }  }  // Класс активного состояния  public class ActiveState : AccountState {  public ActiveState(BankAccount account) : base(account) {}  public override void Deposit(double amount) {  account.AddBalance(amount); // Добавление к балансу  Console.WriteLine($"Пополнено: {amount}");  }  public override void Withdraw(double amount) {  if (amount <= account.GetBalance()) {  account.SubtractBalance(amount); // Вычитание из баланса  Console.WriteLine($"Снято: {amount}");  } else {  Console.WriteLine("Недостаточно средств!"); // Недостаточно средств  }  }  public override void CheckBalance() {  Console.WriteLine($"Баланс: {account.GetBalance()}"); // Отображение баланса  }  }  // Класс замороженного состояния  public class FrozenState : AccountState {  public FrozenState(BankAccount account) : base(account) {}  public override void Deposit(double amount) {  account.AddBalance(amount); // Позволяем пополнение даже при заморозке  Console.WriteLine($"Пополнено: {amount} (Счёт заморожен)");  }  public override void Withdraw(double amount) {  Console.WriteLine("Счёт заморожен. Снятие невозможно."); // Снятие не разрешено  }  public override void CheckBalance() {  Console.WriteLine($"Баланс: {account.GetBalance()} (Счёт заморожен)"); // Отображение баланса  }  }  // Класс закрытого состояния  public class ClosedState : AccountState {  public ClosedState(BankAccount account) : base(account) {}  public override void Deposit(double amount) {  Console.WriteLine("Счёт закрыт. Пополнение невозможно."); // Пополнение не разрешено  }  public override void Withdraw(double amount) {  Console.WriteLine("Счёт закрыт. Снятие невозможно."); // Снятие не разрешено  }  public override void CheckBalance() {  Console.WriteLine("Счёт закрыт. Баланс недоступен."); // Нет баланса для отображения  }  }  // Главная программа  public class Program {  public static void Main() {  BankAccount account = new BankAccount(); // Создание банковского счёта  account.Deposit(100); // Пополнение счёта  account.CheckBalance(); // Проверка баланса  account.Withdraw(50); // Снятие средств  account.SetState(new FrozenState(account)); // Переход в замороженное состояние  account.Deposit(50); // Пополнение при заморозке  account.Withdraw(50); // Попытка снятия при заморозке  account.CheckBalance(); // Проверка баланса  account.SetState(new ClosedState(account)); // Переход в закрытое состояние  account.Deposit(100); // Попытка пополнения при закрытии  account.Withdraw(50); // Попытка снятия при закрытии  account.CheckBalance(); // Проверка баланса  }  } |
| Java |
| // Абстрактный класс состояния  abstract class AccountState {  protected BankAccount account;  public AccountState(BankAccount account) {  this.account = account;  }  public abstract void deposit(double amount);  public abstract void withdraw(double amount);  public abstract void checkBalance();  }  // Класс контекста, представляющий банковский счёт  class BankAccount {  private AccountState state;  private double balance;  public BankAccount() {  state = new ActiveState(this); // Инициализация активным состоянием  }  public void setState(AccountState newState) {  state = newState; // Установка нового состояния  }  public void deposit(double amount) {  state.deposit(amount); // Делегирование вызова состояния  }  public void withdraw(double amount) {  state.withdraw(amount); // Делегирование вызова состояния  }  public void checkBalance() {  state.checkBalance(); // Делегирование вызова состояния  }  public double getBalance() {  return balance; // Получение текущего баланса  }  public void addBalance(double amount) {  balance += amount; // Добавление к балансу  }  public void subtractBalance(double amount) {  balance -= amount; // Вычитание из баланса  }  }  // Класс активного состояния  class ActiveState extends AccountState {  public ActiveState(BankAccount account) {  super(account);  }  public void deposit(double amount) {  account.addBalance(amount); // Добавление к балансу  System.out.println("Пополнено: " + amount);  }  public void withdraw(double amount) {  if (amount <= account.getBalance()) {  account.subtractBalance(amount); // Вычитание из баланса  System.out.println("Снято: " + amount);  } else {  System.out.println("Недостаточно средств!"); // Недостаточно средств  }  }  public void checkBalance() {  System.out.println("Баланс: " + account.getBalance()); // Отображение баланса  }  }  // Класс замороженного состояния  class FrozenState extends AccountState {  public FrozenState(BankAccount account) {  super(account);  }  public void deposit(double amount) {  account.addBalance(amount); // Позволяем пополнение даже при заморозке  System.out.println("Пополнено: " + amount + " (Счёт заморожен)");  }  public void withdraw(double amount) {  System.out.println("Счёт заморожен. Снятие невозможно."); // Снятие не разрешено  }  public void checkBalance() {  System.out.println("Баланс: " + account.getBalance() + " (Счёт заморожен)"); // Отображение баланса  }  }  // Класс закрытого состояния  class ClosedState extends AccountState {  public ClosedState(BankAccount account) {  super(account);  }  public void deposit(double amount) {  System.out.println("Счёт закрыт. Пополнение невозможно."); // Пополнение не разрешено  }  public void withdraw(double amount) {  System.out.println("Счёт закрыт. Снятие невозможно."); // Снятие не разрешено  }  public void checkBalance() {  System.out.println("Счёт закрыт. Баланс недоступен."); // Нет баланса для отображения  }  }  // Главная программа  public class Main {  public static void main(String[] args) {  BankAccount account = new BankAccount(); // Создание банковского счёта  account.deposit(100); // Пополнение счёта  account.checkBalance(); // Проверка баланса  account.withdraw(50); // Снятие средств  account.setState(new FrozenState(account)); // Переход в замороженное состояние  account.deposit(50); // Пополнение при заморозке  account.withdraw(50); // Попытка снятия при заморозке  account.checkBalance(); // Проверка баланса  account.setState(new ClosedState(account)); // Переход в закрытое состояние  account.deposit(100); // Попытка пополнения при закрытии  account.withdraw(50); // Попытка снятия при закрытии  account.checkBalance(); // Проверка баланса  }  } |
| Python |
| # Абстрактный класс состояния  class AccountState:  def \_\_init\_\_(self, account):  self.account = account  def deposit(self, amount):  raise NotImplementedError  def withdraw(self, amount):  raise NotImplementedError  def check\_balance(self):  raise NotImplementedError  # Класс контекста, представляющий банковский счёт  class BankAccount:  def \_\_init\_\_(self):  self.state = ActiveState(self) # Инициализация активным состоянием  self.balance = 0.0  def set\_state(self, new\_state):  self.state = new\_state # Установка нового состояния  def deposit(self, amount):  self.state.deposit(amount) # Делегирование вызова состояния  def withdraw(self, amount):  self.state.withdraw(amount) # Делегирование вызова состояния  def check\_balance(self):  self.state.check\_balance() # Делегирование вызова состояния  def add\_balance(self, amount):  self.balance += amount # Добавление к балансу  def subtract\_balance(self, amount):  self.balance -= amount # Вычитание из баланса  def get\_balance(self):  return self.balance # Получение текущего баланса  # Класс активного состояния  class ActiveState(AccountState):  def deposit(self, amount):  self.account.add\_balance(amount) # Добавление к балансу  print(f'Пополнено: {amount}')  def withdraw(self, amount):  if amount <= self.account.get\_balance():  self.account.subtract\_balance(amount) # Вычитание из баланса  print(f'Снято: {amount}')  else:  print('Недостаточно средств!') # Недостаточно средств  def check\_balance(self):  print(f'Баланс: {self.account.get\_balance()}') # Отображение баланса  # Класс замороженного состояния  class FrozenState(AccountState):  def deposit(self, amount):  self.account.add\_balance(amount) # Позволяем пополнение даже при заморозке  print(f'Пополнено: {amount} (Счёт заморожен)')  def withdraw(self, amount):  print('Счёт заморожен. Снятие невозможно.') # Снятие не разрешено  def check\_balance(self):  print(f'Баланс: {self.account.get\_balance()} (Счёт заморожен)') # Отображение баланса  # Класс закрытого состояния  class ClosedState(AccountState):  def deposit(self, amount):  print('Счёт закрыт. Пополнение невозможно.') # Пополнение не разрешено  def withdraw(self, amount):  print('Счёт закрыт. Снятие невозможно.') # Снятие не разрешено  def check\_balance(self):  print('Счёт закрыт. Баланс недоступен.') # Нет баланса для отображения  # Главная программа  if \_\_name\_\_ == '\_\_main\_\_':  account = BankAccount() # Создание банковского счёта  account.deposit(100) # Пополнение счёта  account.check\_balance() # Проверка баланса  account.withdraw(50) # Снятие средств  account.set\_state(FrozenState(account)) # Переход в замороженное состояние  account.deposit(50) # Пополнение при заморозке  account.withdraw(50) # Попытка снятия при заморозке  account.check\_balance() # Проверка баланса  account.set\_state(ClosedState(account)) # Переход в закрытое состояние  account.deposit(100) # Попытка пополнения при закрытии  account.withdraw(50) # Попытка снятия при закрытии  account.check\_balance() # Проверка баланса |

### Задачи для самостоятельной работы

Задача 2. Создать класс VendingMachine с различными состояниями: "ожидание монеты", "выдача напитка" и "изменение". Реализовать методы для внесения монет, выбора напитка и выдачи напитка.

Задача 3. Определить паттерн State для музыкального плеера с состояниями "воспроизведение", "пауза" и "остановка". Реализовать методы для воспроизведения, приостановки и остановки трека в зависимости от текущего состояния плеера.

Задача 4. Создать систему умного освещения с состояниями "включено", "выключено" и "дими". Реализовать методы для включения, выключения и изменения яркости света в зависимости от текущего состояния.

Задача 5. Определить паттерн State для заказа в ресторане. Заказ может находиться в одном из состояний: "новый", "в процессе", "готов" и "доставлен". Реализовать методы для обновления статуса заказа и получения информации о текущем статусе.

Задача 6. Создать класс Printer с состояниями "готов к печати", "печать" и "отключён". Реализовать методы для печати, проверки состояния и отмены печати в зависимости от текущего состояния устройства.

Задача 7. Определить паттерн State для управления лифтом, который может быть в одном из состояний: "ожидание", "в движении вверх" и "в движении вниз". Реализовать методы для вызова лифта, выбора этажа и движения лифта.

Задача 8. Создать систему интернет-магазина с состояниями "добавление товара в корзину", "оформление заказа" и "завершение покупки". Реализовать методы для добавления товаров, оформления заказа и завершения покупки в зависимости от текущего состояния.

Задача 9. Определить паттерн State для телескопа, который может находиться в состояниях "включен", "выключен" и "калибровка". Реализовать методы для включения, выключения и калибровки телескопа.

Задача 10. Создать класс VideoGame с состояниями "меню", "игра" и "пауза". Реализовать методы для запуска игры, приостановки игры и выхода в меню в зависимости от текущего состояния.

Задача 11. Определить паттерн State для веб-приложения с состояниями "авторизация", "загрузка данных" и "пользовательский интерфейс". Реализовать методы для авторизации, загрузки данных и работы с пользовательским интерфейсом.

Задача 12. Создать класс Printer с состояниями "готов к печати", "печать" и "отсутствие бумаги". Реализовать методы для начала печати, отмены печати и проверки состояния наличия бумаги.

Задача 13. Определить паттерн State для системы контроля доступа с состояниями "вход разрешен", "вход запрещен" и "ожидание аутентификации". Реализовать методы для проверки доступа и обработки аутентификации.

Задача 14. Создать систему управления электронной почтой с состояниями "новое сообщение", "прочитанное" и "удаленное". Реализовать методы для прочтения сообщения, удаления сообщения и проверки статуса сообщения.

Задача 15. Определить паттерн State для управления состояниями документа с состояниями "черновик", "на рассмотрении" и "утвержден". Реализовать методы для перехода между состояниями и отображения текущего состояния документа.

Задача 16. Создать систему управления проектом с состояниями "планирование", "в процессе" и "завершен". Реализовать методы для планирования проекта, изменения статуса и завершения проекта.

Задача 17. Определить паттерн State для учета расхода топлива с состояниями "пустой бак", "полупустой бак" и "полный бак". Реализовать методы для проверки уровня топлива и расчета пробега.

Задача 18. Создать класс SecuritySystem с состояниями "активен", "отключен" и "в режиме ожидания". Реализовать методы для активации, деактивации и проверки состояния системы охраны.

Задача 19. Определить паттерн State для электронного устройства с состояниями "включено", "выключено" и "в спящем режиме". Реализовать методы для включения, выключения и перехода в спящий режим.

Задача 20. Создать систему управления инвентарем с состояниями "в наличии", "продано" и "в резерве". Реализовать методы для обновления статуса товара и проверки наличия товара.

Задача 21. Определить паттерн State для веб-сервиса с состояниями "доступен", "не доступен" и "в обслуживании". Реализовать методы для проверки доступности и перехода между состояниями.

Задача 22. Создать класс Refrigerator с состояниями "работает", "отключен" и "заморозка". Реализовать методы для включения, выключения и настройки режима заморозки.

Задача 23. Определить паттерн State для системы отчетности с состояниями "подготовка отчета", "отчет готов" и "отчет опубликован". Реализовать методы для подготовки, проверки готовности и публикации отчета.

Задача 24. Создать систему управления термостатом с состояниями "включен", "выключен" и "отпуск". Реализовать методы для включения, выключения и настройки температуры.

Задача 25. Определить паттерн State для обучающего курса с состояниями "запланирован", "в процессе" и "завершен". Реализовать методы для изменения состояния курса и проверки статуса.

Задача 26. Создать класс ReviewProcess с состояниями "ожидание рецензии", "в рецензии" и "рецензия завершена". Реализовать методы для начала рецензирования и проверки статуса.

Задача 27. Определить паттерн State для системы расписаний с состояниями "не активен", "активен" и "завершен". Реализовать методы для активации, деактивации и завершения расписания.

Задача 28. Создать класс License с состояниями "активна", "не активна" и "в ожидании активации". Реализовать методы для активации, деактивации и проверки состояния лицензии.

Задача 29. Определить паттерн State для обучающего модуля с состояниями "не начат", "в процессе" и "завершен". Реализовать методы для начала, продолжения и завершения обучения.

Задача 30. Создать систему отслеживания посылок с состояниями "в пути", "доставлена" и "вернулась". Реализовать методы для обновления статуса и проверки состояния посылки.

Задача 31. Определить паттерн State для смарт-термостата с состояниями "включен", "выключен" и "режим ожидания". Реализовать методы для изменения режима работы и проверки состояния термостата.

Задача 32. Создать систему публикаций с состояниями "черновик", "опубликован" и "удален". Реализовать методы для изменения статуса публикации и получения информации о текущем статусе.

Задача 33. Определить паттерн State для системы машинного обучения с состояниями "обучение", "оценка" и "готово". Реализовать методы для начала обучения, оценки модели и завершения процесса.

Задача 34. Создать класс TestingProcess с состояниями "не начато", "в процессе" и "завершено". Реализовать методы для начала тестирования, выполнения тестов и завершения процесса.

Задача 35. Определить паттерн State для платформы обмена сообщениями с состояниями "доступен", "занят" и "не доступен". Реализовать методы для изменения статуса и обработки взаимодействий.

## 6. Паттерн Mediator

Пример решений задач

Задача 1. Создайте класс ChatRoom, который выступает в роли посредника между участниками чата. Реализуйте методы для добавления и удаления участников, а также для отправки сообщений между участниками.

|  |
| --- |
| С++ |
| #include <iostream>  #include <vector>  #include <string>  #include <algorithm>  // Впереди объявление класса Participant  class Participant;  class ChatRoom {  public:  // Метод для добавления участника  void join(Participant\* participant);  // Метод для удаления участника  void leave(Participant\* participant);  // Метод для отправки сообщения от одного участника к другим  void sendMessage(const std::string& message, Participant\* sender);  private:  std::vector<Participant\*> participants; // Список участников  };  class Participant {  public:  // Конструктор, который добавляет участника в чат  Participant(const std::string& name, ChatRoom\* chatRoom) : name(name), chatRoom(chatRoom) {  chatRoom->join(this);  }  // Деструктор, который удаляет участника из чата  ~Participant() {  chatRoom->leave(this);  }  // Метод для отправки сообщения  void send(const std::string& message) {  chatRoom->sendMessage(message, this);  }  // Метод для получения сообщения  void receive(const std::string& message) {  std::cout << name << " получил: " << message << std::endl;  }  private:  std::string name; // Имя участника  ChatRoom\* chatRoom; // Ссылка на чат  };  // Реализация методов класса ChatRoom  void ChatRoom::join(Participant\* participant) {  participants.push\_back(participant);  }  void ChatRoom::leave(Participant\* participant) {  participants.erase(std::remove(participants.begin(), participants.end(), participant), participants.end());  }  void ChatRoom::sendMessage(const std::string& message, Participant\* sender) {  for (auto participant : participants) {  if (participant != sender) {  participant->receive(message);  }  }  }  int main() {  ChatRoom chatRoom; // Создаем чат  Participant john("John", &chatRoom); // Создаем участника John  Participant jane("Jane", &chatRoom); // Создаем участника Jane  john.send("Привет всем!"); // John отправляет сообщение  jane.send("Привет, John!"); // Jane отвечает  return 0;  } |
| C# |
| using System;  using System.Collections.Generic;  class ChatRoom {  private List<Participant> participants = new List<Participant>(); // Список участников  // Метод для добавления участника  public void Join(Participant participant) {  participants.Add(participant);  }  // Метод для удаления участника  public void Leave(Participant participant) {  participants.Remove(participant);  }  // Метод для отправки сообщения  public void SendMessage(string message, Participant sender) {  foreach (var participant in participants) {  if (participant != sender) {  participant.Receive(message); // Отправляем сообщение всем, кроме отправителя  }  }  }  }  class Participant {  private string name; // Имя участника  private ChatRoom chatRoom; // Ссылка на чат  // Конструктор, который добавляет участника в чат  public Participant(string name, ChatRoom chatRoom) {  this.name = name;  this.chatRoom = chatRoom;  chatRoom.Join(this); // Добавляем участника в чат  }  // Метод для отправки сообщения  public void Send(string message) {  chatRoom.SendMessage(message, this); // Отправляем сообщение через чат  }  // Метод для получения сообщения  public void Receive(string message) {  Console.WriteLine($"{name} получил: {message}");  }  }  class Program {  static void Main() {  ChatRoom chatRoom = new ChatRoom(); // Создаем чат  Participant john = new Participant("John", chatRoom); // Создаем участника John  Participant jane = new Participant("Jane", chatRoom); // Создаем участника Jane  john.Send("Привет всем!"); // John отправляет сообщение  jane.Send("Привет, John!"); // Jane отвечает  }  } |
| Java |
| import java.util.ArrayList;  import java.util.List;  class ChatRoom {  private List<Participant> participants = new ArrayList<>(); // Список участников  // Метод для добавления участника  public void join(Participant participant) {  participants.add(participant);  }  // Метод для удаления участника  public void leave(Participant participant) {  participants.remove(participant);  }  // Метод для отправки сообщения  public void sendMessage(String message, Participant sender) {  for (Participant participant : participants) {  if (participant != sender) {  participant.receive(message); // Отправляем сообщение всем, кроме отправителя  }  }  }  }  class Participant {  private String name; // Имя участника  private ChatRoom chatRoom; // Ссылка на чат  // Конструктор, который добавляет участника в чат  public Participant(String name, ChatRoom chatRoom) {  this.name = name;  this.chatRoom = chatRoom;  chatRoom.join(this); // Добавляем участника в чат  }  // Метод для отправки сообщения  public void send(String message) {  chatRoom.sendMessage(message, this); // Отправляем сообщение через чат  }  // Метод для получения сообщения  public void receive(String message) {  System.out.println(name + " получил: " + message);  }  }  public class Main {  public static void main(String[] args) {  ChatRoom chatRoom = new ChatRoom(); // Создаем чат  Participant john = new Participant("John", chatRoom); // Создаем участника John  Participant jane = new Participant("Jane", chatRoom); // Создаем участника Jane  john.send("Привет всем!"); // John отправляет сообщение  jane.send("Привет, John!"); // Jane отвечает  }  } |
| Python |
| class ChatRoom:  def \_\_init\_\_(self):  self.participants = [] # Список участников  # Метод для добавления участника  def join(self, participant):  self.participants.append(participant)  # Метод для удаления участника  def leave(self, participant):  self.participants.remove(participant)  # Метод для отправки сообщения  def send\_message(self, message, sender):  for participant in self.participants:  if participant != sender:  participant.receive(message) # Отправляем сообщение всем, кроме отправителя  class Participant:  def \_\_init\_\_(self, name, chat\_room):  self.name = name # Имя участника  self.chat\_room = chat\_room # Ссылка на чат  chat\_room.join(self) # Добавляем участника в чат  # Метод для отправки сообщения  def send(self, message):  self.chat\_room.send\_message(message, self) # Отправляем сообщение через чат  # Метод для получения сообщения  def receive(self, message):  print(f"{self.name} получил: {message}")  if \_\_name\_\_ == "\_\_main\_\_":  chat\_room = ChatRoom() # Создаем чат  john = Participant("John", chat\_room) # Создаем участника John  jane = Participant("Jane", chat\_room) # Создаем участника Jane  john.send("Привет всем!") # John отправляет сообщение  jane.send("Привет, John!") # Jane отвечает |

### Задачи для самостоятельной работы

Задача 2. Определите класс CentralRegistry, который управляет взаимодействием между разными объектами, такими как Employee, Department и Project. Реализуйте методы для регистрации изменений и уведомления о них.

Задача 3. Создайте класс ProjectManager, который управляет взаимодействием между разными проектами и их участниками. Реализуйте методы для добавления участников в проекты и для обновления статуса проектов.

Задача 4. Определите класс BankSystem, который координирует взаимодействие между различными счетами, клиентами и транзакциями. Реализуйте методы для выполнения транзакций и уведомления о изменениях в состоянии счетов.

Задача 5. Создайте класс LibrarySystem, который управляет взаимодействием между книгами, пользователями и выдачей книг. Реализуйте методы для запроса и возврата книг, а также для обновления статуса книг.

Задача 6. Определите класс ScheduleManager, который координирует расписание встреч между участниками. Реализуйте методы для добавления, удаления и изменения встреч.

Задача 7. Создайте класс StockMarket, который управляет взаимодействием между трейдерами, акциями и котировками. Реализуйте методы для обновления котировок и обработки заявок на покупку и продажу акций.

Задача 8. Определите класс AutomationSystem, который управляет взаимодействием между различными процессами на заводе. Реализуйте методы для координации работы машин и контроля качества продукции.

Задача 9. Создайте класс OrderManager, который управляет взаимодействием между клиентами, заказами и складами. Реализуйте методы для создания заказов, обновления статуса и отслеживания выполнения.

Задача 10. Определите класс MedicalSystem, который управляет взаимодействием между врачами, пациентами и медицинскими записями. Реализуйте методы для записи на прием, обновления медицинских карт и назначения лечения.

Задача 11. Создайте класс UserInteractionManager, который управляет взаимодействием между пользователями и интерфейсом приложения. Реализуйте методы для обработки запросов пользователей и обновления интерфейса.

Задача 12. Определите класс EnergySystem, который координирует работу различных энергетических источников и потребителей. Реализуйте методы для распределения энергии и мониторинга потребления.

Задача 13. Создайте класс InventoryManager, который управляет взаимодействием между инвентарем, складами и закупками. Реализуйте методы для обновления инвентаря и управления заказами на новые поступления.

Задача 14. Определите класс MonitoringSystem, который управляет взаимодействием между различными датчиками и системами оповещения. Реализуйте методы для обработки данных от датчиков и генерации оповещений.

Задача 15. Создайте класс ContentManager, который координирует работу с различными типами контента, такими как статьи, изображения и видео. Реализуйте методы для добавления, обновления и удаления контента.

Задача 16. Определите класс SecuritySystem, который управляет взаимодействием между системами контроля доступа, камерами видеонаблюдения и сигнализациями. Реализуйте методы для настройки и мониторинга систем безопасности.

Задача 17. Создайте класс CurriculumManager, который координирует взаимодействие между преподавателями, курсами и студентами. Реализуйте методы для планирования курсов и отслеживания успеваемости студентов.

Задача 18. Определите класс FinancialManager, который управляет взаимодействием между клиентами, финансовыми отчетами и транзакциями. Реализуйте методы для обработки транзакций и создания отчетов.

Задача 19. Создайте класс ReservationManager, который управляет взаимодействием между клиентами, ресурсами и резервациями. Реализуйте методы для создания, обновления и отмены резерваций.

Задача 20. Определите класс SupportSystem, который координирует работу между клиентами, специалистами поддержки и системами отслеживания запросов. Реализуйте методы для обработки и разрешения запросов.

Задача 21. Создайте класс ResourceManager, который управляет взаимодействием между различными типами ресурсов и их пользователями. Реализуйте методы для распределения и мониторинга использования ресурсов.

Задача 22. Определите класс EventCoordinator, который координирует взаимодействие между организаторами событий, участниками и ресурсами. Реализуйте методы для планирования и управления событиями.

Задача 23. Создайте класс TrainingManager, который управляет взаимодействием между тренерами, участниками и учебными материалами. Реализуйте методы для планирования и проведения тренингов.

Задача 24. Определите класс TransportSystem, который координирует работу различных транспортных средств и маршрутов. Реализуйте методы для планирования маршрутов и управления движением.

Задача 25. Создайте класс AdManager, который управляет взаимодействием между рекламодателями, рекламными кампаниями и пользователями. Реализуйте методы для создания и отслеживания рекламных кампаний.

Задача 26. Определите класс TaskTracker, который координирует взаимодействие между задачами, пользователями и статусами задач. Реализуйте методы для создания, обновления и отслеживания задач.

Задача 27. Создайте класс FileManager, который управляет взаимодействием между файлами, пользователями и хранилищами. Реализуйте методы для загрузки, удаления и поиска файлов.

Задача 28. Определите класс FinancialAdvisor, который координирует работу между клиентами, инвестициями и финансовыми рынками. Реализуйте методы для предоставления рекомендаций и отслеживания инвестиций.

Задача 29. Создайте класс NotificationSystem, который управляет взаимодействием между различными источниками уведомлений и пользователями. Реализуйте методы для создания и отправки уведомлений.

Задача 30. Определите класс ProjectResourceManager, который координирует работу с ресурсами, выделенными для проектов. Реализуйте методы для распределения ресурсов и отслеживания их использования.

Задача 31. Создайте класс EventManagementSystem, который управляет взаимодействием между организаторами событий, участниками и ресурсами. Реализуйте методы для планирования и координации событий.

Задача 32. Определите класс TaskManager, который координирует работу между различными задачами, проектами и командами. Реализуйте методы для распределения задач и отслеживания их выполнения.

Задача 33. Создайте класс IncidentManager, который управляет взаимодействием между инцидентами, их разрешением и отчетами. Реализуйте методы для регистрации и обработки инцидентов.

Задача 34. Определите класс RequestManager, который координирует работу с заявками, их обработкой и выполнением. Реализуйте методы для создания, отслеживания и завершения заявок.

Задача 35. Создайте класс ProjectTaskCoordinator, который управляет взаимодействием между проектами и задачами. Реализуйте методы для добавления задач в проект и отслеживания их статуса.

## 7. Паттерн Chain of Responsibility

Пример решений задач

Задача 1. Реализуйте цепочку обработчиков запросов для класса Request, где каждый обработчик может изменять или обрабатывать запрос. Примером может быть цепочка из обработчиков для фильтрации, проверки и логирования запросов.

|  |
| --- |
| С++ |
| #include <iostream>  #include <string>  // Класс запроса  class Request {  public:  std::string message;  Request(const std::string& msg) : message(msg) {}  };  // Абстрактный класс обработчика  class Handler {  protected:  Handler\* nextHandler;  public:  Handler() : nextHandler(nullptr) {}    void setNext(Handler\* handler) {  nextHandler = handler;  }    virtual void handle(Request\* request) = 0;  };  // Обработчик фильтрации  class FilterHandler : public Handler {  public:  void handle(Request\* request) override {  if (request->message.find("badword") != std::string::npos) {  std::cout << "FilterHandler: Bad word found. Request blocked." << std::endl;  return;  }  std::cout << "FilterHandler: No bad words. Passing to next handler." << std::endl;  if (nextHandler) nextHandler->handle(request);  }  };  // Обработчик проверки  class ValidationHandler : public Handler {  public:  void handle(Request\* request) override {  if (request->message.empty()) {  std::cout << "ValidationHandler: Request is empty. Request blocked." << std::endl;  return;  }  std::cout << "ValidationHandler: Request is valid. Passing to next handler." << std::endl;  if (nextHandler) nextHandler->handle(request);  }  };  // Обработчик логирования  class LoggingHandler : public Handler {  public:  void handle(Request\* request) override {  std::cout << "LoggingHandler: Logging request: " << request->message << std::endl;  if (nextHandler) nextHandler->handle(request);  }  };  int main() {  Request req("This is a test request without badword.");  FilterHandler filter;  ValidationHandler validator;  LoggingHandler logger;  filter.setNext(&validator);  validator.setNext(&logger);  filter.handle(&req);  return 0;  } |
| C# |
| using System;  public class Request {  public string Message { get; }  public Request(string message) {  Message = message;  }  }  public abstract class Handler {  protected Handler nextHandler;  public void SetNext(Handler handler) {  nextHandler = handler;  }  public abstract void Handle(Request request);  }  public class FilterHandler : Handler {  public override void Handle(Request request) {  if (request.Message.Contains("badword")) {  Console.WriteLine("FilterHandler: Bad word found. Request blocked.");  return;  }  Console.WriteLine("FilterHandler: No bad words. Passing to next handler.");  nextHandler?.Handle(request);  }  }  public class ValidationHandler : Handler {  public override void Handle(Request request) {  if (string.IsNullOrEmpty(request.Message)) {  Console.WriteLine("ValidationHandler: Request is empty. Request blocked.");  return;  }  Console.WriteLine("ValidationHandler: Request is valid. Passing to next handler.");  nextHandler?.Handle(request);  }  }  public class LoggingHandler : Handler {  public override void Handle(Request request) {  Console.WriteLine($"LoggingHandler: Logging request: {request.Message}");  nextHandler?.Handle(request);  }  }  class Program {  static void Main() {  var req = new Request("This is a test request without badword.");  var filter = new FilterHandler();  var validator = new ValidationHandler();  var logger = new LoggingHandler();  filter.SetNext(validator);  validator.SetNext(logger);  filter.Handle(req);  }  } |
| Java |
| class Request {  String message;  public Request(String message) {  this.message = message;  }  }  abstract class Handler {  protected Handler nextHandler;  public void setNext(Handler handler) {  nextHandler = handler;  }  public abstract void handle(Request request);  }  class FilterHandler extends Handler {  public void handle(Request request) {  if (request.message.contains("badword")) {  System.out.println("FilterHandler: Bad word found. Request blocked.");  return;  }  System.out.println("FilterHandler: No bad words. Passing to next handler.");  if (nextHandler != null) nextHandler.handle(request);  }  }  class ValidationHandler extends Handler {  public void handle(Request request) {  if (request.message.isEmpty()) {  System.out.println("ValidationHandler: Request is empty. Request blocked.");  return;  }  System.out.println("ValidationHandler: Request is valid. Passing to next handler.");  if (nextHandler != null) nextHandler.handle(request);  }  }  class LoggingHandler extends Handler {  public void handle(Request request) {  System.out.println("LoggingHandler: Logging request: " + request.message);  if (nextHandler != null) nextHandler.handle(request);  }  }  public class Main {  public static void main(String[] args) {  Request req = new Request("This is a test request without badword.");  FilterHandler filter = new FilterHandler();  ValidationHandler validator = new ValidationHandler();  LoggingHandler logger = new LoggingHandler();  filter.setNext(validator);  validator.setNext(logger);  filter.handle(req);  }  } |
| Python |
| class Request:  def \_\_init\_\_(self, message):  self.message = message  class Handler:  def \_\_init\_\_(self):  self.next\_handler = None  def set\_next(self, handler):  self.next\_handler = handler  def handle(self, request):  raise NotImplementedError("You should implement this method.")  class FilterHandler(Handler):  def handle(self, request):  if "badword" in request.message:  print("FilterHandler: Bad word found. Request blocked.")  return  print("FilterHandler: No bad words. Passing to next handler.")  if self.next\_handler:  self.next\_handler.handle(request)  class ValidationHandler(Handler):  def handle(self, request):  if not request.message:  print("ValidationHandler: Request is empty. Request blocked.")  return  print("ValidationHandler: Request is valid. Passing to next handler.")  if self.next\_handler:  self.next\_handler.handle(request)  class LoggingHandler(Handler):  def handle(self, request):  print(f"LoggingHandler: Logging request: {request.message}")  if self.next\_handler:  self.next\_handler.handle(request)  if \_\_name\_\_ == "\_\_main\_\_":  req = Request("This is a test request without badword.")  filter\_handler = FilterHandler()  validation\_handler = ValidationHandler()  logging\_handler = LoggingHandler()  filter\_handler.set\_next(validation\_handler)  validation\_handler.set\_next(logging\_handler)  filter\_handler.handle(req) |

### Задачи для самостоятельной работы

Задача 2. Создайте цепочку валидаторов для класса UserData, которая последовательно проверяет корректность имени, адреса электронной почты и возраста пользователя.

Задача 3. Определите цепочку обработчиков для класса Payment, где каждый обработчик отвечает за проверку наличия средств, обработку налогов и регистрацию транзакций.

Задача 4. Реализуйте цепочку обработчиков для класса Document, где каждый обработчик отвечает за проверку формата документа, проверку подписи и содержание.

Задача 5. Создайте цепочку обработчиков для класса Task, которая отвечает за проверку статуса задачи, её приоритет и назначение ответственного.

Задача 6. Определите цепочку обработчиков для класса LeaveRequest, где каждый обработчик отвечает за проверку даты начала и окончания отпуска, доступные дни отпуска и одобрение руководителя.

Задача 7. Реализуйте цепочку обработчиков для класса CartItem, которая последовательно проверяет доступность товара, применяет скидки и рассчитывает общую стоимость.

Задача 8. Создайте цепочку обработчиков для класса Transaction, которая отвечает за проверку типа операции, её сумму и авторизацию.

Задача 9. Определите цепочку обработчиков для класса ServerRequest, которая последовательно выполняет аутентификацию, авторизацию и обработку запросов.

Задача 10. Реализуйте цепочку обработчиков для класса Event, где каждый обработчик отвечает за уведомление участников, обновление статуса и обработку изменений.

Задача 11. Создайте цепочку обработчиков для класса FormData, которая последовательно проверяет обязательные поля, формат данных и корректность значений.

Задача 12. Определите цепочку обработчиков для класса Project, которая проверяет состояние проекта, его бюджет и сроки выполнения.

Задача 13. Реализуйте цепочку обработчиков для класса Message, где каждый обработчик отвечает за фильтрацию спама, проверку подлинности и распределение по папкам.

Задача 14. Создайте цепочку обработчиков для класса Book, которая последовательно проверяет доступность книги, её состояние и выдачу.

Задача 15. Определите цепочку обработчиков для класса Inventory, которая последовательно проверяет уровень запасов, делает заказы на пополнение и обновляет информацию о наличии.

Задача 16. Реализуйте цепочку обработчиков для класса Order, которая отвечает за проверку наличия товара, расчёт стоимости, применение скидок и оформление заказа.

Задача 17. Создайте цепочку обработчиков для класса BankAccount, которая обрабатывает запросы на пополнение, снятие и перевод средств.

Задача 18. Определите цепочку обработчиков для класса ProjectTask, которая проверяет статус выполнения задачи, её приоритет и ресурсы.

Задача 19. Реализуйте цепочку обработчиков для класса Application, которая отвечает за проверку правильности заявки, её соответствие требованиям и назначение на рассмотрение.

Задача 20. Создайте цепочку обработчиков для класса DiscountRequest, которая проверяет обоснование запроса, доступные скидки и одобрение.

Задача 21. Определите цепочку обработчиков для класса Incident, которая последовательно выполняет регистрацию инцидента, оценку серьёзности и определение дальнейших действий.

Задача 22. Реализуйте цепочку обработчиков для класса SupportRequest, где каждый обработчик отвечает за проверку типа запроса, его направление и предоставление решения.

Задача 23. Создайте цепочку обработчиков для класса User, которая проверяет входные данные, аутентификацию и авторизацию.

Задача 24. Определите цепочку обработчиков для класса Resume, которая последовательно проверяет формат резюме, опыт работы и квалификацию кандидата.

Задача 25. Реализуйте цепочку обработчиков для класса CreditApplication, которая проверяет кредитную историю, доходы и одобрение заявки.

Задача 26. Создайте цепочку обработчиков для класса Email, которая отвечает за фильтрацию спама, проверку вложений и форматирования.

Задача 27. Определите цепочку обработчиков для класса RestaurantOrder, которая последовательно проверяет доступность ингредиентов, готовку и доставку.

Задача 28. Реализуйте цепочку обработчиков для класса Complaint, которая обрабатывает жалобы, делает предварительный анализ и передает в соответствующий департамент.

Задача 29. Создайте цепочку обработчиков для класса Recommendation, которая анализирует интересы пользователя, предлагает товары и услуги и обновляет рекомендации.

Задача 30. Определите цепочку обработчиков для класса ReturnRequest, которая проверяет причину возврата, состояние товара и оформление возврата.

Задача 31. Реализуйте цепочку обработчиков для класса InventoryCheck, которая отвечает за проверку состояния запасов, анализ дефицита и создание отчётов.

Задача 32. Создайте цепочку обработчиков для класса EventPlanning, которая отвечает за планирование места проведения, приглашения и организацию мероприятий.

Задача 33. Определите цепочку обработчиков для класса DataChangeRequest, которая последовательно проверяет тип изменений, их допустимость и одобрение.

Задача 34. Реализуйте цепочку обработчиков для класса TrainingProgram, которая проверяет содержание курса, ресурсы и расписание.

Задача 35. Создайте цепочку обработчиков для класса GameEvent, которая последовательно обрабатывает события в игре.

# Литература

1. Zandstra, M. PHP Objects, Patterns, and Practice. — 5th Edition. — Apress, 2016. — 583 с. — ISBN 978-1-4842-1995-9.
2. Абрамян М. Э. Практикум по программированию на языке Паскаль. 4-е изд. — Ростов-на-Дону: Изд-во «ЦВВР», 2004. — 187 с.
3. Абрамян М. Э., Михалкович С. С. Основы программирования на языке Паскаль: Скалярные типы данных, управляющие операторы, процедуры и функции. — Ростов-на-Дону: Изд-во «ЦВВР», 2004. — 198 с.
4. Абрамян М.Э. 1000 задач по программированию. Часть I (Скалярные типы данных, управляющие операторы, процедуры и функции). - Ростов-на-Дону, 2004. – 43 С.
5. Абрамян М.Э. 1000 задач по программированию. Часть II (Минимумы и максимумы, одномерные и двумерные массивы, символы и строки, двоичные файлы). - Ростов-на-Дону, 2004. – 42 С.
6. Абрамян М.Э. 1000 задач по программированию. Часть III (Текстовые файлы, составные типы данных в процедурах и функциях, рекурсия, указатели и динамические структуры). - Ростов-на-Дону, 2004. – 43 С.
7. Арабов М.К. Основы программирования на С++. - Душанбе. РТСУ, 2018. - 400с.
8. Арабов М.К. Практикум по программированию: учебное пособие для студентов – бакалавров и магистрантов, изучающих информационные технологии и программирование /М.К. Арабов, Б.М. Замонов Душанбе: РТСУ, 2019. 370 с.
9. Арабов М.К. Современные технологии Java /М.К. Арабов. Душанбе: РТСУ, 2020. - 293 с.
10. Арабов М.К., Каландаров Х.Х. Программирование на языке Python: учебное пособие для студентов- бакалавров с 2 курса направления подготовки «Прикладная информатика» / М.К. Арабов, Х.Х. Каландаров. Душанбе: РТСУ, 2021. - 261 с.
11. Арабов М.К., Нуров И.Дж. Основы программирования на языке PascalABC.NET. Учебно-методическое пособие. – Душанбе: РТСУ; 2018. - 280 с.
12. Габасов Р., Кириллова Ф. М. Основы динамического программирования. — Мн.: Изд-во БГУ, 1975. — 262 с.
13. Гамма, Э., Хелм, Р., Джонсон, Р., Влиссидес, Дж. Приемы объектно-ориентированного проектирования. Паттерны проектирования = Design Patterns: Elements of Reusable Object-Oriented Software. — 3-е издание. — М.: Вильямс, 2015. — 368 с. — ISBN 978-5-496-00389-6.
14. Голуб А. И. С и С++. Правила программирования. - М: БИНОМ, 1996 г. - 272 с.
15. Гранд, М. Шаблоны проектирования в JAVA. Каталог популярных шаблонов проектирования, проиллюстрированных при помощи UML = Patterns in Java, Volume 1. A Catalog of Reusable Design Patterns Illustrated with UML. — М.: Новое знание, 2004. — 560 с. — ISBN 5-94735-047-5.
16. Гукин Д. Язык программирования Си для «чайников».— М.: Диалектика, 2006. — 352 с.
17. Дейтел Х., Дейтел П. Как программировать на С++.— М.: БИНОМ, 1998 - 1024 с.
18. Зандстра, М. PHP. Объекты, шаблоны и методики программирования = PHP Objects, Patterns, and Practice. — 5-е изд. — СПб.: Диалектика, 2019. — 736 с. — ISBN 978-5-907144-54-5.
19. Ларман, К. Применение UML 2.0 и шаблонов проектирования = Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development. — М.: Вильямс, 2006. — 736 с. — ISBN 0-13-148906-2.
20. М. Арабов, М. Халилова. Маҷмӯи мисол ва масъалаҳо аз барномасозӣ. Душанбе. ДСРТ, 2018. - 138 с.
21. М. Арабов, М. Халилова. Сборник примеров и задач по программированию. - Душанбе. РТСУ, 2018. - 138 с.
22. Мак-Колм Смит, Дж. Элементарные шаблоны проектирования = Elemental Design Patterns. — М.: Вильямс, 2012. — 304 с. — ISBN 978-5-8459-1818-5.
23. Макконнелл, С. Совершенный код = Code Complete. — СПб.: Питер, 2005. — 896 с. — (Мастер-класс). — ISBN 5-7502-0064-7, 5-469-00822-3.
24. Муллошараф Арабов. Основы программирования на языке PascalABC.NET. - Душанбе. РТСУ - 2018. - 302с.
25. Фаулер, М. Рефакторинг кода на JavaScript: улучшение проекта существующего кода. — 2-е изд. — СПб.: Диалектика, 2019. — 464 с. — ISBN 978-5-907144-59-0.
26. Фаулер, М. Шаблоны корпоративных приложений (Signature Series) = Patterns of Enterprise Application Architecture. — М.: Вильямс, 2012. — 544 с. — ISBN 978-5-8459-1611-2.
27. Фаулер, М., Бек, К., Брант, Дж., Опдайк, У., Робертс, Д. Рефакторинг: улучшение проекта существующего кода. — М.: Диалектика, 2019. — 448 с. — ISBN 978-5-9909445-1-0.
28. Шилдт, Г. C# 4.0: полное руководство = C# 4.0 The Complete Reference. — М.: Вильямс, 2011. — 1056 с. — ISBN 978-5-8459-1684-6.
29. Шилдт, Г. C++: базовый курс, 3-е издание = C++ from the Ground Up, 3rd Edition. — М.: Вильямс, 2012. — 624 с. — ISBN 978-5-8459-0768-4.
30. Шилдт, Г. C++: методики программирования = Herb Schildt's C++ Programming Cookbook. — М.: Вильямс, 2008. — 480 с. — ISBN 978-5-8459-1494-1.
31. Шилдт, Г. Java 8. Полное руководство, 9-е издание = Java 8. The Complete Reference, 9th Edition. — М.: Вильямс, 2015. — 1376 с. — ISBN 978-5-8459-1918-2.
32. Шилдт, Г. Java: методики программирования = Schildt's Java Programming Cookbook. — М.: Вильямс, 2008. — 512 с. — ISBN 978-5-8459-1395-1.
33. Шилдт, Г. Java: руководство для начинающих, 5-е издание = Java: A Beginner's Guide, 5th Edition. — М.: Вильямс, 2012. — 624 с. — ISBN 978-5-8459-1770-6.
34. Шилдт, Г. Библиотека Swing для Java: руководство для начинающих = Swing: A Beginner's Guide. — М.: Вильямс, 2007. — 704 с. — ISBN 978-5-8459-1162-9.
35. Шилдт, Г. Полный справочник по C++, 4-е издание = C++: The Complete Reference, 4th Edition. — М.: Вильямс, 2011. — 800 с. — ISBN 978-5-8459-0489-8.
36. Шилдт, Г. Справочник программиста по C/C++, 3-е издание = C/C++ Programmer's Reference, 3rd Edition. — М.: Вильямс, 2006. — 432 с. — ISBN 978-5-8459-1622-8.
37. <http://cppstudio.com>
38. <http://golang-book.ru/>
39. [https://github.com/](https://github.com/swaroopch/byte_of_python)
40. https://metanit.com/
41. <https://proglib.io/p/30-golang-books/>
42. [https://pythonworld.ru/](https://pythonworld.ru/samouchitel-python)