Приложение

**ФОНД ОЦЕНОЧНЫХ СРЕДСТВ**

ТЕХНОЛОГИИ ПРОМЫШЛЕННОЙ РАЗРАБОТКИ

ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ.

ТРЕБОВАНИЯ

**2 Семестр**

**Раздел 2 Реинжиниринг**

**2.1 Лабораторная работа (ЛР) - 11 Неделя**

**Лабораторная работа №2**

**«Технологии промышленной разработки программного обеспечения. Требования»   
(2 семестр)**

**Варианты заданий**

**Постановка задачи**

Выполнить реинжиниринг базового варианта прикладной программы «Регистратор ресурсов», далее Регистратора. Реинжиниринг понимается как переработка программы с целью изменения её функциональных и/или эксплуатационных параметров в указанных границах. Пояснения относительно процесса проектирования базового варианта Регистратора, включая основные документы проектирования, а также исходные тексты на языках C# и C++, приведены в качестве Приложения №1 к и предоставляются студенту.

Студент выбирает тему реинжиниринга из списка, в котором темы различаются параметрами, сформулированными в терминах функций базового варианта Регистратора.

**Минимальные требования к выполнению заданий**.

1) Должны быть разработаны концептуальная модель выбранного варианта Регистратора, включающая таблицу прецедентов, диаграммы прецедентов, концептуальных классов и системных взаимодействий. Разработку можно вести на основе соответствующих таблиц и диаграмм, содержащихся в описании базового варианта Регистратора.

2) Должно быть проведено архитектурное проектирование выбранного варианта Регистратора в форме разработки диаграмм(ы) проектных классов. Диаграмма классов должна быть доведена до уровня зависимости от платформы.

3) Должно быть проведено детальное проектирование выбранного варианта Регистратора в форме локальной переработки текста базового варианта Регистратора с обоснованием проведённых изменений исходного текста по каждому изменённому фрагменту текста.

4) Спроектированный вариант Регистратора должен быть отлажен.

5) Должен быть составлен план тестирования разработанного варианта Регистратора

6) Должно быть проведено тестирование разработанного варианта Регистратора по составленному плану и протокол тестирования включён в отчёт по лабораторной работе.

7) В рамках детального проектирования должен быть обеспечен подсчёт среднего времени использования ресурса, среднего времени ожидания запроса (если оно предусмотрено).

**Методические указания**.

Студенту рекомендуется согласовать концептуальную модель разрабатываемого варианта Регистратора (таблицу прецедентов, диаграммы прецедентов, концептуальных классов, системных взаимодействий) прежде, чем приступать к архитектурному проектированию, чтобы убедиться в корректном понимании темы.

**Выбор темы проектирования**

Конкретный вариант Регистратора ресурсов характеризуется сочетанием следующих основных независимых параметров:

c - «ёмкость» ресурса,

v – цена ресурса (за единицу времени),

s - «размер» запроса,

pr – приоритет запроса,

wt –время ожидания запроса на обслуживание,

tu – продолжительность обслуживания запроса (владения ресурсом),

pbu – вероятность отказа ресурса во время использования,

pbi - вероятность отказа свободного ресурса.

Примечание. Базовый вариант Регистратора ресурсов может быть характеризована следующим сочетанием параметров:

c = s = 1, v = 0, pr = 0, wt = 0, tu >=0, зависит от действий оператора , pbu = 0, pbi = 0.

**Темы проектирования (реинжиниринга)**

|  | Тема (вариант) | Характеристика варианта \*) | Балл |
| --- | --- | --- | --- |
| 1 | Диверсификация ёмкости ресурса и размера запроса при неделимости запроса | ресурсы и запросы разные, но для использования д. б. c >= s,  v = 0, pr = 0, wt = 0, pbu = 0, pbi = 0  Запрос не может быть разделён, чтобы пользоваться несколькими ресурсами. | 3 |
| 2 | Диверсификация ёмкости ресурса и размера запроса при делимости запроса без ожидания | ресурсы и запросы разные, но для использования д. б. c >= s,  v = 0, pr = 0, wt = 0, pbu = 0, pbi = 0  Запрос может быть разделён, чтобы пользоваться несколькими ресурсами, но отдельный ресурс обслуживает только один запрос (или его часть). | 6 |
| 3 | Диверсификация ёмкости ресурса и размера запроса при делимости запроса с ожиданием | ресурсы и запросы разные, но для использования д. б. c >= s,  v = 0, pr = 0, wt > 0, pbu = 0, pbi = 0  Запрос может быть разделён, чтобы пользоваться несколькими ресурсами, но отдельный ресурс обслуживает только один запрос (или его часть). | 6 |
| 4 | Диверсификация запросов по сроку предоставления ресурса со снятием | c = s = 1, v = 0, pr = 0, wt – переменная x > 0, tu – переменная y > 0, pbu = 0, pbi = 0  Запрос имеет срок T, не позднее которого он должен быть удовлетворён, то есть, владеет ресурсом не менее tu, иначе он снимается. | 9 |
| 5 | Диверсификация запросов по сроку предоставления ресурса с уведомлением | c = s = 1, v = 0, pr = 0, wt – переменная x > 0, tu – переменная y > 0, pbu = 0, pbi = 0  Запрос имеет срок T, не позднее которого он должен быть удовлетворён, то есть, владеет ресурсом не менее tu, иначе уведомляется заказчик, который может продлить ожидание. | 9 |
| 6 | Диверсификация запросов по ёмкости ресурса и сроку предоставления ресурса со снятием | c> = s, v = 0, pr = 0, wt – переменная x > 0, tu – переменная y > 0, pbu = 0, pbi = 0  Запрос имеет срок T, не позднее которого он должен быть удовлетворён, то есть, владеет ресурсом не менее tu, иначе он снимается. | 9 |
| 7 | Диверсификация запросов по важности с вытеснением | c = s = 1, v = 0, pr целое число m > 0, wt не ограниченно, pbu = 0, pbi = 0  Запрос может отнять ресурс у менее важного запроса, вытесненный запрос переходит в очередь ожидания | 9 |
| 8 | Диверсификация запросов по важностибез вытеснения | c = s = 1, v = 0, pr целое число m > 0, wt не ограниченно, tu – переменная y > 0, pbu = 0, pbi = 0  Запрос может «обогнать» менее важный запрос в очереди ожидания | 9 |
| 9 | Пользование ресурсом оплачивается по факту | c = s = 1, v – переменная z > 0, pr = 0, wt = 0, tu – переменная x > 0, pbu = 0, pbi = 0  Плата рассчитывается при освобождении ресурса | 3 |
| 10 | Пользование ресурсом оплачивается со скидкой | c = s = 1, v = 0, pr = 0, wt – переменная x > 0, tu – переменная x > 0, pbu = 0, pbi = 0  Запрос имеет срок T, не позднее которого он должен быть удовлетворён. то есть, владеет ресурсом не менее tu, иначе он снимается. Скидка определяется величиной rb = const за единицу времени нахождения в очереди. | 9 |
| 11 | Пользование ресурсом оплачивается по факту  с предпочтением более дешевого ресурса | c = s = 1, v – переменная z > 0, pr = 0, wt = 0, tu – переменная x > 0, pbu = 0, pbi = 0  При возможности выбора запросу назначается более дешёвый ресурс.  Плата рассчитывается при освобождении ресурса | 6 |
| 12 | Отказы и восстановление ресурсов во время пользования | c = s = 1, v = 0, pr = 0, wt = 0, tu –переменная x > 0, pbu – переменная y > 0, pbi = 0  Запрос отменяется с уведомлением заказчика. | 6 |
| 13 | Отказы и восстановление ресурсов во время пользования  с переводом запроса в очередь | c = s = 1, v = 0, pr = 0, wt переменная x > 0, tu –переменная x > 0, pbu – переменная y > 0, pbi = 0  Запрос переводится в очередь. | 6 |
| 14 | Мультизапросы с одновременным использованием ресурсов | c = 1, s = целое k > 0, v = 0, pr = 0, wt = 0, tu =1, pbu = 0, pbi = 0  Ресурсы одинаковые, но запрос выполняется только при предоставлении всех нужных ему k ресурсов. | 6 |
| 15 | Мультизапросы с последовательным использованием ресурсов | i – целое количество групп ресурсов,  j – целое количество ресурсов в группе, c - ёмкость всех ресурсов = 1,  k - целое количество запросов в мультизапросе > 0, s – размер всех запросов = 1, v = 0, pr = 0, wt не ограниченно, tu –переменная x>0, pbu = 0, pbi = 0  Ресурсы делятся на i групп по j специализированных ресурсов. Мультизапрос находится в очереди или в состоянии выполнения, пока все его запросы не будут удовлетворены. | 6 |
| 16 | Распределение ресурсов с учётом истории их использования | c = s = 1, v = 0, pr = 0, wt не ограниченно, tu – переменная y > 0, pbu = 0, pbi = 0  Например, назначается свободный ресурс с наименьшим временем использования за сеанс. | 9 |
| 17 | Отказы и восстановление ресурсов во время пользования | c = s = 1, v = 0, pr = 0, wt не ограниченно, tu –переменная x > 0, pbu – переменная y > 0, pbi = 0  Запрос переводится в «голову» очереди ожидания | 6 |
| 18 | Отказы и восстановление ресурсов во время пользования | c = s = 1, v = 0, pr = 0, wt = 0, tu –переменная x > 0, pbu – переменная y > 0, pbi = 0  Отказавший ресурс должен быть заменён из резерва. Количество резервных ресурсов определяется целочисленной константой k. При исчерпании резерва запрос отменяется. | 9 |
| 19 | Бронирование ресурсов | c = s = 1, v = 0, pr = 0, wt – переменная x > 0, tu – переменная y > 0, pbu = 0, pbi = 0  Запрос имеет срок T, не позднее и не ранее которого он должен быть удовлетворён, то есть, владеет ресурсом не менее tu, иначе он снимается. | 9 |
| 20 | Бронирование ресурсов c ожиданием | c = s = 1, v = 0, pr = 0, wt – переменная x > 0, tu – переменная y > 0, pbu = 0, pbi = 0  Запрос имеет срок T, не позднее и не ранее которого он должен быть удовлетворён, иначе он ставится в очередь неограниченного ожидания. | 9 |
| 21 | Диверсификация запросов по ёмкости и по времени нахождения в очереди | c >= s, v = 0, 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. | 6 |
| 22 | Диверсификация запросов по сроку предоставления ресурса с добавлением ресурсов | c = s = 1, v = 0, pr = 0, wt – переменная x > 0, tu – переменная y > 0, pbu = 0, pbi = 0  Запрос имеет срок T, не позднее которого он должен быть удовлетворён, иначе добавляется ресурс, способный удовлетворить этот запрос | 6 |

**Критерии оценки**

| 1. | Уровень сложности темы | - балл в таблице тем проектирования (реинжиниринга)  - с имитацией потока запросов | 3 – 9  +3 | |
| --- | --- | --- | --- | --- |
| 2. | Качество программного кода | * стиль (в т.ч.: имена, отступы и проч.) * оптимальность решения | +/- 1  балл | |
| 3. | Сбережение наследия | Минималистичность изменений базового варианта | +/- 1 балл | |
| 4. | Качество оформления проекта | Грамотность комментариев, корректность диаграмм, полнота выражения замысла, полнота тестирования | +/- 1 балл | |
|  | Итого | | | 0-15 | |

**Приложение №1**

К Лабораторной работе №1

**Регистратор ресурсов (обзор базового варианта как проекта)**

1. Постановка задачи регистрации ресурсов
   1. На пороге разработки

Дан конечный набор ресурсов, предоставляемых во временное пользование неким клиентам. Имеется также лицо, далее «клерк», который контролирует процесс занятия и освобождения ресурсов, неким образом взаимодействуя с клиентами и ресурсами.

Такая система встречается во множестве частных случаев, каковыми являются любая вокзальная камера хранения, библиотека, гостиница, система резервирования мест на пассажирском транспорте. Она входит как элемент в множество более сложных хозяйственных и технических систем. С другой стороны, её можно обобщить ещё дальше, допуская, например, что контролёром и, может быть, регулятором ресурсов является не лицо – человек, а некоторый неодушевлённый механизм.

Положим, что данная система изначально не содержит средств автоматизации, но является «доменом проектирования» – частью реального или воображаемого мира, в пределах которой ставится и затем решается задача компьютерной поддержки определённых протекающих в ней процессов. Разумеется, их нужно указать прежде, чем можно будет приступить к созданию поддержки. Пусть это будут процессы принятия решений, которые приходится выполнять клерку, когда он контролирует занятие и освобождение ресурсов. При этом, чтобы уклониться от разработки аппаратуры и ограничиться исключительно программистской разработкой, мы сознательно воздерживаемся от полной автоматизации системы, устранения клерка как лица.

* 1. Анализ и моделирование области приложения

Будем считать, что с этого момента мы приступаем к разработке означенной поддержки, хотя ясно, что ни предмет поддержки, ни её цели, ни требования к ней не определены достаточно чётко. Однако современные методологии разработки не только допускают, но полагают закономерным и обязательным уточнение задачи разработке в ходе проектирования, главным образом, на ранних его этапах. При этом погружение в процесс разработки по конкретной технологии способствует успешному уточнению задачи, поскольку обеспечивает разработчика конкретными языками, инструментами и правилами их использования.

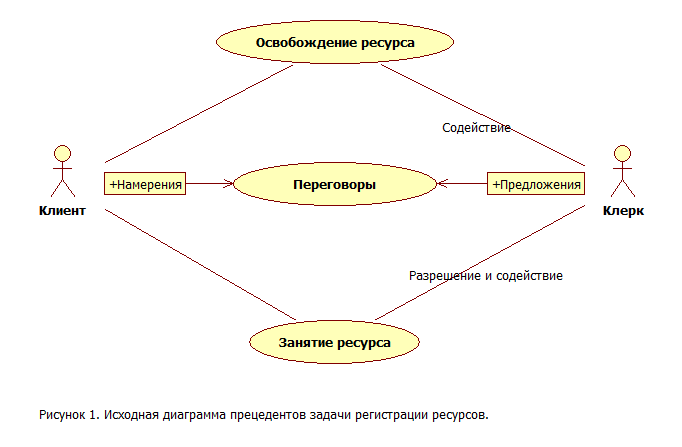
В данном пособии для проектирования отобрана технология (модель жизненного цикла программных продуктов) RUP, и по её правилам собственно проектированию программного продукта предшествует определение и исследование домена проектирования – среды, в которой этот продукт должен функционировать. В результате возникают концептуальная диаграммы классов, а также описания прецедентов в виде таблицы и соответствующей диаграммы. Эти документы («артефакты» разработки) выражают бизнес-модель домена и требования к разрабатываемому продукту.

К моменту начала разработки мы имеем её «идею», которая может быть лишена конкретных деталей, но так или иначе очерчивает домен проектирования, требования к разработке и цель разработки. Уже первая, исходная версия артефактов разработки несёт в себе определённую интерпретацию «идеи», детали, которые так или иначе уменьшают степень её общности. Эта версия не должна показывать проектные решения, относящиеся к разрабатываемому продукту и изменения, связанные с включением программного продукта в домен проектирования. Они появляются позже, в диаграмме системных взаимодействий и диаграмме (диаграммах) проектных классов, а также в связанных с ними артефактах.

Диаграмму системных взаимодействий и диаграмму проектных классов разрабатывают на основе описания прецедентов и концептуальной диаграммы классов, которая может уже в первой своей версии достигнуть детальности, достаточной для перехода к проектированию, или же первая версия подвергается ряду последовательных уточнений. Вторая возможность позволяет построить иерархию описаний доменов проектирования и обосновывать их выбор в терминах применяемой технологии.

В нашем случае Табл.1 описывает прецеденты. На Рис.1 это описание иллюстрировано исходной диаграммой прецедентов.

| Наименование прецедента | Переговоры |
| --- | --- |
| Исполнители | Клиент, Клерк |
| Предусловие | Система работает (предоставляет ресурсы). |
| Описание | **Основной сценарий (поток)**  Клиент высказывает запрос на использование ресурса.  Клерк сообщает клиенту, может ли быть его запрос выполнен и на каких условиях.  Между клиентом и клерком достигается соглашение об использовании ресурса. |
| Постусловие | Клиент может реализовать запрос |
| Описание | **Альтернативный сценарий (поток)**  Клиент высказывает запрос на использование ресурса.  Клерк сообщает клиенту, может ли быть его запрос выполнен и на каких условиях.  Клиент и клерк не достигают соглашения об использовании ресурса. |
| Постусловие | Клиент отзывает запрос. |
| Наименование прецедента | Занятие ресурса |
| Исполнители | Клиент, Клерк |
| Предусловия | Система работает (предоставляет ресурсы),  Между клиентом и клерком достигнуто соглашение об использовании ресурса. |
| Описание | **Основной сценарий (поток)**  Клиент вступает в использование ресурса. Клерк оказывает ему положенное содействие. |
| Постусловие | Клиент занимает запрошенный ресурс. |
| Наименование прецедента | Освобождение ресурса |
| Исполнители | Клиент, Клерк |
| Предусловие | Клиент, занимающий определённый ресурс, намерен его освободить. |
| Описание | **Основной сценарий (поток)**  Клиент освобождает ресурс. Клерк оказывает ему положенное содействие. |
| Постусловие | Занятый ресурс стал свободным. |

Таблица 1. Исходная таблица прецедентов задачи регистрации ресурсов.

Интерпретация исходной идеи в этих описаниях характеризуется следующими особенностями:

клиент во всех сценариях один,

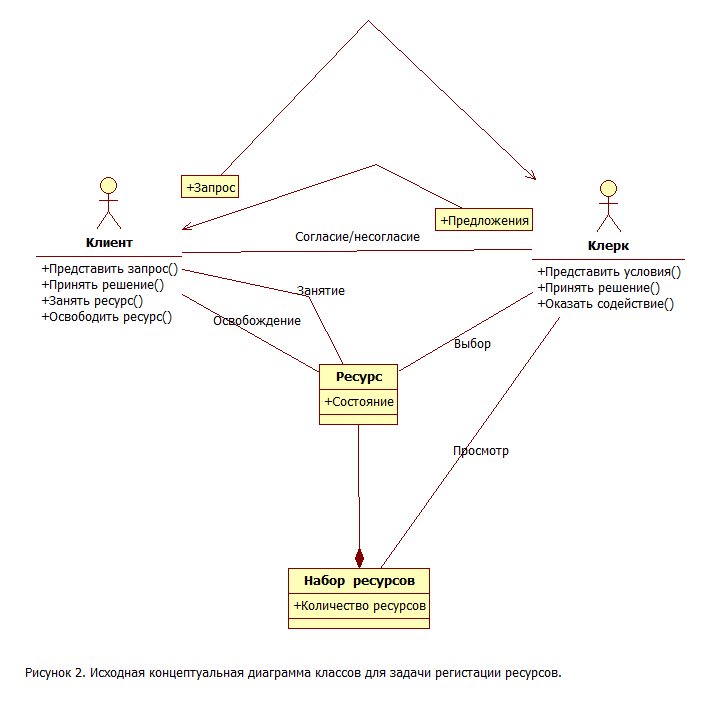
связь между клиентами различных прецедентов не предусмотрена,

ресурс освобождается только по инициативе клиента,

ресурсы запрашиваются, занимаются и освобождаются по одиночке,

содействие клерка клиенту осуществляется после принятия ими решения об использовании ресурса и с этим решением не связано.

Исходная диаграмм классов представлена на Рис.2.

На этой диаграмме показано, что ресурсы имеют лишь два атрибута – Состояние и Номер, причём состояние принимает значение “ресурс свободен” либо “ресурс занят”. Соответственно, клиент может запрашивать или ресурс с конкретным номером, или любой. Клерк, со своей стороны, может предложить клиенту на выбор все или некоторое подмножество свободных ресурсов по их номерам. То есть, указанная интерпретация сводит свободу решений клерка к тривиальному случаю: выбору ресурса из свободных в данный момент.

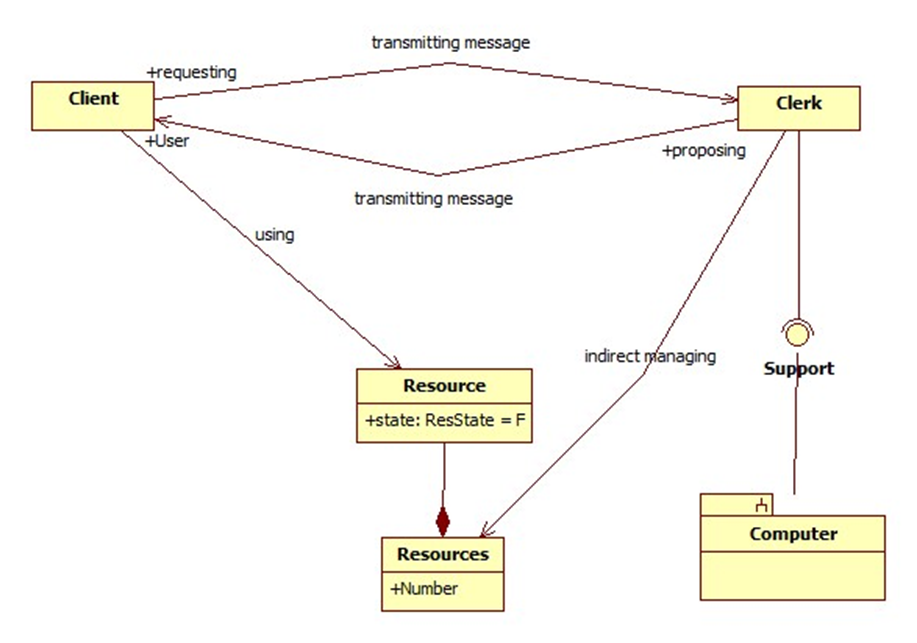
Выбор такой интерпретации оправдывается потенциальной простотой программного решения, и далее можно будет увидеть, что эта интерпретация не связывает развития проекта по многим направлениям.

* 1. Обоснование требований к решению задачи

В контексте приведённых выше артефактов разработки исходная идея (требование) компьютерной поддержки процессов принятия решений клерка приводит разработчиков к решению возложить на компьютер реализацию ассоциативной связи “Выбор”. То есть, компьютер должен указывать клерку, какой ресурс может быть предоставлен очередному клиенту. Если реализовать автоматическое взаимодействие компьютера с ресурсами, при котором компьютер без участия клерка получает информацию о занятости ресурсов, то связь “Просмотр” между клерком и набором ресурсов не нужна. Но если разработчики отказываются реализовать автоматическое взаимодействие компьютера с ресурсами, то компьютер должен получать нужную информацию о состоянии ресурсов от клерка или клиента. Если стремиться к минимальному изменению диаграммы классов, то остаётся принять решение о преобразовании ассоциации “Просмотр” между клерком и набором ресурсов в ассоциацию между клерком и компьютером с наименованием “Опрос и актуализация” и соответствующей семантикой.

В результате будет получена первая диаграмма проектных классов, показанная на Рис.3. Эта диаграмма должна пополняться и уточняться, подводя к разработке соответствующих программ на языках программирования. Поскольку они преимущественно англоязычны, уже начиная с первой диаграммы проектных классов словесные обозначения образованы на основе английского языка и записаны латиницей. Компьютер, аппаратура и программы, впервые появившиеся на диаграмме на Рис.3, представлены подсистемой System.

Рисунок 3. Начальная диаграмма классов проекта.

REQUEST – команда (без параметров), по которой программа выбирает первый свободный ресурс и выдаёт его номер, состояние ресурса не изменяется;

FREE – инициирует освобождение ресурса, номер освобождаемого ресурса указывается дополнительно;

OCCUPY – инициирует занятие ресурса, номер освобождаемого ресурса указывается дополнительно.

Соответствие между словарём Рис.2 и словарём Рис.3 показано в следующем перечне:

Клерк – clerk,

Клиент – customer,

Запрос – request,

Предложения – proposal,

Занятие – occupation,

Освобождение – releasing,

Согласие – agreement,

Несогласие – disagreement,

Ресурс – cell,

Набор ресурсов – cells,

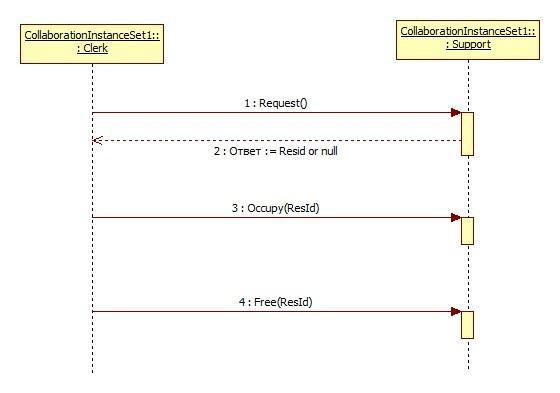
Состояние – state,

Количество ресурсов – cell number,

Выбор - selection,

Актуализация – updating.

Клерк 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 клерком более детально показано на диаграмме системных взаимодействий на Рис.4.

Рисунок 4. Начальная диаграмма системных взаимодействий

Диаграммы на Рис.3 и Рис.4 фактически представляют формулировку задачи регистрации ресурсов в нотации языка UML. Решение этой задачи состоит в последовательном уточнении и детализации этих диаграмм вплоть до получения текста прикладной программы на исходном языке программирования, реализующей решение задачи в среде, которая также должна быть точно специфицирована в ходе разработки.

Требования к дальнейшему проектированию и конструированию класса Program:

Выбор простейших решений,

Сценарии, выраженные последовательностями команд, независимы.

Приложение должно быть готово к исполнению сценариев от запуска до завершения.

Приложение не обязано работать в режиме “non stop”, но сеансы “работы” приложения, от запуска до завершения, не ограничиваются по длительности, равно как и количество запусков.

Преимущественно использовать средства, предоставляемые средой, а не воспроизводить их на низком уровне.

Приложение не требует каких-либо дополнительных средств или действия для подготовки к работе. В частности, внешнее хранилище данных создаётся самим приложением (в частности, не нужно отдельно создавать базу данных, но инсталлирование СУБД, конечно, нужно, если вариант приложения использует базу данных).

Из проведённого анализа и выполненных построений нетрудно заключить, что разрабатываемое приложение может быть отнесено к классу приложений, которым свойственна структура Модель – Вид – Контроллер (MVC). В соответствии с этим заключением делим приложение на следующие части:

диалоговая часть, реализующая интерфейс приложения и пользователя (соответствует “виду” структуры MVC;

модель, представляющая в приложении набор регистрируемых ресурсов;

контроллер, реализующий команды пользователя, анализируя и актуализируя состояние модели.

Такое разделение соответствует шаблону GRASP, подсказывающему декомпозицию приложения на фасадную часть, контроллер и информационный эксперт.

Нетрудно видеть, что модель вряд ли можно построить независимой от конкретных условий использования приложения. В частности, она будет зависеть от количества ресурсов. Этого соображения достаточно, чтобы ввести в структуру приложения часть, которая в диалоге с пользователем “выясняет” условия применения и выполняет необходимую настройку.

Уже сейчас, на предварительном уровне проектирования, можно выделить в модели часть, отвечающую за хранение состояния модели между запусками приложения.

Эти решения будут отражены на диаграммах UML в ходе проектирования.

1.4. Переход от постановки задачи к её решению

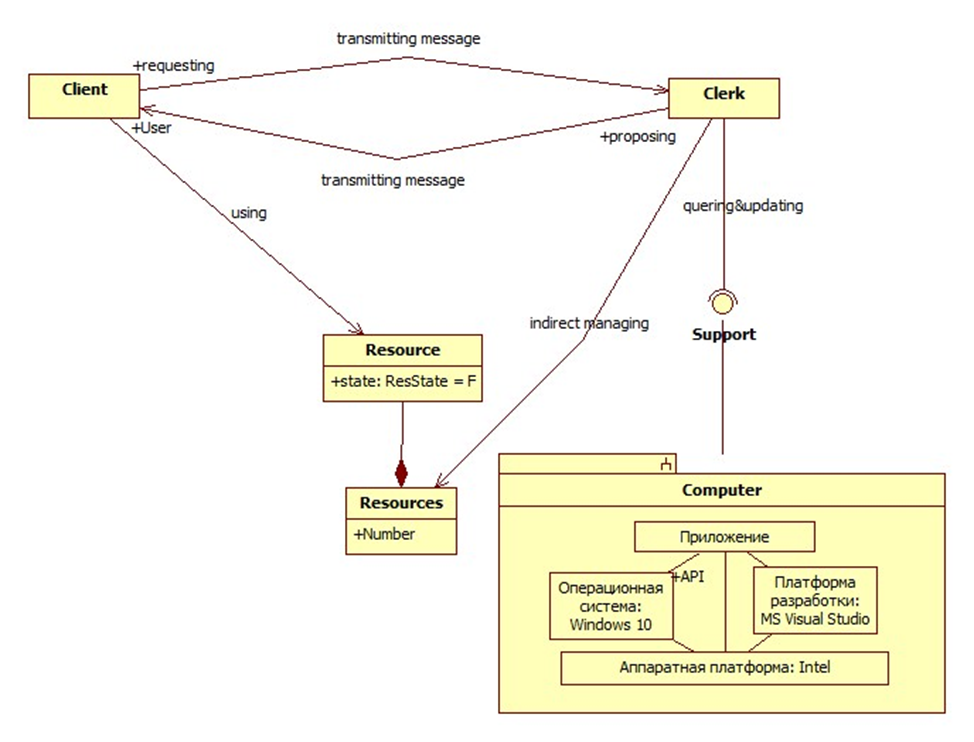
Проектирование складывается из последовательности решений.

Первые решения относятся к выбору компьютера и операционной системы. Очевидно, что решаемая задача не предъявляет никаких особых требований к производительности компьютера и его архитектуре. Задачу равно успешно можно решить в среде любой из общеизвестных операционных систем, применяемых на персональных компьютерах: Windows, Unix/Linux, Mac OS. Выбор разумно остановить на таком сочетании аппаратурной платформы и операционной системы, которое с большой вероятностью найдётся “под рукой” у читателя, то есть на сочетании персонального компьютера, оснащённого микропроцессором с архитектурой Intel Pentium или выше, и операционной системы Microsoft Windows уровня не ниже XP. В этой связи следует заметить, что, начиная с этого момента, читателю полезно все рассмотрения и построения данного пособия повторять на имеющемся в его распоряжении компьютере.

Следующее решение – выбор интегральной среды разработки (IDE). Предлагается остановиться на Microsoft Visual Studio, которая включает в себя ряд технологий, интересных для выполняемого проекта. Показанные далее примеры программ составлены и отлажены в Visual Studio издания 13, но, за единичными исключениями, они выполнимы и в Visual Studio издания не ниже 8. С не меньшим успехом можно было бы развивать проект в среде Delphi, однако новейшие версии этой IDE вряд ли можно считать доступными для читателя-студента.

Общий подход при принятии последующих решений: предпочтение отдаётся простейшим решениям, более сложные могут быть реализованы дополнительно.

Эти решения отражены на диаграмме классов, представленной на Рис.5. Она является развитием модели, показанной на Рис.4.

Рисунок 5. Диаграмма классов с первыми проектными решениями.

Следующее решение – выбор языка программирования и шаблона проекта. Простейший шаблон проекта – “консольный” – оставляет две возможности для выбора языка: C++ и C#. Далее он реализуется в виде “консольного приложения” на обоих языках, но в качестве основного принимается проект на языке C#, поскольку он позволяет, не меняя языка, перейти к более богатому возможностями проекту – проекту типа WindowsForms.

Протокол взаимодействия пользователя и программы определён на диаграмме последовательностей на Рис.4. Он должен реализовываться в течение всего промежутка времени от запуска программы до завершения её как приложения Windows. В ответ на введённую команду программа выдаёт сообщение. После запуска и после выдачи ответного сообщения программа должна быть готова к вводу команды.

Для обеспечения интерфейса между приложением и пользователем консольное приложение предоставляет консольное окно, открытое с момента запуска приложения до момента его завершения, а для ввода – клавиатуру. Пользователь вводит информацию с клавиатуры при наличии запроса в консольном окне приложения. При наличии запроса, пока ввод не закончен нажатием клавиши ENTER, приложение как процесс находится в состоянии ожидания.

Класс Console пространства имён System содержит методы, реализующие вывод из приложения в окно, ввод с клавиатуры и синхронизацию действий пользователя и приложения. Метод WriteLine выводит строку, метод ReadLine переводит запрос ввода на следующую строку и ждёт до тех пор, пока пользователь завершил ввод. При этом метод возвращает введённую информацию, позволяя приложению её обрабатывать.

Консольное приложение, представленное в шаблоне, содержит единственный модуль на языке C#, имеющий следующий вид:

using System;

namespace Cons2ReAr

{

class Program

{

static void Main(string[] args)

{

}

}

}

При запуске приложения первым получит управление метод Main. Соответственно, дальнейшие шаги по конструированию приложения можно планировать как формирование тела метода Main и, в случае необходимости, других компонентов класса Program.

Разработка класса Program должна обеспечить реализацию взаимодействий, показанных на Рис.4. В консольных проектах диалог между компьютером и работником (в нашем случае – клерком) организуется исключительно в виде обмена сообщениями, часть из которых может задерживаться на экране дисплея.

1. “Консольное” решение задачи регистрации ресурсов
   1. Организация диалога с пользователем

Методы класса Console устроены так, что готовность консольного приложения к вводу команды с клавиатуры может быть реализована вызовом метода ReadKey или ReadLine. Поскольку во многих случаях вводить понятнее более одной литеры, из этих двух методов будет использоваться ReadLine. Перед ней целесообразно выводить сообщение, напоминающее пользователю, какую информацию ему предлагается вводить.

Пусть вводимая команда присваивается переменной command. Тогда элемент интерфейса “ввод команды – получение ответного сообщения” начинается следующей парой предложений, помеченной для удобства ссылок на них:

L: Console.WriteLine("Введите команду:");

Command = Console.ReadLine();

Затем в зависимости от того, какая команда введена, выполняется та или иная обработка и выдаётся соответствующее сообщение. После этого, не теряя общности программы, можно вернуться к метке L. То есть, диалог реализуется циклом, который действует на протяжении всего сеанса работы приложения. Простейший его вариант может быть представлен так:

do

{

Console.WriteLine("Введите команду:");

Command = Console.ReadLine();

/\* анализ и выполнение команды \*/

}

while(true);

Для удобства пользователя расширим спецификацию, введя два новых положения:

- Если в качестве команды вводится пустая строка, то приложение завершается.

- Анализ и выполнение введённой команды не изменяют значения переменной Command.

Тогда фрагмент приложения, реализующий диалог с пользователем, будет выглядеть так:

string Command;

do

{

Console.WriteLine("Введите команду:");

Command = Console.ReadLine();

/\* анализ и выполнение команды \*/

}

while(Command != "");

Положим, что этот фрагмент помещён в тело Main класса Program, а команды исполняются методами того же класса, заголовок которых совпадает с заголовком команд на диаграмме Рис. 4.

Тогда приложение приобретёт следующий вид:

using System;

namespace Cons2ReAr

{

class Program

{

static string Request(){}

static void Occupy(string cn){}

static void Free(string cn){}

static void Main(string[] args)

{

string Command;

do

{

Console.WriteLine("Введите команду:");

Command = Console.ReadLine();

if (Command == "REQUEST") Console.WriteLine(Request());

if (Command == "OCCUPY")

{

Console.WriteLine("Введите номер ячейки:");

Occupy(Console.ReadLine());

Console.WriteLine("Ячейка стала занята.");

};

if (Command == "FREE")

{

Console.WriteLine("Введите номер ячейки:");

Free(Console.ReadLine());

Console.WriteLine("Ячейка освобождена.");

};

}

while(Command != "");

}

}

}

Этот текст успешно компилируется и запускается на выполнение, хотя тела методов Request, Occupy и Free пусты и при их вызове никакие действия не выполняются. Тем не менее, начиная с этого текста, можно проводить автономную отладку диалоговой части разрабатываемого приложения. Для того, чтобы считать эту часть полностью написанной, необходимо добавить операторы обработки особых случаев и соответствующие классы согласно спецификации. В результате будет получен следующий текст;

using System;

namespace ResRegV1cons

{

class ResAreBusy : Exception { }

class ResIdInvalid : Exception { }

class UnRecommended : Exception { }

class ResIsBusy : Exception { }

class ResWasFree : Exception { }

class Program

{

static string Request(){}

static void Occupy(string cn){}

static void Free(string cn){}

static void Main(string[] args)

{

string Command;

do

{

Console.WriteLine("Введите команду:");

Command = Console.ReadLine();

Command = Command.ToUpper();

try

{

if (Command == "REQUEST") Console.WriteLine(Request());

if (Command == "OCCUPY")

{

Console.WriteLine("Введите номер ячейки:");

Occupy(Console.ReadLine());

Console.WriteLine("Ячейка стала занята.");

};

if (Command == "FREE")

{

Console.WriteLine("Введите номер ячейки:");

Free(Console.ReadLine());

Console.WriteLine("Ячейка освобождена.");

};

}

catch (OverflowException) { Console.WriteLine("Такой ячейки нет."); }

catch (FormatException) { Console.WriteLine("Такой ячейки нет."); }

catch (ResIdInvalid) { Console.WriteLine("Такой ячейки нет."); }

catch (ResWasFree) { Console.WriteLine("Ячейка была свободна."); }

catch (ResAreBusy) { Console.WriteLine("Все ячейки заняты."); }

catch (ResIsBusy) { Console.WriteLine("Ячейка уже занята."); }

}

while(Command != "");

}

}

Тело диалогового цикла взято погружено в try-блок, и особые случаи, распознаваемые при выполнении команд, будут обрабатываться в операторах catch. Обработка состоит в выдаче соответствующего сообщения, после чего приложение продолжит функционировать.

Попутно спецификация расширена разрешением вводить команды как строчными, так и заглавными буквами в любом сочетании. Для этого в тело диалогового цикла перед анализом команды введено предложение, переводящее введённую строку в верхний регистр (заглавные буквы).

Несистемные особые случаи должны возбуждаться операторами **throw** в методах, исполняющих команды.

Выполненное построение можно расценивать как первую итерацию в ходе выполнения данного проекта.

Вместо использования механизма try … catch для обработки несистемных особых случаев можно было бы использовать возвращаемый параметр методов обработки команд, и на некоторые обусловленные значения реагировать в теле диалогового цикла выдачей соответствующего сообщения. Однако этот вариант отклоняется как решение более низкого уровня, которое, к тому же, заметно усложняет текст диалоговой части.

Анализ команды реализуется последовательным сравнением введённой строки с эталонными названиями команд. При совпадении вызываются соответствующие методы. Для обеспечения отладки диалога эти, пока ещё не разработанные методы заменяются отладочными.

if (Command == "REQUEST") Console.WriteLine(Request());

if (Command == "OCCUPY")

{

Console.WriteLine("Введите номер ячейки:");

Occupy(Console.ReadLine());

};

if (Command == "FREE")

{

Console.WriteLine("Введите номер ячейки:");

Free(Console.ReadLine());

};

Диаграмма последовательностей на Рис.6 описывает диалог между классом Program и классом Console.

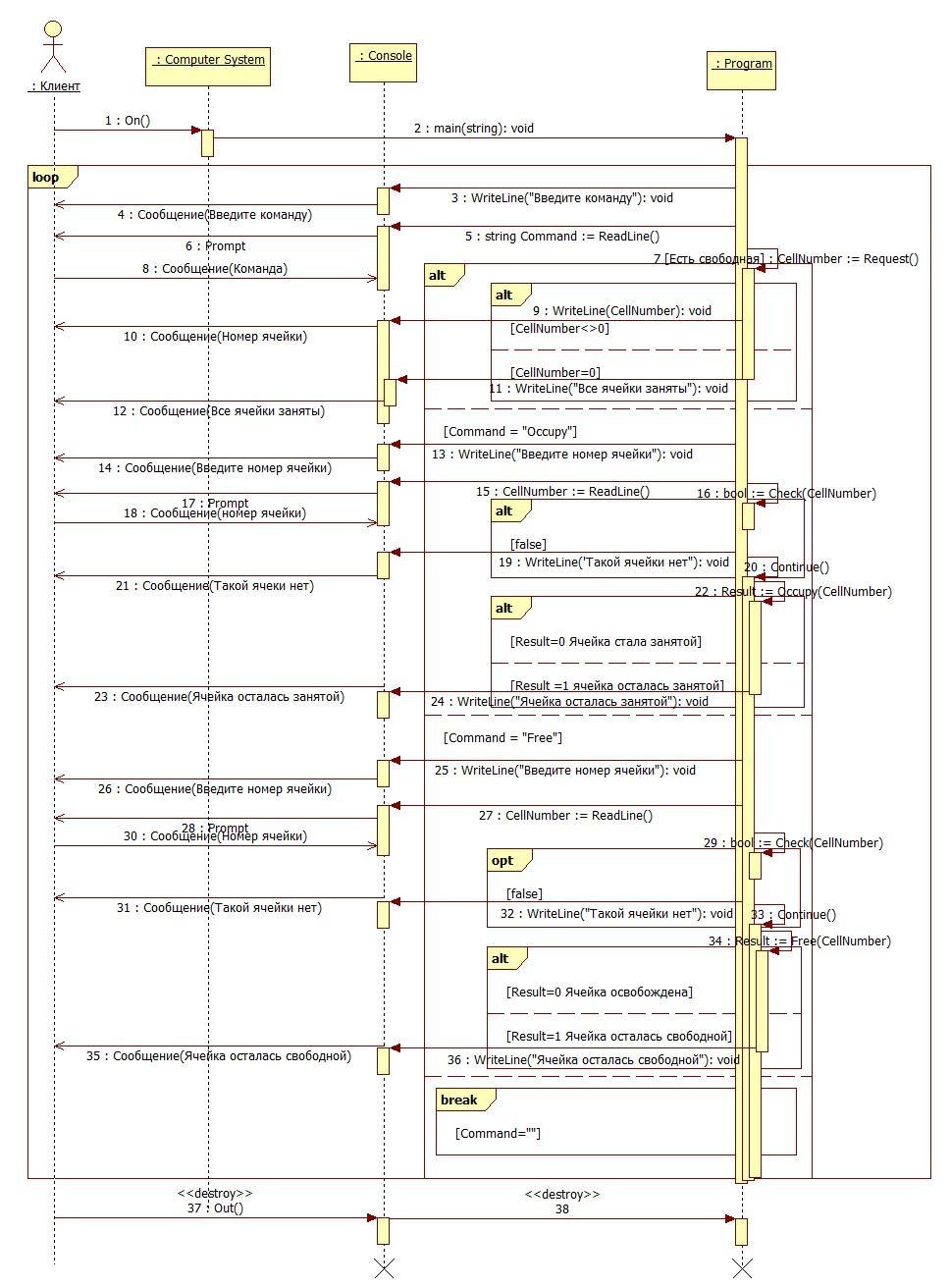


Рисунок 6. Диаграмма последовательностей консольного диалога.

Все принятые решения позволяют трансформировать диаграмму классов проекта, как показано на Рис.7. На трансформированной диаграмме показан также клерк (класс clerk), остальные классы и их связи, которые находились на диаграмме классов на Рис.3, далее показываться не будут, поскольку разработка их не затрагивает.

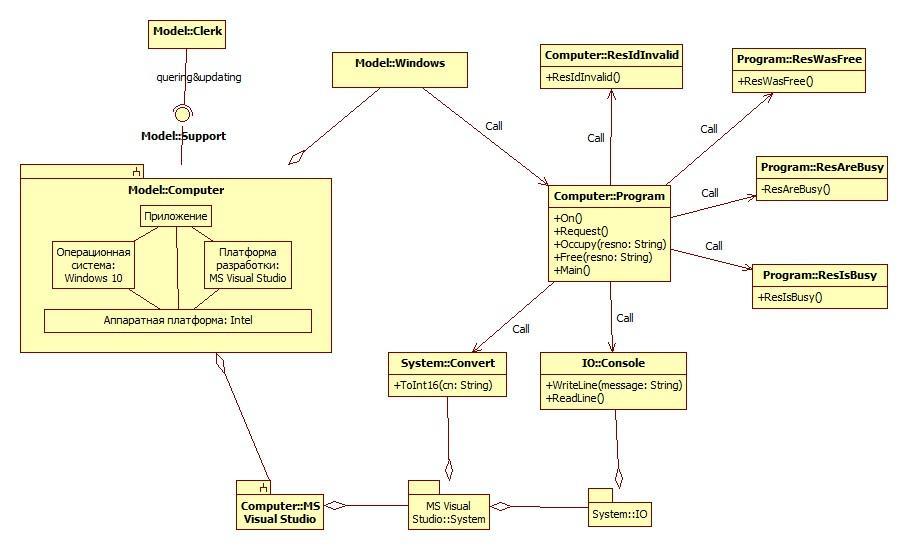


Рисунок 7. Диаграмма классов с обеспечением консольного диалога.

* 1. Реализация бизнес-логики и выбор модели данных

Реализация бизнес-логики и выбор модели данных оказываются тесно связанными. Согласно спецификации набор ресурсов моделируется множеством объектов – моделями ресурсов. Команды Occupy и Free изменяют состояние ресурса, указанного параметром команды, а команда Request – выдаёт один из “свободных” ресурсов, выбранных по некоторому правилу, которое спецификацией не установлено.

Считаем очевидным, что все три команды могут быть реализованы перебором множества моделей ресурсов, причём команды Occupy и Free выбирают ресурс, сравнивая его идентификатор со своим входным параметром, а Request выбирает, например, первый попавшийся “свободный”. Перебор в командах Occupy и Free может быть заменён прямым доступом, если он обеспечен на множестве моделей ресурсов. Эти рассуждения выявляют два независящих от платформы подхода к реализации бизнес-логики разрабатываемого приложения, дальнейшая конкретизация которых требует рассмотрения возможностей выбранной платформы и среды проектирования.

В Microsoft Visual Studio прямой доступ по индексу или набору индексов обеспечен для встроенного в C# массива и, соответственно, класса Array в пространстве имён System. Это лишь частное представление множества, частное, прежде всего в двух отношениях:

* ёмкость ограничивается при инициировании,
* доступ к элементу по номеру.

Ресурсы пространства имён System.Collections.Generic снимают первое из указанных ограничений и ослабляют второе.

Ресурсы пространствах имён System.Collections.Generic обеспечивают в программе сущности, соответствующие множествам как в общеязыковом, неформальном понимании этого слова, так и в строгом математическом.

Интерфейс ICollection<T> можно расценивать как центральный по универсальности и функциональной наполненности.

Методы Add, Remove и Contains c параметром (Object value), а также Clear() обеспечивают чисто теоретико-множественные операции с множеством (“коллекцией”). Мощность (количество элементов) выражается значением свойства Count.

Перебор коллекции обеспечивается методом GetEnumerator, унаследованным от интерфейса IEnumerable<T> и реализующим интерфейс IEnumerator<T>.

Реализации интерфейса ICollection<T> дополняют его и специализируют. Основные реализации:

классом LinkedList<T>,

классом HashSet<T>, причём этот класс реализует не только непосредственно ICollection<T>, а IHashSet<T>, потомок ICollection<T>;

классом List<T>, причём этот класс реализует не только непосредственно ICollection<T>, а IList<T >, потомок ICollection<T>. Интерфейс IList<T> обеспечивает индексирование методами IndexOf, Insert и RemoveAt, а также свойством Item. Класс List<T> содержит также методы, обеспечивающие поиск по заданному предикату и сортировку.

Интерфейс ICollection<T> наследует интерфейс IEnumerable<T> (и IEnumerable), представляющий методы, которые обеспечивают функциональность, похожую на функциональность оператора Select языка SQL. Эти методы (более 130 методов) реализуются в статическом классе Enumerable, находящемся в пространстве имён System.Linq. Эти методы вызываются в порядке расширения интерфейса ICollection<T> и других.

Для представления можества можно также использовать класс Stack.

Приведённы обзор обосновывает в качестве простейшей следующую реализацию модели в виде массива:

static string[] vRes\_s;//Модель набора ресурсов

и команд Occupy и Free:

static void Occupy(string cn)//cn –идентификатор (номер) ресурса

{

vRes\_s[Convert.ToInt16(cn) - 1] = "B";//Ресурс становится “свободным”

}

static void Free(string cn) //cn – идентификатор (номер) ресурса

{

vRes\_s[Convert.ToInt16(cn) - 1] = "F";//Ресурс становится “занятым”

}

Спецификация уточняется: идентификатором ресурса служит его номер от 1. Поэтому значение входного параметра уменьшается на единицу, поскольку элементы массива нумеруются с 0.

Ресурс моделируется литерой (“B” или “F”), а идентификатор, определённый в спецификации как поле, в модели представлен неявно – номером элемента массива. Таким образом реализация оказывается проще спецификации.

Реализация команды Request возвращает номер первого свободного элемента массива, попавшегося при его переборе от начала.

static string Request()

{

for(int i=0;i<vRes\_s.Length;i++)

{

if(vRes\_s[i] == "F") return Convert.ToString(i + 1);

}

}

Включение в эти методы распознавания особых случаев позволяет получить их окончательный вид:

static void Occupy(string cn)

{

if ((Convert.ToInt16(cn) > vRes\_s.Length) | (Convert.ToInt16(cn) < 0)) throw new ResIdInvalid();

if(vRes\_s[Convert.ToInt16(cn) - 1] == "B")throw new ResIsBusy();

vRes\_s[Convert.ToInt16(cn) - 1] = "B";

}

static void Free(string cn)

{

if ((Convert.ToInt16(cn) > vRes\_s.Length) | (Convert.ToInt16(cn) < 0)) throw new ResIdInvalid();

if (vRes\_s[Convert.ToInt16(cn) - 1] == "F") throw new ResWasFree();

vRes\_s[Convert.ToInt16(cn) - 1] = "F";

}

static string Request()

{

for(int i=0;i<vRes\_s.Length;i++)

{

if(vRes\_s[i] == "F") return Convert.ToString(i + 1);

}

throw new ResAreBusy(); ;

}

* 1. Сохранение модели ресурсов

Выбор способа сохранения модели ресурсов.

Модель ресурсов сохраняется как последовательный файл методом WriteAllLines и восстанавливается методом ReadAllLines класса File пространства имён IO. Причём метод WriteAllLines превращает каждый строковый элемент массива в строку файла, а метод ReadAllLines – строку файла в строковый элемент массива.

*Почему?* Вызов этих методов является простейшим способом (нужен всего по одному их вызову) пересылки массива строк в файл и обратно. Иные способы, например, использование потоков для сохранения в файле или обращение к базе данных, требуют более сложных программ.

Выбор файла во время работы приложения.

Предлагаются два варианта: а) файл со стандартным именем в директории, в которой находится исполняющийся модуль приложения, и б) пользователь указывает имя файла (включающее в себя полный путь к файлу).

Имя директории, содержащей исполняемый модуль приложения, может быть найдено методом GetCurrentDirectory. Находящимся в пространстве имён IO.

*Почему?* Вариант (а) предусмотрен спецификацией. Вариант (б) как дополнительный увеличивает степень дружественности приложения, позволяет использовать несколько фалов, в которых сохраняется модель.

Когда пользователь может выбирать файл для хранения модели.

Один раз при запуске приложения.

*Почему?* Чтобы не усложнять основной протокол взаимодействия пользователя с приложением.

Возможности воздействия пользователя на файл, предназначенный для хранения модели.

При выборе файла или не изменять файл или “переустановить модель”, то есть указать количество ресурсов и установить их в модели в свободное состояние.

*Почему?* Обеспечиваются возможности: а) продолжить обслуживание ресурсов после перезапуска приложения и б) установить естественное (свободное) начальное состояние набора ресурсов.

Резюме: модель сохраняется предложением программы:

File.WriteAllLines(Path, vRes\_s);

и восстанавливается предложением:

vRes\_s = File.ReadAllLines(Path);

Path – переменная, содержащая полный путь к файлу. Путь к стандартному файлу, пусть его имя будет Resmod00.txt, может быть получено выполнением предложения:

Path = Directory.GetCurrentDirectory() + @"\Resmod00";

Если файл при записи отсутствует, то он создаётся.

Интеграция бизнес-логики и сохранения ресурсов.

Позволит провести автономную отладку разрабатываемого приложения.

Включение бизнес-логики и модели в приложение.

Поле vRres\_s (модель ресурсов), а также методы Request, Occupy и Free образуют новый статическй класс Model. Временные определения этих методов из класса Program удаляются. Методы объявляются public для обеспечения их доступности их других классов. Поле vRres\_s инициировано.

Выделение класса Model продиктовано принципиальными соображениями

Обеспечение сохранения модели.

В начало цикла вводится предложение, сохраняющее модель в файле по адресу Path.

В класс Program вводится определение переменной Path, а в начало метода Main вводится предложение, которое присваивает этой переменной значение – путь к файлу, предназначенному для сохранения модели.

Расположение определения переменной Path и присвоения ей значения временно, служит целям отладки. Так же временно реализовано инициирование модели.

Сохранение модели ресурсов в начале каждого диалогового цикла позволяет сохранять модель после каждой выполненной команды и проверять возможность сохранения перед началом работы. Иначе можно было бы ввести специальную команду сохранения, но это усложнило бы пользовательский интерфейс приложения.

Резюме:

Модуль ResRegV1cons приложения приобретает следующий вид:

using System;

using System.IO;

namespace ResRegV1cons

{

class ResAreBusy : Exception { }

class ResIdInvalid : Exception { }

class UnRecommended : Exception { }

class ResIsBusy : Exception { }

class ReswasFree : Exception { }

static class Model

{

static string[] vRes\_s = new[] { "1", "2", "3" };//Модель набора ресурсов

public static void Occupy(string cn)

{

if ((Convert.ToInt16(cn) > vRes\_s.Length) | (Convert.ToInt16(cn) < 0)) throw new CellIdInvalid();

if(vRes\_s[Convert.ToInt16(cn) - 1] == "B")throw new CellIsBusy();

vRes\_s[Convert.ToInt16(cn) - 1] = "B";

}

public static void Free(string cn)

{

if ((Convert.ToInt16(cn) > vRes\_s.Length) | (Convert.ToInt16(cn) < 0)) throw new CellIdInvalid();

if (vRes\_s[Convert.ToInt16(cn) - 1] == "F") throw new CellwasFree();

vRes\_s[Convert.ToInt16(cn) - 1] = "F";

}

public static string Request()

{

for(int i=0;i<vRes\_s.Length;i++)

{

if(vRes\_s[i] == "F") return Convert.ToString(i + 1);

}

throw new CellsAreBusy(); ;

}

}

class Program

{

// static string Request(){}

// static void Occupy(string cn){}

// static void Free(string cn){}

static string Path; //путь к файлу, сохраняющему модель

static void Main(string[] args)

{

string Command;

Path = Directory.GetCurrentDirectory() + @"\Resmod00";

do

{

File.WriteAllLines(Path, Model.vRes\_s);//сохранение модели

Console.WriteLine("Введите команду:");

Command = Console.ReadLine();

Command = Command.ToUpper();

try

{

if (Command == "REQUEST") Console.WriteLine(Model.Request());

if (Command == "OCCUPY")

{

Console.WriteLine("Введите номер ресурса:");

Model.Occupy(Console.ReadLine());

Console.WriteLine("Ресурс стал занятым.");

};

if (Command == "FREE")

{

Console.WriteLine("Введите номер ресурса:");

Model.Free(Console.ReadLine());

Console.WriteLine("Ресурс освобождён.");

};

}

catch (OverflowException) { Console.WriteLine("Такого ресурса нет."); }

catch (FormatException) { Console.WriteLine("Такого ресурса нет."); }

catch (ResIdInvalid) { Console.WriteLine("Такого ресурса нет."); }

catch (ResWasFree) { Console.WriteLine("Ресурс был свободен."); }

catch (ResAreBusy) { Console.WriteLine("Все ресурсы заняты."); }

catch (ResIsBusy) { Console.WriteLine("ресурс уже занят."); }

}

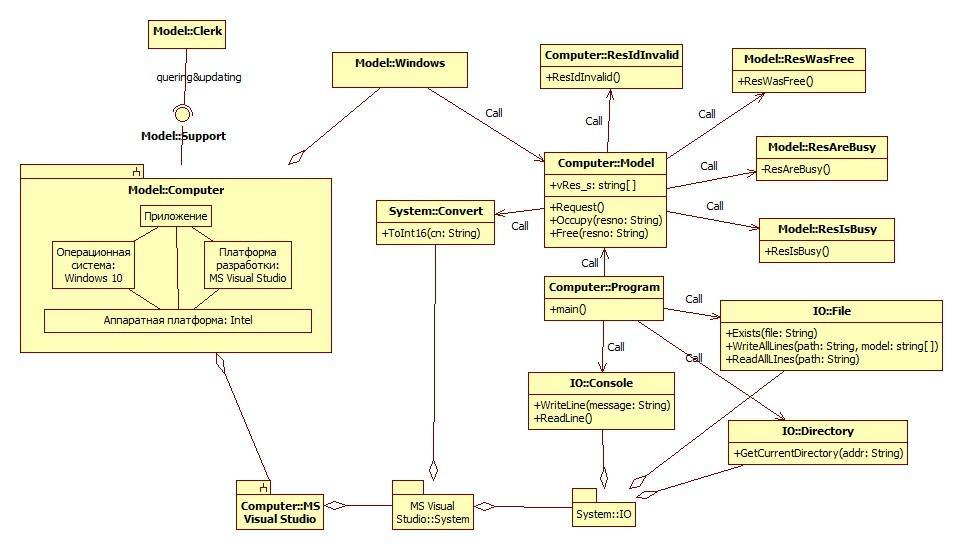
while(Command != "");

}

}

}

Соответственно, в диаграмму проектных классов добавляются классы ввода-вывода с используемыми методами, а в класс Pogram – поле, представляющее модель ресурсов. Актульное состояние диаграммы классов показано на Рис. 7.

Рисунок 7. Диаграмма классов с выбором и сохранением модели данных.

* 1. Начальные действия после запуска программы

Организуется диалог с пользователем, в ходе которого принимаются следующие решения:

- какой файл будет использоваться для сохранения модели ресурсов,

- каково начальное состояние модели (обновлённое (все ресурсы “свободны”) или сохранённое в файле),

- если модель обновляется, то каково количество ресурсов.

Начальные действия определяются принятыми ранее решениями и произведёнными построениями.

Обновление модели.

Статический метод ClearModel

static void ClearModel()

{

Console.WriteLine("Укажите количество ресурсов:");

try

{

int Capacity = Convert.ToInt32(Console.ReadLine());

Model.vRes\_s = new string[Capacity];

for (int i = 0; i < Capacity; i++) Model.vRes\_s [i] = "F";

}

catch

{

Console.WriteLine("Введено некорректное число!");

ClearModel();

}

}

В случае некорректного ввода числа ресурсов запрос к пользователю повторяется.

Восстановление модели.

Статический метод GetModel запрашивает пользователя, нужно ли обновить, и если “да”, то вызывает метод ClearModel. В противном случае восстанавливает модель из файла.

static void GetModel()

{

Console.WriteLine("Обновить файл?");

if (Console.ReadLine().ToUpper() == "Y") ClearModel();

else

{

Model.vRes\_s = File.ReadAllLines(Path);

}

}

Диалог выбора файла для сохранения модели.

Статический метод On проверяет наличие стандартного файла для сохранения модели и, если он имеется, то пользователю направляется запрос, согласен ли тот его использовать. В случае положительного ответа вызывается метод GetModel. Иначе пользователю направляется запрос, желает ли он в данном запуске приложения работать со стандартным файлом. В случае положительного ответа вызывается модуль ClearModel.

Если пользователь не намерен работать со стандартным файлом, ему предлагается ввести адрес нестандартного файла. Если указанный файл существует, то вызывается метод GetModel. В противном случае вызывается.

Метод ClearModel не создаёт файла, но инициирует модель в оперативной памяти, затем её инициированное состояние будет сохранено перед запросом команды в методе Main класса Program. Таким образом работоспособность выбранного стандартного или нестандартного файла будет проверена. Если будет обнаружено, что в файл невозможно писать, то пользователю получит соответствующее сообщение и ему придётся перезапустить приложение. Если указанный нестандартный файл не существует, то метод On возвращает значение false, чтобы модуль On мог быть перезапущен.

static bool On()

{

try

{

if (File.Exists(Directory.GetCurrentDirectory() + @"\Resmod00"))

{

Console.WriteLine("Использовать существующий стандартный файл Resmod00?");

if (Console.ReadLine().ToUpper() == "Y")

{

Path = Directory.GetCurrentDirectory() + @"\Resmod00";

GetModel();

return true;

}

}

else

{

Console.WriteLine("Создать стандартный файл?");

if (Console.ReadLine().ToUpper() == "Y")

{

Path = Directory.GetCurrentDirectory() + @"\Resmod00";

ClearModel();

return true;

}

};

Console.WriteLine("Введите полный адрес нестандартного файла:");

Path = Console.ReadLine();

if (File.Exists(Path))

{

GetModel();

return true;

}

else

{

ClearModel();

return true;

}

}

catch (IOException) { Console.WriteLine("Файл не открылся."); return false; }

catch (Exception) { Console.WriteLine("Ошибка ввода-вывода."); return false; }

}

* 1. Интеграция и тестирование варианта V1 приложения.

Интеграция.

В модуль ResRegV1.cs вводится новый статический класс SetUp, включающий методыClearModel, GetModel. Публичный метод On вызывается в начале метода Main класса Program в цикле, который заканчивает, если On возвращает значение true.

Поле Path удаляется из класса Program и становится полем класса SetUp. Инициирование поля Path из метода Main удаляется. При обращении к полю Path в методе On необходимо уточнить, что Setup следует брать из класса SetUp, а не из пространства имён System.IO, в котором оно обозначает тип.

Образование отдельного класса для выполнения начальных действий при запуске приложения обусловлено соблюдением шаблонов GRASP.

На Рис.8 приведена актуальная диаграмма классов приложения.

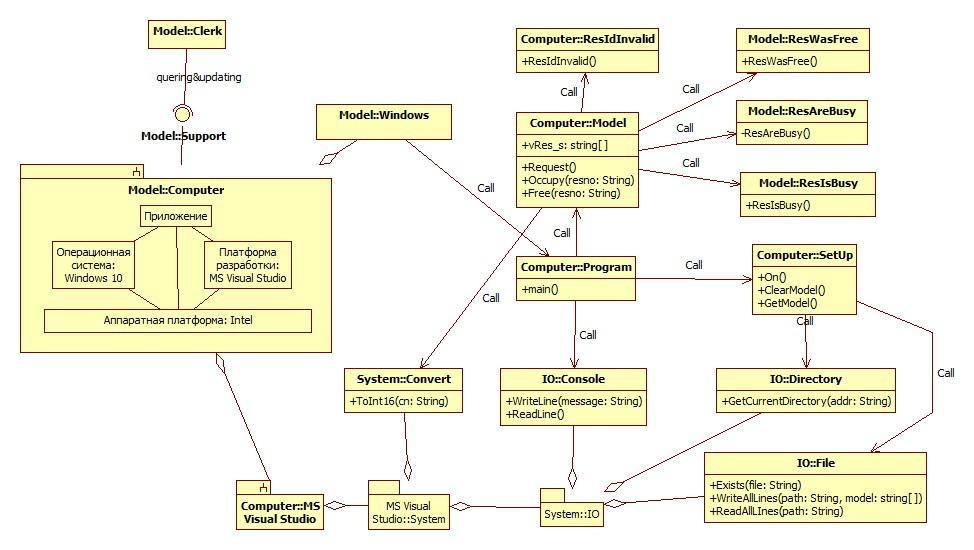


Рисунок 8. Финальная диаграмма классов первой версии Регистратора ресурсов.

Текст модуля и протокол его тестирования приведён в Приложении.

Тестирование.

Тестирование протоколируется «Средством записи действий» операционной системы.

План тестирования: