Министерство науки и высшего образования
Российской Федерации
Федеральное государственное бюджетное
образовательное учреждение
высшего образования
Национальный исследовательский ядерный университет
«МИФИ»

Институт интеллектуальных кибернетических систем

Кафедра №22 «Кибернетика»

ОТЧЁТ

К РАБОТЕ ПО ДИСЦИПЛИНЕ
Технология промышленной разработки ПО

НА ТЕМУ:
«Лабораторная работа №2»

Студент М23–524 _____ Леонов В. В.
(Группа) (Подпись, дата) (И. О. Фамилия)

Преподаватель _____ Гагарин А. П.
(Подпись, дата) (И. О. Фамилия)

Москва, 2024 г.

СОДЕРЖАНИЕ

1 Постановка задачи

Условие: конкретная специализация Регистратора ресурсов характеризуется сочетанием следующих основных независимых параметров:

- c — ёмкость ресурса;
- v — цена ресурса (за единицу времени);
- s — размер запроса;
- pr — приоритет запроса;
- wt — приемлемое время ожидания запроса на обслуживание (если это время превышается, то запрос снимается);
- tu — необходимая продолжительность обслуживания запроса;
- pbu — вероятность отказа ресурса во время использования;
- pbi — вероятность отказа свободного ресурса.

Вариант 21. Диверсификация запросов по времени нахождения в очереди:

- c = 1;
- v = 0;
- s = 1;
- pr = 0;
- wt не ограниченно;
- tu — переменная y > 0;
- pbu = 0;
- pbi = 0.

Запрос располагается в очереди ожидания в зависимости от продолжительности нахождения в ней.

2 Ход работы

2.1 Концептуальная модель приложения

Построение таблицы прецедентов (Use Case Table) — это важный шаг в анализе требований и проектировании программного обеспечения. Такая таблица помогает систематизировать и документировать функциональные требования к системе, показывая взаимодействие пользователей (актеров) с системой для выполнения определенных действий (прецедентов).

В рамках решаемой задачи предлагается рассмотреть следующие прецеденты (таблицы 2.1–2.3).

Таблица 2.1 – Прецедент «Добавление запроса»

Название	**Добавление запроса**
Исполнители	Клиент, менеджер
Предусловие	Система работает (ресурсы инициализированы)
Описание	**Сценарий 1** Клиент формирует запрос на занятие ресурса. Запрос содержит информацию о существующем ресурсе. Ресурс свободен. Запрос в обработке. Занятие ресурса.
Постусловие	Ресурс успешно занят.
Предусловие	Система работает (ресурсы инициализированы)
Описание	**Сценарий 2** Клиент формирует запрос на занятие ресурса. Запрос содержит информацию о существующем ресурсе. Ресурс занят. Добавление запроса в очередь запросов.
Постусловие	Запрос добавлен в очередь.
Предусловие	Система работает (ресурсы инициализированы)

Таблица 2.1 – Прецедент «Добавление запроса» (продолжение)

Название	**Добавление запроса**
Описание	**Сценарий 3** Клиент формирует запрос на занятие ресурса. Запрос содержит информацию о несуществующем ресурсе. Вывод сообщения о ошибке.
Постусловие	Запрос обработан.

Таблица 2.2 – Прецедент «Освобождение ресурса по таймеру»

Название	**Освобождение ресурса по таймеру**
Исполнители	Менеджер
Предусловие	Система работает (ресурсы инициализированы). Истекло время использования ресурса. В очереди запросов есть запрос на занятие данного ресурса.
Описание	**Сценарий 1** Изменение статуса ресурса. Удаление запроса из текущих запросов. Начать обработку наиболее раннего запроса на использование ресурса. Новый запрос в обработке. Занятие ресурса.
Постусловие	Запрос обработан. Ресурс успешно занят.
Предусловие	Система работает (ресурсы инициализированы). Истекло время использования ресурса. В очереди запросов нет запроса на занятие данного ресурса.
Описание	**Сценарий 2** Изменение статуса ресурса. Удаление запроса из текущих запросов.
Постусловие	Запрос обработан. Ресурс освобожден.

Таблица 2.3 – Прецедент «Освобождение ресурса по запросу»

Название	**Освобождение ресурса по запросу**
Исполнители	Клиент, Менеджер
Предусловие	Система работает (ресурсы инициализированы). Ресурс занят. В очереди запросов есть запрос на занятие данного ресурса.
Описание	**Сценарий 1** Клиент формирует запрос на освобождение ресурса. Запрос содержит информацию о существующем ресурсе. Изменение статуса ресурса. Удаление запроса из текущих запросов. Начать обработку наиболее раннего запроса на использование ресурса. Новый запрос в обработке. Занятие ресурса.
Постусловие	Запрос обработан. Ресурс успешно занят.
Предусловие	Система работает (ресурсы инициализированы). Ресурс занят. В очереди запросов нет запроса на занятие данного ресурса.
Описание	**Сценарий 2** Клиент формирует запрос на освобождение ресурса. Запрос содержит информацию о существующем ресурсе. Изменение статуса ресурса. Удаление запроса из текущих запросов.
Постусловие	Запрос обработан. Ресурс освобожден.
Предусловие	Система работает (ресурсы инициализированы). Ресурс свободен.

Таблица 2.3 – Прецедент «Освобождение ресурса по запросу» (продолжение)

Название	**Освобождение ресурса по запросу**
Описание	**Сценарий 3** Клиент формирует запрос на освобождение ресурса. Запрос содержит информацию о существующем ресурсе. Вывод сообщения об ошибке.
Постусловие	Запрос обработан.
Предусловие	Система работает (ресурсы инициализированы).
Описание	**Сценарий 4** Клиент формирует запрос на освобождение ресурса. Запрос содержит информацию о несуществующем ресурсе. Вывод сообщения об ошибке.
Постусловие	Запрос обработан.

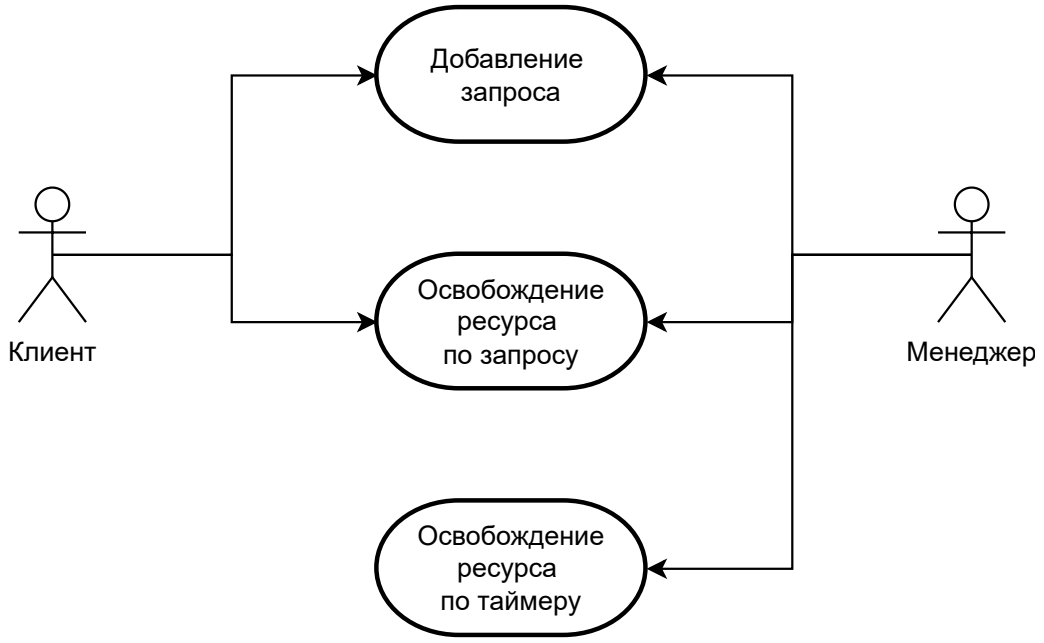

Рисунок 2.1 – Диаграмма прецедентов

Диаграмма прецедентов для рассматриваемого приложения представляет собой схему, которая описывает основные взаимодействия пользователей

с системой (см. рисунок 2.1). В данном примере мы рассматриваем систему управления ресурсами, где основными актерами являются Клиент и Менеджер. Диаграмма прецедентов отражает сценарии взаимодействия этих актеров с системой для выполнения различных операций.

На рисунке 2.4 приведена концептуальная диаграмма классов приложения. В соответствии с решаемой задачей потребовалось изменить базовую концептуальную диаграмму классов:

- добавление класса **Запрос**, который содержит статус, идентификатор ресурса и время его использования.
- добавление класса **Очередь запросов**, который содержит набор запросов в зависимости от времени их прибытия в систему.

Предполагается, что Клиент не может взаимодействовать с системой напрямую, а функционирует через Менеджера, который оперирует всеми системными сущностями.

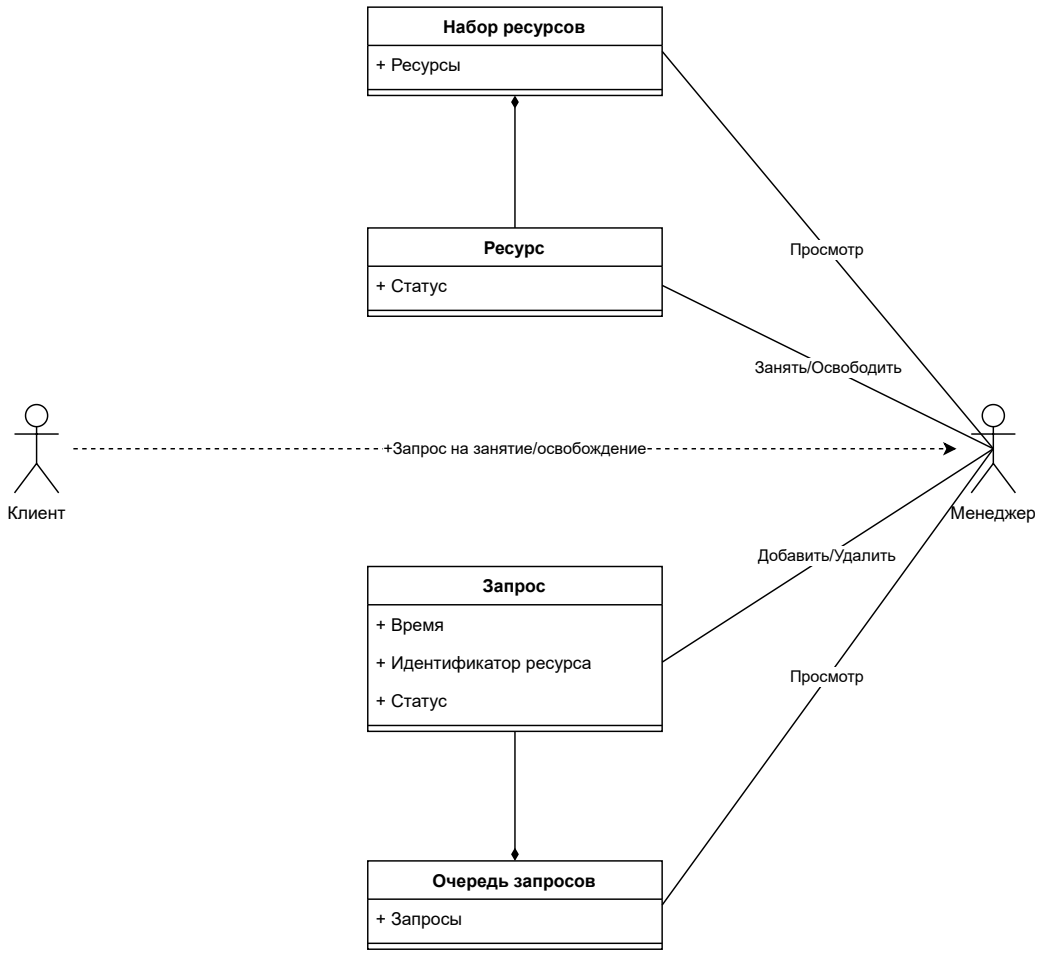

Рисунок 2.2 – Концептуальная диаграмма классов

Менеджер выдаёт компьютеру команды и получает информацию об их выполнении. В модели это взаимодействие представлено интерфейсом. Команды определены как его операции и доступны в окне свойств. Взаимодействие между компьютером и менеджером более детально показано на диаграмме системных взаимодействий на рисунке 2.3.

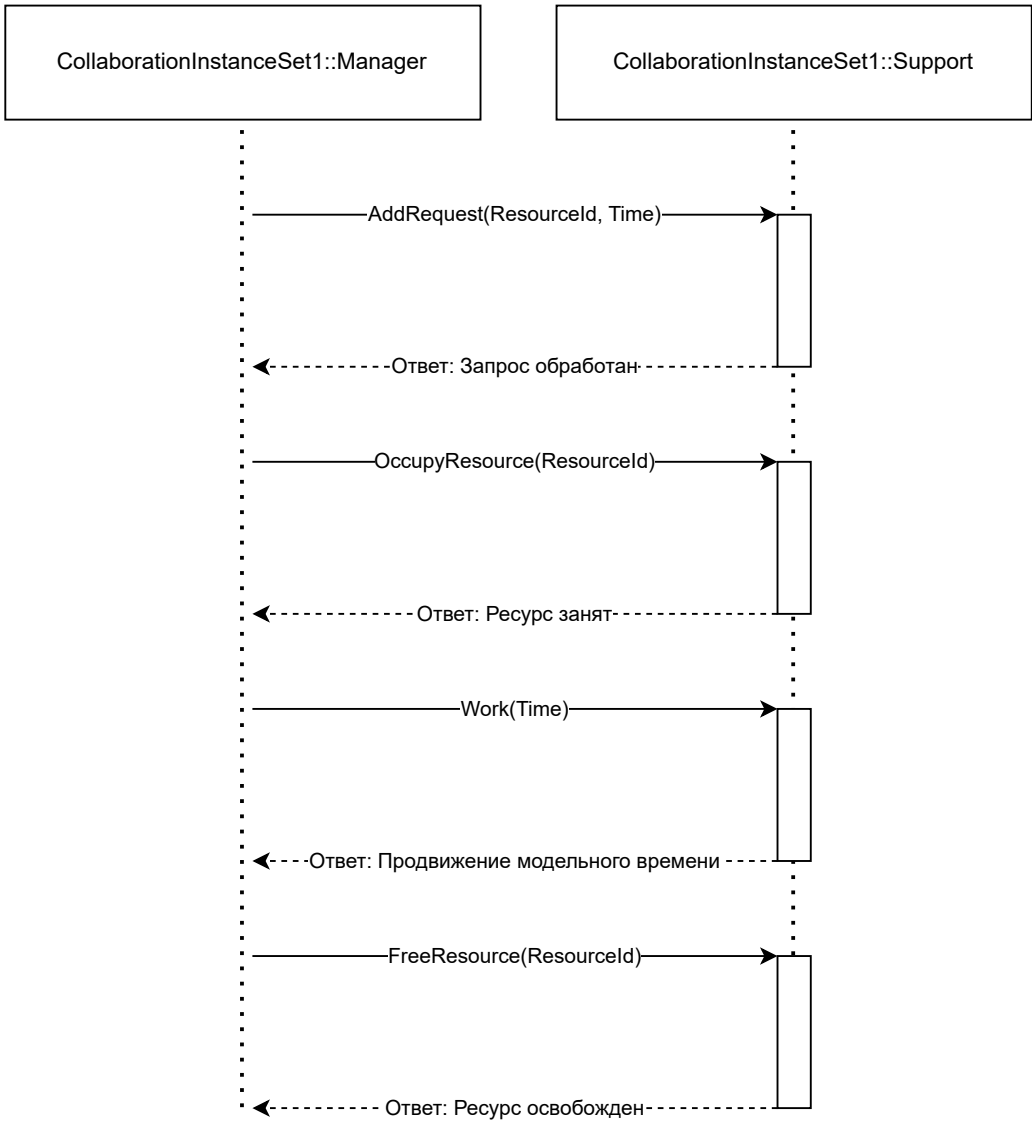

Рисунок 2.3 – Диаграмма системных взаимодействий

2.2 Архитектурное проектирование приложения

На основе концептуальной модели, построенной ранее, была разработана диаграмма проектных классов.

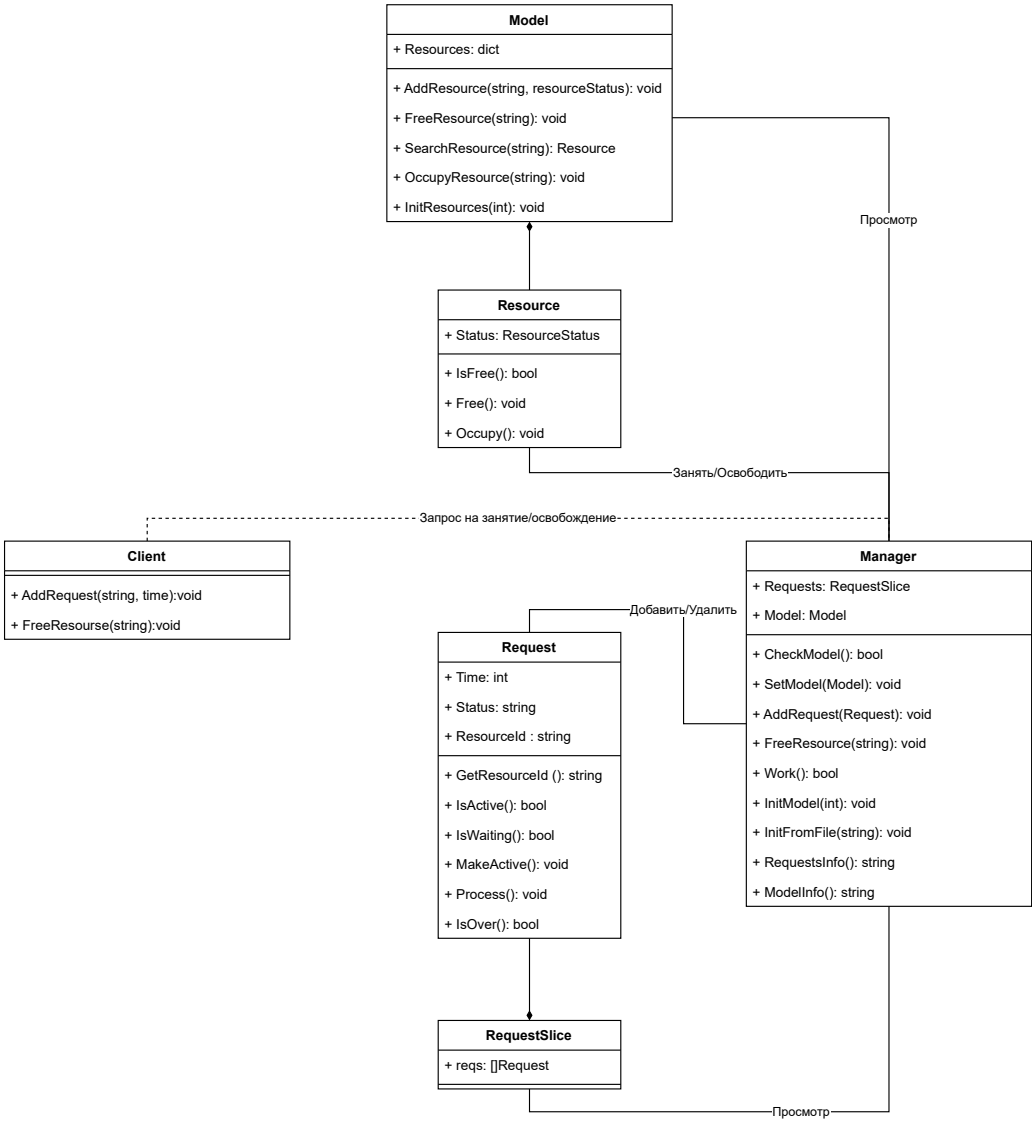

Рисунок 2.4 – Диаграмма классов

2.3 Детальное проектирование приложения в форме реинжиниринга

За основу приложения будет взят предложенный файл проекта на языке C#. Первым шагом выполнена структурная реорганизация и использование подхода многофайлового проекта:

- файл **main.cs** содержит точку входа в программу, а также реализует интерфейс, через который происходит взаимодействие с системой;
- модуль **model**:
  - файл **errors** описывает ошибки, которые могут возникнуть в рамках данного модуля;
  - файл **io.cs** описывает вспомогательные функции ввода/вывода;
  - файл **model.cs** описывает класс модели;
  - файл **resource.cs** описывает класс ресурса.
- модуль **manager**:
  - файл **manager.cs** описывает класс менеджера;
  - файл **request.cs** описывает класс запроса.

main.cs

Метод **MenuPrint** выводит на экран меню с опциями для пользователя. Меню структурировано и отображается последовательно, что делает его легко читаемым и понятным для пользователя. Каждый пункт меню четко обозначен, что упрощает навигацию и выбор нужной опции. Пользователь может быстро выбрать нужный пункт, введя соответствующий номер. Это минимизирует ошибки при вводе и ускоряет взаимодействие с программой.

Метод **MenuHandle** обрабатывает ввод пользователя и выполняет соответствующие действия на основе выбранной опции.

Метод **Main** — главный метод программы, который инициализирует менеджер ресурсов и запускает основной цикл для взаимодействия с пользователем.

```
static void MenuPrint()
{
    Console.WriteLine("Меню:");
    Console.WriteLine("\t1) Создать ресурсы");
    Console.WriteLine("\t2) Загрузить значения ресурсов из файла");
    Console.WriteLine("\t3) Добавить запрос");
    Console.WriteLine("\t4) Освободить ресурс");
    Console.WriteLine("\t5) Такт");
    Console.WriteLine("\t6) Информация о ресурсах");
    Console.WriteLine("\t7) Информация о запросах");
    Console.WriteLine("\t0) Выход");
}
```

```
static void MenuHandle(Manager manager, int option)
{
    switch (option)
    {
        case 1:
            Console.Write("\tВведите число ресурсов: ");
            if (int.TryParse(Console.ReadLine(), out int resCnt))
            {
                manager.InitModel(resCnt);
            }
            else
            {
                Console.WriteLine("Некорректный ввод, не число.");
            }
            break;
        ...
```

```
static void Main()
{
    var manager = new Manager(new Model());
    while (true)
    {
        MenuPrint();
        Console.Write("Выберите опцию: ");
        if (int.TryParse(Console.ReadLine(), out int option))
        {
            MenuHandle(manager, option);
        }
        else
        {
            Console.WriteLine("Некорректный ввод, не число.");
        }
    }
}
```

model/errors.cs

Файл **errors.cs** является частью проекта и находится в пространстве имен **ResourceRegistrator.Model**. Этот файл содержит статический класс

Errors, который используется для хранения строковых констант, представляющих различные сообщения об ошибках, которые могут возникнуть в процессе работы программы.

Класс **Errors** является статическим, что означает, что его нельзя создавать в экземпляре, и все его члены также должны быть статическими. Данный класс предназначен для того, чтобы сделать обработку ошибок в коде более удобной и централизованной. Вместо того чтобы вручную вводить строки сообщений об ошибках в разных частях кода, разработчики могут использовать константы из этого класса. Это улучшает читаемость и поддерживаемость кода, а также упрощает процесс внесения изменений в тексты сообщений об ошибках.

```
1  namespace ResourceRegistrator.Model
2  {
3      public static class Errors
4      {
5          public const string ResourceNotFound = "Ресурс не найден.";
6          public const string ResourceIsBusy = "Ресурс занят.";
7          public const string IncorrectFile = "Некорректный файл.";
8      }
9  }
```

model/io.cs

Класс **Io** является статическим и содержит методы расширения для работы с объектом типа Model, связанные с инициализацией модели ресурсов из файла. Метод **InitFromFile** открывает файл по указанному пути с помощью StreamReader и последовательно читает его построчно. Каждая строка файла разбивается на части (тег ресурса и его статус) с использованием разделителя пробела. Метод проверяет, что каждая строка содержит ровно две части. В противном случае выбрасывается исключение с сообщением об ошибке. В зависимости от значения второй части строки присваивается статус ресурса (Free или Occupied). Если статус не распознан, выбрасывается исключение. Метод добавляет ресурс в модель с использованием метода AddResource.

```
namespace ResourceRegistrator.Model
{
    public static class Io
    {
        public static void InitFromFile(this Model model, string filePath)
        {
            using (var file = new StreamReader(filePath))
            {
                string line;
                while ((line = file.ReadLine()) != null)
                {
                    var parts = line.Split(" ");
                    if (parts.Length != 2)
                    {
                        throw new Exception(Errors.IncorrectFile);
                    }
                    var resourceTag = parts[0].Trim();
                    var resourceStatus = parts[1].Trim();

                    ResourceStatus status;
                    switch (resourceStatus)
                    {
                        case nameof(ResourceStatus.Free):
                            status = ResourceStatus.Free;
                            break;
                        case nameof(ResourceStatus.Occupied):
                            status = ResourceStatus.Occupied;
                            break;
                        default:
                            throw new Exception(Errors.IncorrectFile);
                    }
                    model.AddResource(resourceTag, status);
                }
            }
        }
    }
}
```

model/model.cs

Файл **Model.cs** является частью проекта и находится в пространстве имен ResourceRegistrator.Model. Этот файл содержит класс Model, который отвечает за управление ресурсами в системе. Класс **Model** включает в себя словарь resources, который хранит ресурсы, идентифицированные по строковым ключам. Основная задача этого класса — управление состояниями ресурсов, включая их добавление, освобождение, поиск и занятие.

Метод **AddResource** добавляет новый ресурс в словарь resources. Этот метод принимает два параметра: имя ресурса и его начальный статус. Внутри метода создается новый объект Resource с указанным статусом, который затем

добавляется в словарь под соответствующим именем. Метод **FreeResource** освобождает ресурс с указанным именем. Он проверяет наличие ресурса в словаре и, если ресурс найден, вызывает метод Free у объекта ресурса, чтобы изменить его статус на свободный. Если ресурс не найден, выбрасывается исключение с сообщением об ошибке. Метод **SearchResource** осуществляет поиск ресурса по его имени и возвращает объект ресурса, если он найден. Если ресурс не найден, метод возвращает null.

```
1  namespace ResourceRegistrator.Model
2  {
3      public class Model
4      {
5          private Dictionary<string, Resource> resources = new Dictionary<string,
           ↪  Resource>();
6          public void AddResource(string nameTag, ResourceStatus status)
7          {
8              var resource = new Resource(status);
9              resources[nameTag] = resource;
10         }
11         public void FreeResource(string nameTag)
12         {
13             if (!resources.TryGetValue(nameTag, out var resource))
14             {
15                 throw new Exception(Errors.ResourceNotFound);
16             }
17             resource.Free();
18             resources[nameTag] = resource;
19         }
20         public Resource SearchResource(string nameTag)
21         {
22             if (resources.TryGetValue(nameTag, out var resource))
23             {
24                 return resource;
25             }
26             return null;
27         }
28
```

Метод **OccupyResource** предназначен для занятия ресурса с указанным именем. Он проверяет наличие ресурса в словаре и, если ресурс найден, вызывает метод Occupy у объекта ресурса, чтобы изменить его статус на занятый. Если ресурс не найден, выбрасывается исключение с сообщением об ошибке. Метод **InitResources** инициализирует заданное количество ресурсов. Он принимает один параметр - количество ресурсов для инициализации. Внутри метода в цикле вызывается метод AddResource для добавления указанного количества ресурсов с уникальными именами и статусом Free.

```
        public void OccupyResource(string nameTag)
        {
            if (!resources.TryGetValue(nameTag, out var resource))
            {
                throw new Exception(Errors.ResourceNotFound);
            }
            resource.Occupy();
            resources[nameTag] = resource;
        }
        public void InitResources(int resourceCount)
        {
            for (int i = 0; i < resourceCount; i++)
            {
                AddResource($"res{i}", ResourceStatus.Free);
            }
        }
```

Метод **ToString** переопределяет стандартный метод преобразования объекта в строку. Он создает и возвращает строку, содержащую список всех ресурсов и их статусов. Метод **IsEmpty** проверяет, пуст ли словарь ресурсов, и возвращает true, если ресурсов нет, и false в противном случае. Класс Model предназначен для управления коллекцией ресурсов, предоставляя методы для их добавления, поиска, освобождения и занятия.

```
        public override string ToString()
        {
            var sb = new StringBuilder();
            foreach (var resource in resources)
            {
                sb.AppendLine($"{resource.Key}: {resource.Value.Status}");
            }
            return sb.ToString();
        }
        public bool IsEmpty()
        {
            return resources.Count == 0;
        }
    }
}
```

Это ключевой компонент системы, который обеспечивает централизованное управление состояниями ресурсов и упрощает взаимодействие с ними в других частях приложения.

model/resource.cs

Файл **Resource.cs** является частью проекта и находится в пространстве имен ResourceRegistrator.Model. Этот файл содержит два основных компонента: перечисление **ResourceStatus** и класс **Resource**. Перечисление ResourceStatus определяет два возможных состояния ресурса: Free (свободный) и Occupied (занятый). Эти состояния используются для управления доступностью ресурсов в системе.

Класс Resource представляет собой модель ресурса с определенным статусом. В нем определено одно приватное поле Status типа ResourceStatus, которое указывает текущий статус ресурса. Это поле доступно только для чтения извне класса, но может изменяться методами самого класса. Конструктор класса Resource принимает параметр status типа ResourceStatus и инициализирует поле Status переданным значением. Таким образом, при создании нового объекта Resource его статус задается явно.

```
1  namespace ResourceRegistrator.Model
2  {
3      public enum ResourceStatus
4      {
5          Free,
6          Occupied
7      }
8
9      public class Resource
10     {
11         public ResourceStatus Status { get; private set; }
12
13         public Resource(ResourceStatus status)
14         {
15             Status = status;
16         }
17
18
```

Метод IsFree возвращает логическое значение, указывающее, находится ли ресурс в свободном состоянии. Если статус ресурса равен ResourceStatus.Free, метод возвращает true, в противном случае false. Метод Free изменяет статус ресурса на Free. Этот метод используется для освобождения ресурса, делая его доступным для других запросов. Метод Occupy изменяет статус ресурса на Occupied, если он в данный момент свободен. Внутри метода происходит проверка текущего статуса ресурса с использова-

нием метода IsFree. Если ресурс свободен, его статус изменяется на Occupied. Если ресурс уже занят, выбрасывается исключение с сообщением об ошибке, используя константу из класса Errors.

```
        public bool IsFree()
        {
            return Status == ResourceStatus.Free;
        }

        public void Free()
        {
            Status = ResourceStatus.Free;
        }

        public void Occupy()
        {
            if (IsFree())
            {
                Status = ResourceStatus.Occupied;
            }
            else
            {
                throw new Exception(Errors.ResourceIsBusy);
            }
        }
    }
}
```

Класс Resource предназначен для управления состоянием отдельного ресурса в системе. Он предоставляет методы для проверки текущего статуса ресурса, а также для изменения его состояния на свободный или занятый. Этот класс является ключевым компонентом модели данных и используется другими частями системы для управления ресурсами. Перечисление ResourceStatus и класс Resource вместе обеспечивают базовую функциональность для управления доступностью ресурсов в системе, что позволяет эффективно распределять и контролировать использование ресурсов в приложении.

manager/manager.cs

Файл **Manager.cs** является частью проекта и находится в пространстве имен ResourceRegistrator.Manager. Этот файл содержит класс Manager, который отвечает за управление запросами на ресурсы и взаимодействие с моделью данных ресурсов. Класс Manager включает в себя два основных свойства: Requests и Model. Requests представляет собой коллекцию запросов на ресурсы, а Model содержит текущую модель ресурсов.

Конструктор класса Manager принимает объект модели Model в качестве параметра и инициализирует свойство Requests новым объектом RequestSlice. Это позволяет создать менеджер с пустым списком запросов и заданной моделью ресурсов. Метод CheckModel проверяет, пуста ли модель ресурсов, вызывая соответствующий метод у объекта Model. Это используется для определения, были ли инициализированы ресурсы в модели.

```
namespace ResourceRegistrator.Manager
{
    public class Manager
    {
        public RequestSlice Requests { get; set; }
        public Model.Model Model { get; set; }

        public Manager(Model.Model model)
        {
            Requests = new RequestSlice();
            Model = model;
        }

        public bool CheckModel()
        {
            return Model.IsEmpty();
        }
```

Метод SetModel позволяет заменить текущую модель ресурсов новой моделью, предоставленной в качестве параметра. Это может быть полезно, когда нужно переключить контекст на другую модель ресурсов. Метод AddRequest добавляет новый запрос в коллекцию запросов. Он сначала ищет ресурс по идентификатору, указанному в запросе, и если ресурс не найден, выбрасывает исключение. Если ресурс найден, запрос добавляется в коллекцию запросов.

```
        public void SetModel(Model.Model model)
        {
            Model = model;
        }

        public void AddRequest(Request request)
        {
            var resource = Model.SearchResource(request.ResourceId);
            if (resource == null)
            {
                throw new Exception(Errors.ResourceNotFound);
            }
            Requests.Add(request);
        }
```

Метод FreeResource освобождает ресурс с указанным идентификатором, вызывая соответствующий метод у объекта Model. После этого он ищет активный запрос на этот ресурс в коллекции запросов и удаляет его, если такой запрос найден. Метод Work выполняет основной цикл обработки запросов. Он проходит по всем запросам в коллекции и пытается занять ресурсы для запросов, которые находятся в состоянии ожидания. Если ресурс занят, обработка продолжается для следующего запроса. Если запрос успешно занимает ресурс, его статус изменяется на "Active". Время выполнения каждого запроса уменьшается на единицу, и если время выполнения запроса истекло, ресурс освобождается. Запросы с истекшим временем выполнения удаляются из коллекции.

```
1         public void FreeResource(string resourceId)
2         {
3             Model.FreeResource(resourceId);
4             var req = Requests.FirstOrDefault(r => r.ResourceId == resourceId && r.Status
              ↪  == "Active");
5             if (req != null)
6             {
7                 Requests.Remove(req);
8             }
9         }
10
11        public void Work()
12        {
13            foreach (var r in Requests)
14            {
15                if (r.Status == "Waiting")
16                {
17                    try
18                    {
19                        Model.OccupyResource(r.ResourceId);
20                    }
21                    catch (Exception ex)
22                    {
23                        if (ex.Message == Errors.ResourceIsBusy)
24                        {
25                            continue;
26                        }
27                    }
28                    r.Status = "Active";
29                }
30                r.Time -= 1;
31                if (r.Time <= 0)
32                {
33                    Model.FreeResource(r.ResourceId);
34                }
35            }
36            Requests.RemoveAll(r => r.Time <= 0);
37        }
```

Метод InitModel инициализирует модель ресурсов, создавая заданное количество ресурсов. Метод InitFromFile загружает конфигурацию модели ресурсов из файла, вызывая соответствующий метод у объекта Model. Методы RequestsInfo и ModelInfo возвращают строковое представление текущего состояния запросов и модели ресурсов соответственно. Эти методы позволяют получить текстовую информацию о текущем состоянии системы для вывода на экран или логирования.

```
        public void InitModel(int resourceCount)
        {
            Model.InitResources(resourceCount);
        }

        public void InitFromFile(string path)
        {
            Model.InitFromFile(path);
        }

        public string RequestsInfo()
        {
            return Requests.ToString();
        }

        public string ModelInfo()
        {
            return Model.ToString();
        }
    }
}
```

Класс Manager предназначен для управления запросами на ресурсы и их состояниями. Он взаимодействует с моделью ресурсов для выполнения операций добавления, освобождения и использования ресурсов, а также управляет коллекцией запросов, обеспечивая их корректную обработку и выполнение.

manager/request.cs

Файл **Request.cs** также содержит класс Request и вложенный класс RequestSlice, которые являются важными компонентами системы управления ресурсами.

Класс Request представляет собой модель запроса на использование ресурса. Он содержит три свойства: ResourceId, Time и Status. ResourceId представляет идентификатор ресурса, к которому относится запрос. Time указывает на продолжительность времени, в течение которого ресурс бу-

дет использоваться. Status отображает текущее состояние запроса и может принимать значения "Waiting"(ожидает) или "Active"(активный).

Конструктор класса Request принимает два параметра: resourceId и time, и инициализирует соответствующие свойства, устанавливая статус запроса в "Waiting". Метод GetResourceId возвращает идентификатор ресурса, к которому относится запрос. Методы IsActive и IsWaiting проверяют текущий статус запроса, возвращая логическое значение, указывающее, находится ли запрос в активном состоянии или в ожидании соответственно. Метод MakeActive переводит статус запроса в "Active".

```
namespace ResourceRegistrator.Manager
{
    public class Request
    {
        public string ResourceId { get; set; }
        public int Time { get; set; }
        public string Status { get; set; }

        public Request(string resourceId, int time)
        {
            ResourceId = resourceId;
            Time = time;
            Status = "Waiting";
        }

        public string GetResourceId()
        {
            return ResourceId;
        }

        public bool IsActive()
        {
            return Status == "Active";
        }

        public bool IsWaiting()
        {
            return Status == "Waiting";
        }

        public void MakeActive()
        {
            Status = "Active";
        }
```

Метод Process уменьшает оставшееся время выполнения запроса на единицу, моделируя прохождение времени в системе. Метод IsOver проверяет, завершено ли выполнение запроса, возвращая true, если оставшееся время выполнения меньше или равно нулю. Метод ToString переопределяет

стандартный метод преобразования объекта в строку, возвращая строковое представление запроса с его текущими значениями свойств.

```
        public void Process()
        {
            Time -= 1;
        }

        public bool IsOver()
        {
            return Time <= 0;
        }

        public override string ToString()
        {
            return $"\tRequest ResourceId={ResourceId}, Time Left={Time},
            ↪  Status={Status}";
        }
    }
```

Класс RequestSlice является производным от List<Request> и представляет собой коллекцию запросов. Он переопределяет метод ToString, чтобы возвращать строковое представление всех запросов в коллекции. Метод создает список строковых представлений каждого запроса, объединяет их и возвращает в виде форматированной строки.

```
    public class RequestSlice : List<Request>
        {
            public override string ToString()
            {
                var strs = new List<string>(this.Count);
                foreach (var r in this)
                {
                    strs.Add(r.ToString());
                }
                return $"[\n{string.Join("\n", strs)}\n]";
            }
        }
}
```

Эти классы играют ключевую роль в управлении запросами на ресурсы в системе. Класс Request моделирует отдельный запрос, обеспечивая методы для управления его состоянием и отслеживания времени выполнения. Класс RequestSlice предоставляет удобный способ хранения и обработки множества запросов, а также позволяет легко получать их строковое представление для вывода на экран или логирования. Эти компоненты интегрируются с

классом Manager, который управляет их созданием, обновлением и удалением в соответствии с текущим состоянием системы и бизнес-логикой приложения.

Далее приведена трасса вывода, демонстрирующая работу ПО.

```
Меню:
        1) Создать ресурсы
        2) Загрузить значения ресурсов из файла
        3) Добавить запрос
        4) Освободить ресурс
        5) Такт
        6) Информация о ресурсах
        7) Информация о запросах
        0) Выход
Выберите опцию: 1
        Введите число ресурсов: 10
Меню:
        1) Создать ресурсы
        2) Загрузить значения ресурсов из файла
        3) Добавить запрос
        4) Освободить ресурс
        5) Такт
        6) Информация о ресурсах
        7) Информация о запросах
        0) Выход
Выберите опцию: 6
res0: Free
res1: Free
res2: Free
res3: Free
res4: Free
res5: Free
res6: Free
res7: Free
res8: Free
res9: Free

Меню:
        1) Создать ресурсы
        2) Загрузить значения ресурсов из файла
        3) Добавить запрос
        4) Освободить ресурс
        5) Такт
        6) Информация о ресурсах
        7) Информация о запросах
        0) Выход
Выберите опцию: 3
        Введите название ресурса: res1
        Введите продолжительность запроса: 5
Меню:
        1) Создать ресурсы
        2) Загрузить значения ресурсов из файла
        3) Добавить запрос
        4) Освободить ресурс
        5) Такт
        6) Информация о ресурсах
```

```
        7) Информация о запросах
        0) Выход
Выберите опцию: 3
        Введите название ресурса: res1
        Введите продолжительность запроса: 6
Меню:
        1) Создать ресурсы
        2) Загрузить значения ресурсов из файла
        3) Добавить запрос
        4) Освободить ресурс
        5) Такт
        6) Информация о ресурсах
        7) Информация о запросах
        0) Выход
Выберите опцию: 3
        Введите название ресурса: res1
        Введите продолжительность запроса: 7
Меню:
        1) Создать ресурсы
        2) Загрузить значения ресурсов из файла
        3) Добавить запрос
        4) Освободить ресурс
        5) Такт
        6) Информация о ресурсах
        7) Информация о запросах
        0) Выход
Выберите опцию: 7
[
        Request ResourceId=res1, Time Left=5, Status=Waiting
        Request ResourceId=res1, Time Left=6, Status=Waiting
        Request ResourceId=res1, Time Left=7, Status=Waiting
]
Меню:
        1) Создать ресурсы
        2) Загрузить значения ресурсов из файла
        3) Добавить запрос
        4) Освободить ресурс
        5) Такт
        6) Информация о ресурсах
        7) Информация о запросах
        0) Выход
Выберите опцию: 5
Меню:
        1) Создать ресурсы
        2) Загрузить значения ресурсов из файла
        3) Добавить запрос
        4) Освободить ресурс
        5) Такт
        6) Информация о ресурсах
        7) Информация о запросах
        0) Выход
Выберите опцию: 5
Меню:
        1) Создать ресурсы
        2) Загрузить значения ресурсов из файла
        3) Добавить запрос
        4) Освободить ресурс
```

```
        5) Такт
        6) Информация о ресурсах
        7) Информация о запросах
        0) Выход
Выберите опцию: 5
Меню:
        1) Создать ресурсы
        2) Загрузить значения ресурсов из файла
        3) Добавить запрос
        4) Освободить ресурс
        5) Такт
        6) Информация о ресурсах
        7) Информация о запросах
        0) Выход
Выберите опцию: 5
Меню:
        1) Создать ресурсы
        2) Загрузить значения ресурсов из файла
        3) Добавить запрос
        4) Освободить ресурс
        5) Такт
        6) Информация о ресурсах
        7) Информация о запросах
        0) Выход
Выберите опцию: 5
Меню:
        1) Создать ресурсы
        2) Загрузить значения ресурсов из файла
        3) Добавить запрос
        4) Освободить ресурс
        5) Такт
        6) Информация о ресурсах
        7) Информация о запросах
        0) Выход
Выберите опцию: 7
[
        Request ResourceId=res1, Time Left=5, Status=Active
        Request ResourceId=res1, Time Left=7, Status=Waiting
]
Меню:
        1) Создать ресурсы
        2) Загрузить значения ресурсов из файла
        3) Добавить запрос
        4) Освободить ресурс
        5) Такт
        6) Информация о ресурсах
        7) Информация о запросах
        0) Выход
Выберите опцию: 6
res0: Free
res1: Occupied
res2: Free
res3: Free
res4: Free
res5: Free
res6: Free
res7: Free
res8: Free
res9: Free

Выберите опцию:0
Завершение работы...
```